University of Zurich[UZH]

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

Mitigating Cyberattacks Affecting Resource-constrained Devices Through Moving Target Defense (MTD) Mechanisms

Jordan Cedeño
Zürich, Switzerland
Student ID: 14-713-440

Supervisor: Alberto Huertas, Jan von der Assen
Date of Submission: April 11, 2022

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

ifi

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

Most Internet of Things (IoT) devices such as radio spectrum sensors are not designed and built with security in mind. The static nature of such IoT devices coupled with the resource constrains under which they operate, makes such devices a lucrative target for cyberattacks. One option when it comes to dealing with such cyberattacks is employing Moving Target Defense (MTD) in which some system parameters are "moved" in order to disrupt an ongoing attack. This thesis aims to propose, design and implement a prototypical lightweight MTD based framework (MTD Framework) for Linux based IoT devices such as radio spectrum sensors, which is capable of deploying host-based MTD security solutions (MTD Solutions) based on reported attacks/events from an external program monitoring for attacks/events. Furthermore, this thesis implements a total of four MTD based security solutions to deal with the following three malware families once they have already infected the system: command & control based malware, crypto ransomware, user-level rootkits (using preloads).
To test the effectiveness of the MTD Framework and the MTD Solutions they were tested against real malware to see how they perform. Additionally some performance data is gathered to present the additional resource consumption that the MTD Framework incurs. The results are promising and suggest that the MTD Framework combined with the MTD Solutions proposed and implemented in this thesis work well as an additional security layer which is capable of disrupting/disabling running malware of the above mentioned malware families.

Acknowledgments

The assistance provided by Alberto Huertas and Jan von der Assen was greatly appreciated. As supervisors of this thesis they were present during the entire research process, design process as well as the writing process. Their continuous feedback and input at every step was crucial for this thesis and is therefore greatly appreciated.

Contents

Chapter 1

Introduction

1.1 Motivation

In today's society the use cases for IoT (Internet of Things) devices are ubiquitous and varied. Therefore, it is no surprise that the number of IoT devices in use is in the billions as of the year 2021 and is projected to keep rising [1]. These devices have become essential in today's society by contributing to agriculture, infrastructure management, industrial automation, home automation and more [2]. Given that many IoT devices communicate via wireless connections, this can put a lot of strain on some of the available radio frequencies [3]. Some radio frequency bands can therefore become congested/over crowded, while others remain underused [3]. Such congestions then in turn lead to degrading service quality, because retransmissions become necessary, which reduces the effective data throughput [3].
This is where radio spectrum sensors come in handy. These sensors make it possible to analyze the radio frequency spectrum and allow Cognitive Radio Networks (CRN) to assign IoT devices to underpopulated frequency bands as a form of load balancing [3]. Especially helpful for CRNs are platforms such as ElectroSense which are capable of providing regional spectrum data [3]. ElectroSense gathers its data by way of crowdsensing which allows peers (spectrum sensors) registered to the platform to collect spectrum data across wide geographical areas. ElectroSense's stated goal is: "to sense the entire spectrum in populated regions of the world and to make the data available in real-time for different kinds of stakeholders which require a deeper knowledge of the actual spectrum usage" [4].
For these CRNs to be able to dynamically assign IoT devices to underpopulated radio frequency bands effectively, it is necessary that the sensed radio frequency spectrum data is accurate and not manipulated by any malicious actors [3]. Such manipulation is even more likely in a crowdsensing setting where resource constrained IoT devices such as a Rasperry Pi are employed as spectrum sensors [3], [5]. Being connected to the internet is integral for these spectrum sensors. Just as with other use cases of IoT devices, spectrum sensors rely on communication and therefore on an active Internet connection for them to work properly. But being exposed to the Internet also brings its fair share of downsides and challenges with it. IoT devices are by no means immune to cyberattacks and often times do present an easier and also more lucrative target than conventional computer

systems, because of their static and resource constrained nature [2], [5], [6].
This is where MTD (Moving Target Defense) as a paradigm comes in as a promising solution. Instead of addressing specific exploits as is usually done by conventional security measures to prevent attacks, MTD does not actually try to prevent attacks from happening, but rather aims to disrupt attacks, such that their execution is not successful [6], [7]. This is usually achieved by changing properties, data or configurations of the system in a dynamic way, such that the likelihood of the attacks being successful is diminished [6], [8]. The problem is, that given the resource constraints found in many IoT devices such as the aforementioned spectrum sensors, already existing or proposed MTD based solutions and approaches are often not applicable directly for IoT devices [6]. This thesis aims to propose and implement an MTD framework, which can be deployed on an ElectroSense spectrum sensor.

1.2 Description of Work

This thesis proposes and implements a lightweight framework for a spectrum sensor (Raspberry Pi 4, running Raspberry OS with 4GB of Memory and less than 2GB usable disk space), capable of deploying MTD based security solutions in reaction to an ongoing attack (or another event). This framework consists of three main components: the **MTD Deployer Server**, the **MTD Deployer Client** and the **MTD Solutions**. The MTD Deployer Server is meant to be run permanently as a service on the Raspberry Pi. It listens for attack reports which can be send via the MTD Deployer Client. Once an attack report has been sent, the MTD Deployer Server deploys an MTD based security solution (MTD Solution) for the type of occuring attack. This MTD framework is meant to be expandable and configurable, meaning that new MTD based security solutions (MTD Solutions) can easily be added and parameters with which the MTD based security solutions are run, can be easily adjusted.
Aside from the MTD Framework itself this thesis proposes and implements MTD based security solutions to deal with the following three types of malware families: **command & control** based malware, **crypto ransomware** and **user-level rootkits** (using preloading).
The MTD Solution against command & control based malware works by dynamically migrating the Raspberry Pi to a new private IP address after the malware has established a connection to its command & control server. This approach is capable of disrupting the established connection.
As part of this thesis two MTD Solutions against crypto ransomware are proposed. One focuses on keeping predefined data safe from encryption, by changing its file extensions while the other focuses on trapping the encrypting ransomware with dummy data such that it gets stuck encrypting worthless data instead of valuable system data.
For user-level rootkits a MTD Solution is proposed which is capable of removing rootkits by preloading legitimate C libraries such that any rootkit behaviour which would normally hinder the removal of the user-level rootkit is temporarily disabled. During this time frame the MTD Solution cleans up the system such that it is restored to its previous state.
Lastly this thesis evaluates the real world applicability of the MTD Framework and the

implemented MTD Solutions by testing its effectiveness against real malware. Additionally some resource consumption data is gathered as well to paint a better picture.

1.3 Thesis Outline

In a first step this thesis provides context as to what Moving Target Defense is, how it works and what types of Moving Target Defense based approaches exist. In a second step the thesis presents the reader with existing Moving Target Defense approaches in the IoT field specifically. Next the thesis proposes a framework which can be deployed on a Raspberry Pi 4 acting as an ElectroSense spectrum sensor and then moves on to discuss its design and implementation. Aside from the framework itself the thesis then also presents a few exemplary MTD based security solutions for the above mentioned types of malware. Then the implemented framework and the MTD based security solutions are evaluated for their effectiveness. In the last section, this thesis provides an interpretation of the data gathered during the evaluation step. Then it gives a short overview of the limitations of this thesis. Lastly it discusses potential future research opportunities and then it presents its conclusion.

Chapter 2

Background

This section will provide a short overview about the main concepts of Moving Target Defense. Then it will discuss the scenario (ElectroSense, spectrum sensors) for which the MTD Framework was built and lastly it will introduce the three malware families for which MTD based security solutions will be implemented.

2.1 Moving Target Defense

One of the biggest issues, when it comes to cybersecurity is the asymmetric nature between the resources of the attacker and the resources of the defense [9]. The issue arrives from the fact, that attackers usually have a time advantage and are capable of analyzing their target for possible vulnerabilities before they launch an actual attack [9]. Furthermore, once one or more suitable vulnerabilities have been found, the attacker often times can keep launching attacks ad nauseam, as long as the vulnerability through which the attack is launched persists[9]. This means, that if a system needs to be secured, the defenders have to keep looking for potential vulnerabilities and patch them before they are abused as an attack avenue [9]. This approach is also ineffective, when one considers that there exist software and hardware vulnerabilities that are either not easily patchable or in the worst case not patchable at all, which would leave the system in a perpetually insecure state.
This is where Moving Target Defense as a paradigm comes in. Moving Target Defense is a security approach that admits that perfect security is not achievable for a computer system [7]. Instead of trying to prevent attacks from happening or focusing on fixing vulnerabilities that enable attacks in the first place, Moving Target Defense tries to reverse the above mentioned asymmetry and tries achieve security through diversification or obfuscation [9], [10].The goal is not to prevent attacks from being launched, but rather to thwart them after they were launched [7]. This is achieved by dynamically changing system configurations or data on which the success of the launched attack depends, such that the attacks fails during their execution [7].
An apt way to conceptualize how Moving Target Defense works, is to compare it to the shell game, in which a person shows another person a pea or some other small object,

puts it under one of three shells and then starts shuffling the shells, while asking a player of the game to guess where the pea is hidden after the shuffling, while promising some price money it the player gets it right [7], [10]. In this scenario the attacker takes the place of the player who has to guess under which shell the pea is in the end, the pea can be considered to be the current system state on which the attack depends, while the shuffling can be considered to be the Moving Target Defense mechanism which creates uncertainty and the price represents an successful attack. So even if the player (attacker) originally knew where the pea was (how the system was configured), through the shuffling (Moving Target Defense) the player (attacker) is now left in uncertainty when it comes to the current whereabouts of the pea (system configurations), which makes his guess less likely to be right and therefore diminishes his chances of getting his desired price.

2.2 ElectroSense Scenario

While the MTD Framework proposed later in this thesis is capable of running on any Linux based machine, the main focus when designing and implementing the MTD Framework and its MTD Solutions was put on the use case in which the MTD Framework would be used to keep a Raspberry Pi 4 acting as a spectrum sensor safe from outside interference.

2.2.1 Crowdsensing

The stated goal of the ElectroSense platform is to collect and analyze radio spectrum data across polulated regions by using a crowdsensing approach and to make the collected data available for everyone in real time [4]. Interested people can either apply for a sensor kit or simply buy the required hardware themselves and set up their sensor and then register it on the ElectroSense platform [4]. Once a spectrum sensor is up and running it will start monitoring and gathering radio spectrum data for its surroundings and send the gathered data to the ElectroSense platform.

2.2.2 Sensor Hardware & Setup

The sensor kit consists of a Raspberry Pi 3 or higher, a software defined radio receiver (RTL-SDR USB dongle) and a dipole antenna-set. For this thesis a Raspberry Pi 4 Model B with 4GB of memory was set up as a radio spectrum sensor. Once the software defined radio receiver with the antennas attached is plugged into the Raspberry Pi 4 and the Raspberry Pi 4 is connected to the Internet through an Ethernet connection the hardware setup is complete. To get the sensor up and running the only step left to do is burning a preconfigured image of Raspberry OS which is provided by ElectroSense to an SD card and then inserting the SD card into the Raspberry Pi. Afterwards the Raspberry Pi can be registered as a spectrum sensor on the ElectroSense platform and it immediately starts sending data to ElectroSense.

2.2.3 Threat Model

Given that the spectrum sensors can be physically set up anywhere where they can be supplied with a tethered Internet connection the sensors may be exposed to malicious actors which seek to interfere with the devices' integrity. By being connected to the Internet the spectrum sensor is also exposed to malicious actors over the web. In the next section three malware families will be introduced that pose a potential risk to these spectrum sensors.

2.3 Malware of Interest

For this thesis three malware families were considered that pose a potential threat to a Raspberry Pi 4 which is being used as a spectrum sensor. This section will provide a short overview about these three malware families.

2.3.1 Command & Control Based Malware

Command & Control (CnC) based malware is malware which establishes a communication channel between a victim machine and a CnC server and then uses said established channel to command and control the victim machine for a malicious purpose [11]. Examples of CnC based malware include but are not limited to botnets and backdoors. CnC elements can also be found across many types of malware such as some ransomware and rootkits, where they are used to orchestrate the attack or change the malware's behaviours over the network.
Given that the spectrum sensors are used in a passive manner and that they are not actively used by a person, they are especially attractive as a potential target for a botnet and other malware which takes control of the device. This is due to the fact that such attacks are less likely to be detected on a system that is not actively used by a person.

2.3.2 Crypto Ransomware

Crypto ransomware is a type of malware that is designed around monetary gain. Crypto ransomware typically encrypts a system's files or a subsection thereof using strong encryption techniques and then sends a message to the owner/user of the device stating that they are required to pay a ransom if they want the system's files to be decrypted/returned to their original state [12], [13]. Crypto ransomware typically does not aim to encrypt an entire hard disk or files that are essential for the system to function, but rather aims to encrypt specific files of predefined file types such as (for example) text files, videos, pictures and more file types that could be of interest to the victim [14].
The trend shows that ransomware attacks have been increasing over the years[14]. IoT devices (such as a Raspberry Pi 4) which themselves can hold records of data, can be under threat of being attacked by crypto ransomware [14].

2.3.3 User-Level Rootkits

User-level rootkits are rootkits that operate in the user space and work by abusing Linux's dynamic linker for shared libraries [15]. Linux allows its users to preload shared libraries by adding the path of a shared library or object to /etc/ld.so.preload [15]. These shared libraries are preloaded before any other libraries are loaded. This is done on a system wide level meaning that any application run on the system will also indiscriminately be preloading these shared libraries specified in /etc/ld.so.preload [16]. Linux also allows for multiple definitions of symbols across shared libraries, but the symbols are only resolved once meaning that the shared libraries/objects specified in /etc/ld.so.preload take precedence when checking to resolve said symbols [15], [16]. This opens up the possibility for symbols to be resolved to malicious code instead of legitimate code [15]. This is the entry point through which user-level rootkits on Linux operate. They create shared libraries with malicious versions of regular functions and then append the path of these shared libraries to /etc/ld.so.preload such that these overwrite/hijack specific symbols in order to gain the desired rootkit behavior.
The biggest issue present is that some rootkits also allow for /etc/ld.so.preload or any other files to be hidden. This is achieved by preloading C library symbols with custom implementations. This causes problems when trying to sanitize /etc/ld.so.preload, since a file that cannot be found also cannot be opened or adjusted in any other way. Usually writing to the path of a hidden file is also disabled, meaning that it is not possible to create a new file with the same name as the hidden file. Even worse, some rootkits go as far as replacing where Linux's dynamic linker checks for shared libraries/objects, meaning that /etc/ld.so.preload is unlinked as the place where the lookup should be done. By doing this the rootkit gains the ability to hide its manipulation since /etc/ld.so.preload will remain untouched in this case and look exactly as it did before installing the rootkit. Instead the file that replaces /etc/ld.so.preload will be appended with the shared libraries that hijack system functions.
Similarly to CnC based malware rootkits aim to give an attacker control over a system while remaining hidden. This makes the spectrum sensors especially attractive as a potential target for a rootkit since the rootkit behaviour might not be noticed by anyone given that the sensors are not actively used by a person.

Chapter 3

Related Work

3.1 MTD Design Principles

Every Moving Target Defense technique works by dynamically changing some parameters in order to make the system less predictable and therefore more secure. These parameters are called the moving parameter (MP) [2]. Every Moving Target Defense technique/approach needs to consider the following three design questions when it comes to the MP: WHAT to move, WHEN to move it and HOW to move it [2], [9].

3.1.1 WHAT

The WHAT is concerned with the actual components of the system which are dynamically changed in an effort to secure the system [2], [9]. The following taxonomy of moving parameter types has been proposed [17]:

- **Dynamic Data:** The format, syntax, encoding or representation of data is changed [17].
- **Dynamic Software:** An application's code is dynamically changed by modifying the program instructions, their order, their grouping and their format [17].
- **Dynamic Runtime Environments:** The runtime environment with which an application/program interacts is dynamically changed [17].
- **Dynamic Platforms:** These are concerned with changing the platform properties such as the currently running Operating System, but can also go as far as changing hardware such as swapping the CPU and therefore switching from one CPU architecture to another [17].
- **Dynamic Network Configurations:** These are concerned with changing some network properties, such as protocols or addresses [17].

3.1.1.1 Data Transformations

Moving Target Defense approaches falling under the data transformations category, dynamically change the format, syntax, encoding or representation of data [17], this can also include modifying the storage location or the format of data [18]. Data transformation have also been stated to be a way to achieve runtime transformations [18], which may make the distinction between MTD approaches using data transformations and MTD approaches using runtime Transformations blurry at times.

3.1.1.2 Runtime Transformations

Moving Target Defense approaches falling under the runtime transformations category, most of the time either employ Instruction Set Randomization in order to prevent code injection attacks from being successful [17], [19] or they employ Address Space Randomization in order to invalidate attacks, which rely on specific memory layouts such as stack overflow attacks[17], [20].

3.1.1.3 Software Transformations

Moving Target Defense approaches falling under the software transformations category, use software, which could be used as an attack vector, as the moving parameter [9]. The basic principle by which this technique works, is by creating multiple variants (diversification) of the same software or parts thereof, such that the software is functionally the same for its intended purpose, but different in its out of spec behaviour [9]. Creating different variants for diversification, can be achieved in many ways. This can be done on a compiler based approach, by hand, by outsourcing parts of the computation through the web and much more [9], [21].

3.1.1.4 Dynamic Platforms

Moving Target Defense approaches falling under the dynamic platforms category work by using the execution environment as the moving parameter [9]. One of the most used approaches in this category makes use of virtualization technologies, where an application or service is run in an instance of a virtual environment (active instance), while having a pool of backup instances from which the application can also be run [9], [22]. The backup instances are setup in a way to maximize diversity (different OS and/or configurations) between them, in hopes that an attacker would not be able to know beforehand, with which instance he would deal and therefore could not easily cater his attack to specific instance's vulnerabilities [22].

3.1.1.5 Network Address Shuffling

Approaches falling under the network address shuffling category use the network address or other network configurations as the moving parameter [9]. The basic principle by which they work is by invalidating knowledge gained through reconnaissance attacks, by periodically moving the system that would be a target of an attack from one address to another, such that the knowledge gained cannot be easily capitalized on [9].

3.1.2 WHEN

The WHEN is concerned with when the dynamic state change should occur [2], [9]. Depending on the attack type or the Moving Target Defense technique, it may not be enough to simply change the system state at any arbitrary time. For example, if an attack depends on an IP-address and the IP-address is changed before the attack setup/configuration happens, then the dynamic reallocation of the IP-address will not actually make any difference [10], since the attack setup takes the most recent IP-address into consideration, as can be seen in figure 3.1. This is because the actual attack setup takes place after the moving parameter was already moved, meaning that the new System State 2, was the system state for which the attack was actually preconfigured, such that the attack execution actually succeeds.
Similarly the changing of the moving parameter may be too late, such that the attack setup and attack execution actually take place in between the moving of the system states, which could then lead to a successful attack, as can be seen in figure 3.2.
So to recapitulate, often times timing is key when it comes to a Moving Target Defense technique, meaning that finding a sweet spot, in which it works is necessary as seen in figure 3.3. But timing might not be as crucial for every Moving Target Defense technique. For example it is imaginable, that an attack cannot be easily reconfigured during the attack setup to work with the new system state, even if the system state was changed prematurely. This would be the case for example, if an attack only works on a specific version of a specific OS. If the OS is now changed to a different version, this approach would work in every case in which the OS version was changed before the attack execution while it is irrelevant, if it was done before or after the attack setup as long as it was done before the attack happened.
Furthermore, the WHEN is also concerned with the frequency in which the MP should be moved and if should be moved as a reaction to an event or every so often after a given time interval has passed instead [9]. The following three options are possible: change the moving parameters triggered by an event, after a given time interval or by using a combination of the two [2]. The MTD based framework proposed in this thesis works on a event based deployment. This does not mean that periodically executing MTD Solutions through the framework is impossible, but to achieve this an attack/event report would need to be send to the framework each time the MTD Solution should be deployed.

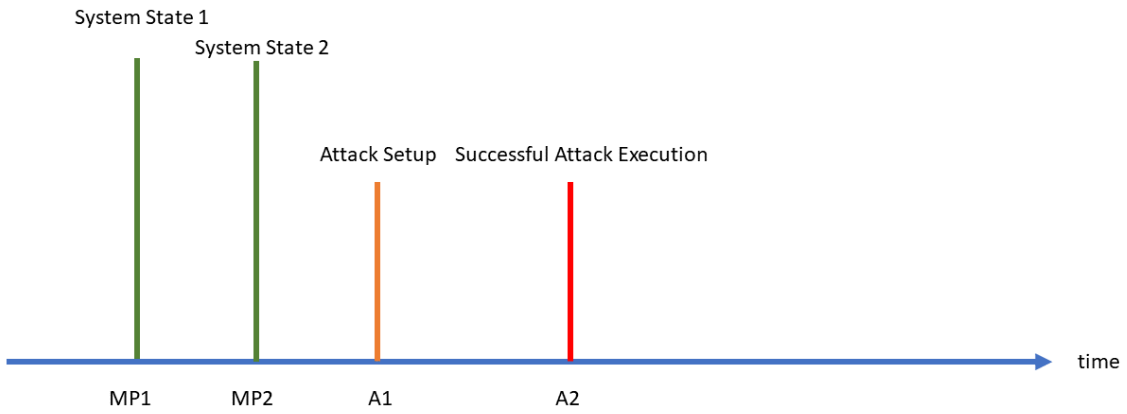


Figure 3.1: Attack is successful, because system state was changed too early.

3.1.3 HOW

The HOW is concerned with the means through which the moving parameters are actually moved and focuses on the following two operations: selection and replacement [9]. The selection operation, is the operation through which the next valid system state to which the moving parameters should be moved is chosen, while the replacement operation is concerned with the actual act of changing the moving parameters from the previous system state to the newly selected one [9].

3.2 Moving Target Defense in IoT

3.2.1 Distribution of Categories of Moving Target Defense in IoT

While many Moving Target Defense approaches have been proposed and tested over the years, only a small subsection of these approaches are meant to be used in IoT devices [2]. As previous literature has shown, the overall distribution of the MTD types used in IoT differs quite a bit from MTD types used in general [2]. For MTD in general the distribution looks as follows [2], [17] (see Figure 3.4):

1. Dynamic Runtime Environment at 35%
2. Dynamic Network Configurations at 21%
3. Dynamic Platform at 20%

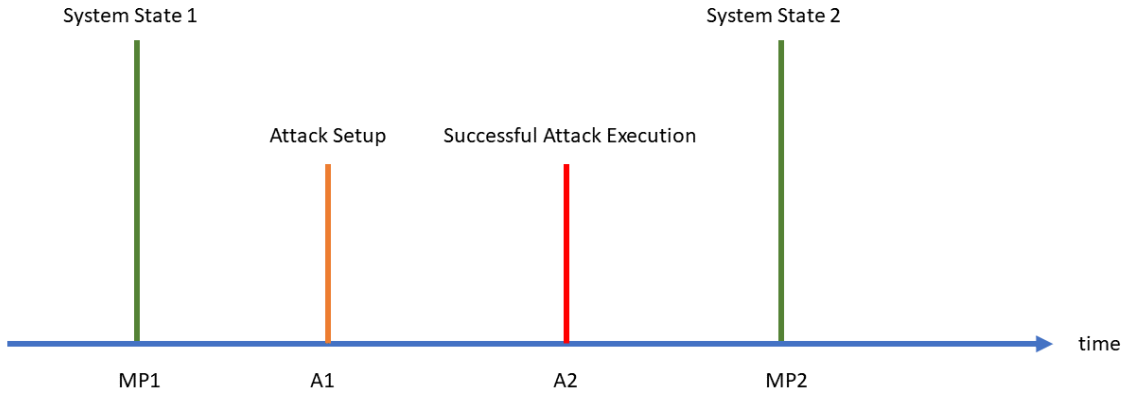


Figure 3.2: Attack is successful, because system state was changed too late.

4. Dynamic Software at 15%
5. Dynamic Data at 9%

In contrast the distribution for MTD in IoT looks as follows [2] (see Figure 3.5):

1. Dynamic Network Configurations at 54%
2. Dynamic Runtime Environment at 20%
3. Dynamic Software at 13%
4. Dynamic Data at 10%
5. Dynamic Platform at 3%

As can be seen above, approaches using the network configurations as the moving parameter are the most common ones by a large margin in IoT, while ones using a dynamic platform are almost non existent. This may be the case because of 1) the inability to change the hardware or parts thereof of an IoT device, 2) the inability to migrate to another operating system and because 3) virtualization as a means of changing between operating systems on the fly is also off the table considering the resource constrained nature of IoT devices such as the Raspberry Pi.

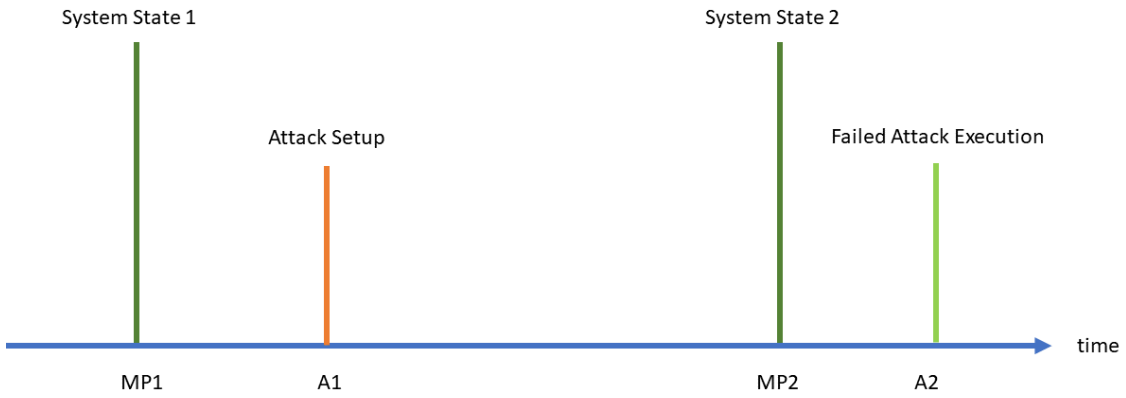


Figure 3.3: Attack fails, because system state was changed after attack setup up, but before attack execution.

3.2.2 MTD Solutions Not Considered Due to The Given Scenario

For this thesis only host-based MTD approaches were considered based on the spectrum sensor scenario. In this scenario the spectrum sensor is added to an already preexisting network as is, meaning that no additional hardware or setup should be required. This means that MTD approaches which rely on external devices were not considered in this thesis.
In theory using a dynamic platform based MTD approach making use of virtualization would make dealing with all three malware families presented in Section 2.3 trivial. Once malware would be detected a clean backup instance would be able to take over. But due to hardware limitations of the Raspberry Pi 4 it's virtualization capabilities are rather restricted which makes this type of approach unfeasible for the given scenario.
MTD approaches using software transformations try to keep a system safe by diversifying the software running on the system and making attacks based on software vulnerabilities less likely to succeed. For the malware families presented in Section 2.3 this category of MTD is not very relevant considering that the malware families do not work/achieve their intended goal by leveraging software vulnerabilities.
Network based MTD approaches using Software Defined Networking or Network Functions Virtualization to hide/change the IP address of the device are also not applicable within the given scenario, since they require additional Hardware.

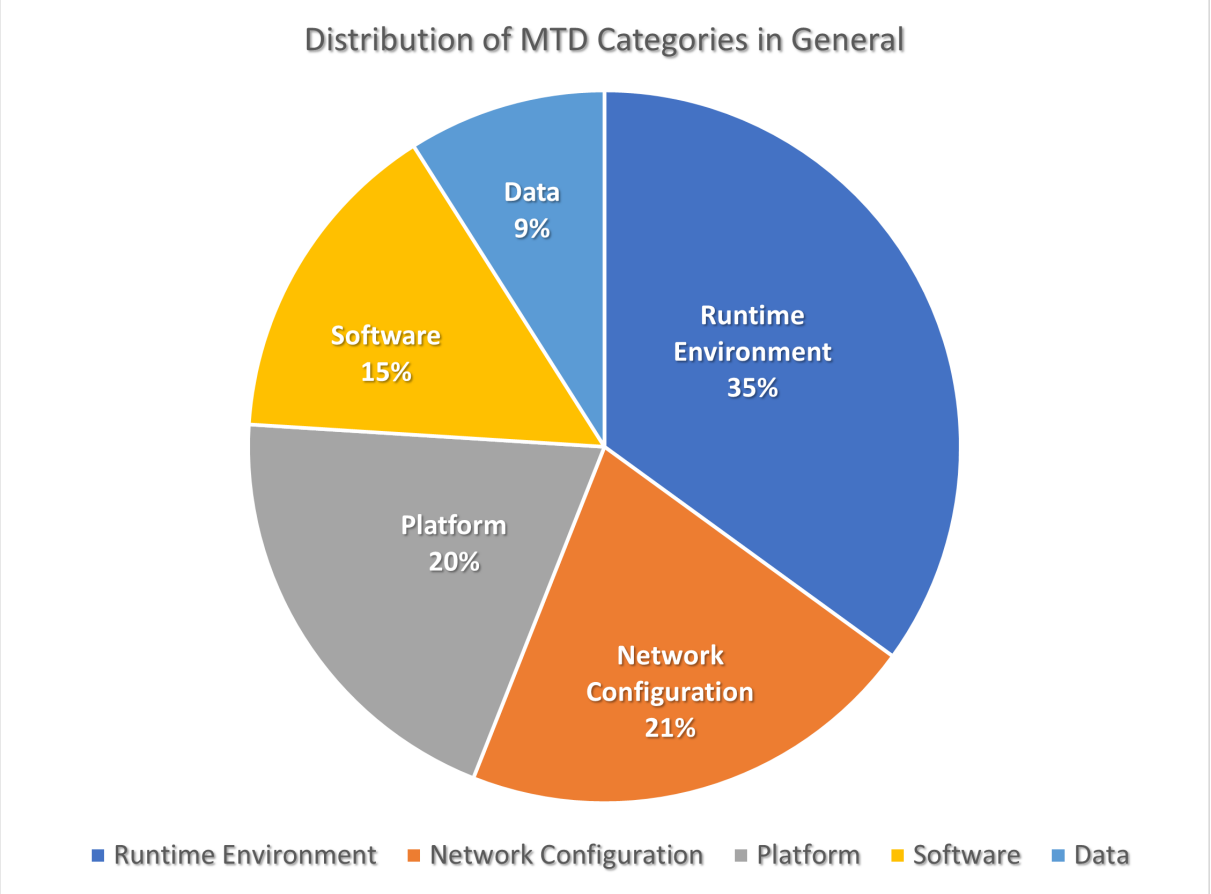


Figure 3.4: Distribution of MTD Categories in General

3.2.3 Noteworthy MTD Solutions

In the paper "MTD, Where Art Thou? A Systematic Review of Moving Target Defense Techniques for IoT" from 2021 Navas et al. explored the current state of MTD techinques for IoT [2]. In this paper the researches went through a rigorous literature review in order to find and classify most of the current MTD approaches for IoT. This section will present noteworthy MTD approaches for the given scenario.
One of the MTD approaches mentioned in [2] revolves around diversifying parts of the OS like symbol names of OS libraries to make it impossible for malware to call system functions[23], [24]. On a first look this approach might sound promising to deal with user-level rootkits that hijack such symbols, but it is important to understand that this would require the entire system to be built around the diversified symbol names. Therefore, this approach is not feasible for the given scenario where a pre-configured Raspberry OS image is provided by ElectroSense.
While there are no MTD approaches mentioned in [2] that are specifically intended to be used against CnC based malware, this thesis proposes that changing the private IP address might be a feasible way to disrupt a connection between a CnC server and an infected machine. To that extend Navas et al present multiple MTD approaches which use the IP addresses as their MP [2]. But the issue is that these MTD approaches do not directly apply to the given scenario. AShA [25] aims to change the IP addresses of an en-

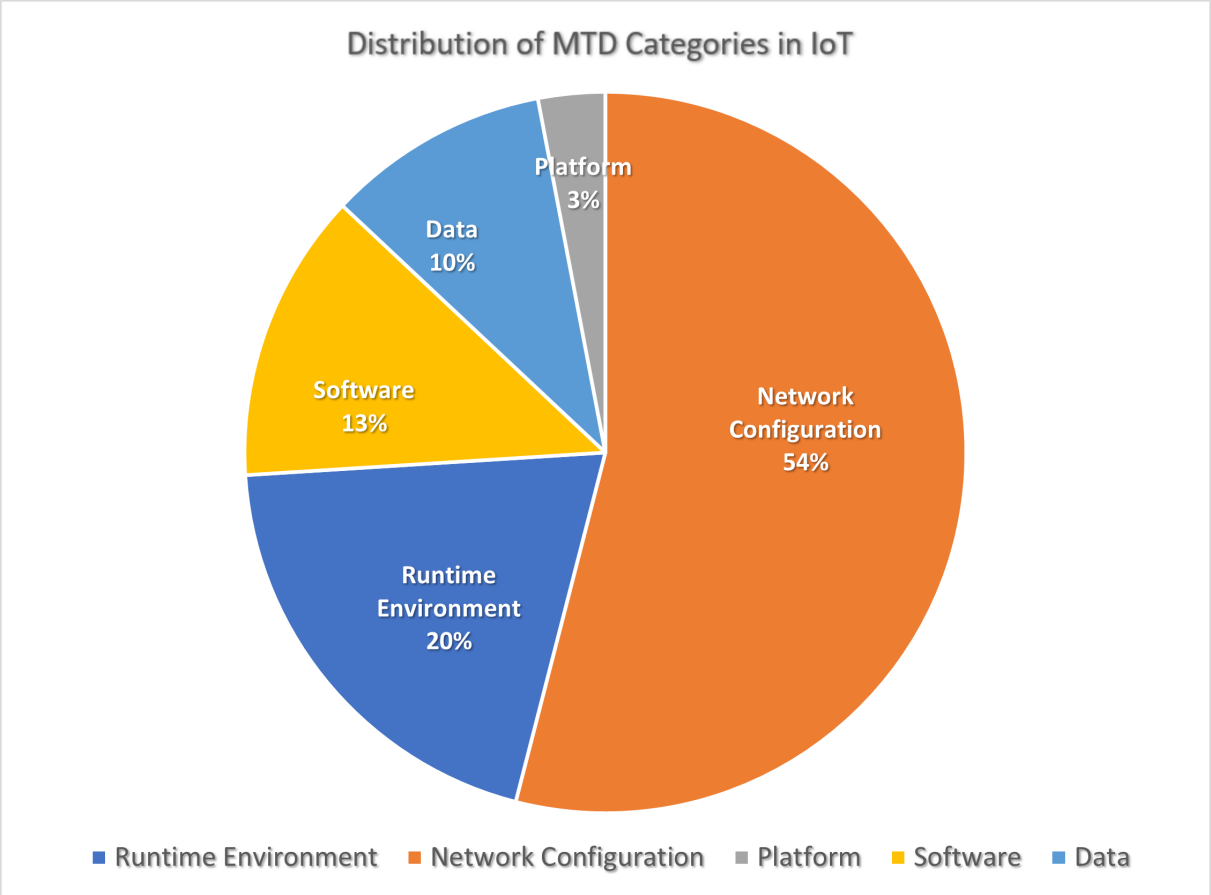


Figure 3.5: Distribution of MTD Categories in IoT

tire network of devices, 6HOP [26] and μM6TD [27] focuses on how to periodically change to a new IP6-address while also making said address accessible (using shared secrets) to certain devices outside of the network which is not necessary in the given scenario, because the spectrum sensor initiates a connection to the ElectroSense servers and starts sending data their way. There is no case in which the ElectroSense server would initiate the connection and would need to know the private IP address of the spectrum sensor.
While not directly applicable to the given scenario there is a deception based MTD approach mentioned in [2]. This MTD approach uses smartphones to act as virtual IoT modules to trap an attacker and waste his resources [28]. While this type of honeypot is not relevant for the given context, it still served as a inspiration for one of the MTD Solutions implemented in this thesis which relies on honeypotting a crypto-ransomware with dummy data (see Section 5.3.2).
Another deception based MTD approach which was proposed in the literature although without having IoT in mind, is an MTD approach meant to keep data safe from encryption by changing the file extensions of files [12]. By changing the file extensions the true file type of the file is hidden from the ransomware and a pseudo file type is presented in its place. While the proposed MTD approach was only implemented and tested on Windows, this MTD approach served as direct inspiration for the second MTD Solution against crypto ransomware (see Section 5.3.3).

3.2.4 MTD Framework

While there are quite a few MTD based security solutions for IoT, most of these solutions either cover a very specific use case or depend on specialized hardware to work. When looking at MTD in IoT, there is no literature on a generic MTD based framework which is capable of deploying MTD based security solutions based on attack/event reports. This thesis proposes and implements such a framework for IoT devices that run on Linux based operating system. This MTD based framework passively listens for attack reports and then deploys MTD based security solutions that were implemented beforehand.
Additionally to the MTD framework this thesis will propose and implement MTD based security solutions which are capable of dealing with a predefined set of malware families discussed in Section 2.3. These MTD based security solutions will then be used in conjunction with the MTD based Framework.

3.2.5 Research Gap

So far, there is no mention in the literature of a MTD based framework capable of deploying MTD based security solution in response to reported attacks/events. The goal of this thesis is to propose such an event based MTD framework and implement a prototypical version of said framework.
Furthermore, for the malware families mentioned in Section 2.3 there barely exist any literature discussing MTD approaches to deal with them. The only approach that was found during the literature review process of this thesis that directly addressed one of the malware families of interest was the MTD approach which changes the file extensions of files in order to keep data safe from an encrypting ransomware [12].
While an IP-address rotation/hopping scheme could be capable of dealing with CnC based malware, the literature mainly discusses such schemes in connection with DDoS attacks or reconnaissance attacks. The MTD approaches using IP-address rotation/hopping in the literature are also over-complicated for the given scenario, since they usually focus on rotating the IP address on a network wide level or they focus on a rotation scheme in which trusted entities are kept aware of the current IP-address. Both of these cases are irrelevant for the given use case in which the goal would be to simply disrupt an already established connection by changing the IP address after said connection was established.
So in conclusion, this thesis will also look into the possibilities of dealing with the malware families specified in Section 2.3 by using novel MTD approaches where no MTD approaches exist so far or by adapting MTD approaches which exist so they fit the given scenario.

Chapter 4

MTD Framework Architecture

This thesis proposes an MTD based framework (MTD Framework) which is capable of deploying preprogrammed MTD based security approaches/solutions (MTD Solutions) in response to attack reports send to the MTD Framework by a external monitoring program/application which is capable of recognizing and identifying ongoing attacks and/or running malware. The external program monitoring for attacks/malware is not part of this proposal.
While the proposed MTD Framework is capable of working on a multitude of Linux based systems, this thesis will be focusing on its usage as a security measure to keep a Raspberry Pi 4 which is being used as a spectrum sensor secure from cyberattacks.

4.1 Design

The MTD based framework consists of three main components: the **MTD Deployer Server**, the **MTD Deployer Client** and the **MTD Solutions**. The MTD Deployer Server is meant to be run as a Linux service at all times. It is responsible for passively listening for attack/event reports from external programs which's responsibility it is to monitor the system for malware infections or ongoing attacks. These external programs can send attack reports to the MTD Deployer Server by using the MTD Deployer Client which provides an interface to communicate to the MTD Deployer Server. The MTD Deployer Client and the external program do not necessarily need to be run on the Raspberry Pi 4 on which the MTD Deployer Server is running, as will be shown later. This opens up the possibility for attack reports from programs that monitor attack patterns that are not localized to the Raspberry Pi 4 alone (e.g. monitoring across an entire network).
Once an attack report has been send with the MTD Deployer Client, the MTD Deployer Server then deploys an appropriate MTD Solution from the set of available MTD Solutions. This is done by checking against a config file if for the reported type of attack there are any MTD Solutions specified which should be deployed in case of an ongoing attack. If a MTD Solution is configured for the reported type of attack the MTD Deployer Server then launches the corresponding MTD Solution's script. The MTD Solutions themselves are the actual components which are responsible of mitigating or disrupting occurring attacks/running malware.

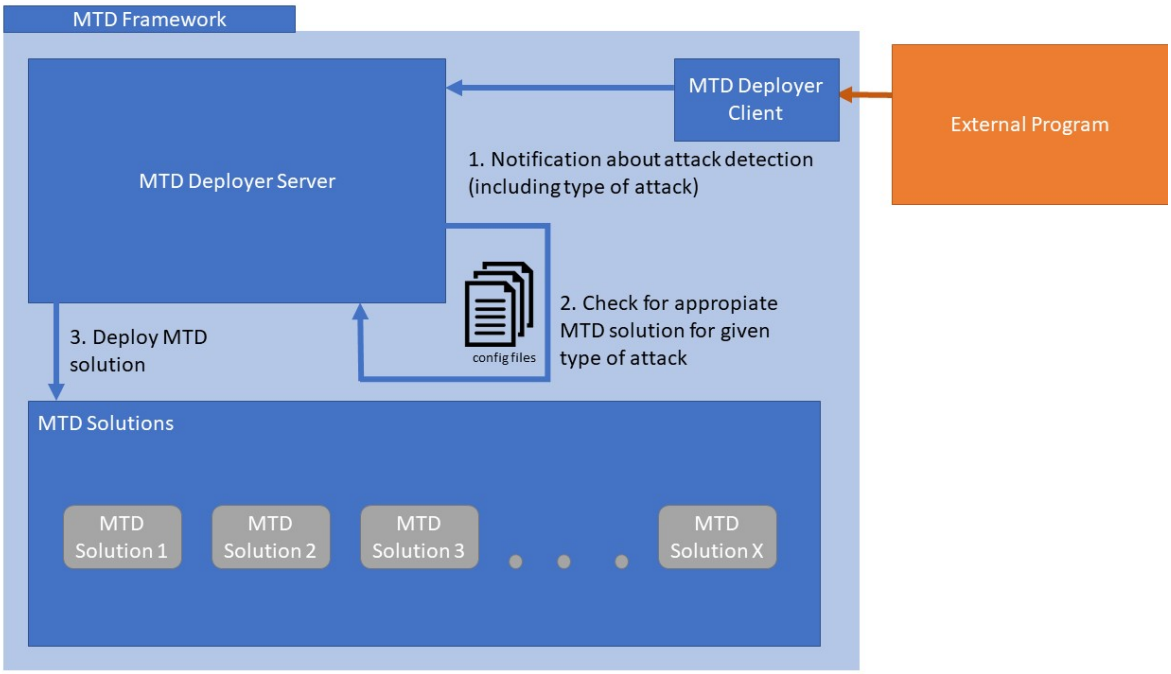


Figure 4.1: High Level Overview of MTD Framework

4.2 Necessary Properties

Given the landscape of evolving cyberattacks and considering that this framework is primarily designed to be used on a Raspberry Pi 4 operating as a spectrum sensor, there are three key properties which the framework must abide to. Firstly the framework should be **modular** and **configurable** by a system administrator. Secondly the framework needs to be **lightweight** and should therefore not consume a significant amount of system resources while it is passively running and waiting for attack reports.

4.2.1 Modularity & Configurability

Since cyberattacks are constantly evolving and new types of vulnerabilities are found on a regular basis there needs to be a possibility to easily adapt to changing circumstances. This is achieved by the modular nature of the framework which easily allows the system's administrator to add and remove new MTD Solutions or Attack Types by simply adjusting the config file. Once a new MTD Solution script is added to the Raspberry Pi 4, the only thing left to do is adjusting the config file such that it reflects the existence of a new MTD Solution such that the MTD Deployer Server is able to actually deploy the MTD Solution when needed.
Apart from the Attack Types and the MTD Solutions, there are also other aspects of the MTD Framework which need to be configurable, such as if attack reports from other machines should be allowed and which port the MTD Deployer Server should listen to for attack reports. These can also be configured through the config file.

4.2.2 Lightweight

While the computational resources on a Raspberry Pi 4 may not be as constrained as on many other IoT-Devices, the resources are by no means abundant, especially when compared to full fledged computer systems. Taking the resource constrained nature of the Raspberry Pi 4 into consideration and considering that the MTD Deployer Server is meant to be run as a service that starts up at boot and runs at all times, it is necessary for the MTD Deployer Server to be as lightweight as possible to free up system resources for the Raspberry Pi 4. While the MTD Deployer Server is running as a service and passively listening for attack reports it should not have any significant performance impact on the Raspberry Pi 4 such that the Raspberry Pi 4 can still fully operate as an ElectroSense spectrum sensor. This is achieved by effectively only doing work when an attack reports comes in, meaning that while the MTD Deployer Server is passively listening for an attack report there are no operations executed until a report comes in.

Chapter 5

MTD Framework & MTD Solutions Implementation

In this chapter the thesis will provide a general technological overview of the MTD Framework as well as present the MTD Solutions which were implemented to defend against the following three malware types which may affect a spectrum sensor: Command & Control based malware, crypto ransomware and user-level rootkits (in Linux). The source code for the MTD Framework and the MTD Solutions can be found in [29].

5.1 Overview

The MTD Framework is implemented in Python. The MTD Deployer Server (MTDDeployerServer.py) works by opening a TCP socket on a preconfigured port (see Listing 5.1, lines 77 to 70) and listening for attack reports which can be sent using MTD Deployer Client (MTDDeployerClient.py). To generate and send an attack report an external program must run the MTDeployerClient.py with the following arguments: --ip, --port, --attack (e.g. "python MTDDeployerClient.py --ip 192.168.1.136 --port 1234 --attack Test"). The --ip argument specifies the IP address of the Raspberry Pi 4 on which the MTDDeployerServer.py is currently running and similarly the --port argument specifies the port number of the socket on which MTDDeployerServer.py is listening for attack reports. Lastly the --attack argument specifies which type of attack is ongoing or which type of malware is running. This --attack argument is the parameter by which MTDDeployerServer.py identifies which MTD Solution needs to be deployed. When MTDDeployerClient.py is run it tries to establish a connection to the socket on the specified IP address and port and then sends a message with the specified attack type if a connection was able to be established.
For configuration the MTD Framework relies on a json file (config.json) which needs to be in the same directory as MTDDeployerServer.py. The configuration file is built around and should adhere to the schema presented in Listing 5.1. Failing to do so, will result in MTDDeployerServer.py throwing an error.

Listing 5.1: Schema of Config File

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "AttackTypes": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "Type": {
            "type": "string"
          },
          "MTDSolutions": {
            "type": "array",
            "items": {
              "type": "object",
              "properties": {
                "Priority": {
                  "type": "integer"
                },
                "ScriptName": {
                  "type": "string"
                },
                "AbsolutePath": {
                  "type": "string"
                },
                "RunWithPrefix": {
                  "type": "string"
                },
                "Params": {
                  "type": "string"
                }
              },
              "required": [
                "Priority",
                "ScriptName",
                "AbsolutePath",
                "RunWithPrefix"
              ]
            }
          },
          "DeploymentPolicy": {
            "type": "object",
            "properties": {
              "ScriptName": {
                "type": "string"
```

```
                },
                "AbsolutePath": {
                  "type": "string"
                },
                "RunWithPrefix": {
                  "type": "string"
                }
              },
              "required": [
                "ScriptName",
                "AbsolutePath",
                "RunWithPrefix"
              ]
            }
          },
          "required": [
            "Type",
            "MTDSolutions"
          ]
        }
      },
      "AllowAllExternalReports": {
        "type": "boolean"
      },
      "WhiteListForExternalReports": {
        "type": "array",
        "items": {
          "type": "string"
        }
      },
      "PortToUse": {
        "type": "integer"
      }
    },
    "required": [
      "AttackTypes",
      "AllowAllExternalReports",
      "WhiteListForExternalReports",
      "PortToUse"
    ]
}
```

If not configured to the contrary only attack reports stemming from the Raspberry Pi 4 lead to the deployment of an MTD Solution which means that MTDDeployerClient.py would have to be run on the Raspberry Pi 4 itself, to result in the deployment of any MTD Solution. This is to limit the possibility of a malicious entity triggering the deployment of MTD Solutions when not necessary. Nonetheless, the framework can be configured via the

config.json to allow for reports from different devices over the network. This can be done by setting AllowAllExternalReports (see Listing 5.1, lines 68 to 70) to true. When this json property is set to true any incoming report will be processed by the MTD Framework. If this behaviour is not desired and only selected devices should be able to report attacks over the network, AllowAllExternalReports can be set to false and a list of trusted IP addresses can be defined in WhiteListForExternalReports instead (see Listing 5.1, lines 71 to 75).
To add a new attack type for which MTDSolutions should be deployed all the administrator has to do is add a new entry in "AttackTypes". This is done by creating an entry that has a property named "Type", an optional one named "DeploymentPolicy" and one named "MTDSolutions". The latter's value is an array of subentries which themselves have the following required properties: "Priority", "ScriptName" ,"AbsolutePath" and "RunWithPrefix" and the optional property "Params". This could for example look as follows:

Listing 5.2: Example AttackType Entry

```
1   {
2     "Type": "Test",
3     "MTDSolutions": [
4       {
5         "Priority": 1,
6         "ScriptName": "Test.py",
7         "AbsolutePath": "/home/username/Desktop",
8         "RunWithPrefix": "python3",
9         "Params": "--int 1"
10      },
11      {
12        "Priority": 2,
13        "ScriptName": "Test2.py",
14        "AbsolutePath": "/home/username/Desktop",
15        "RunWithPrefix": "python3"
16      }
17    ]
18  }
```

The property "Type" defines the key which should be given to MTDDeployerClient.py as the --attack argument. If an attack is reported MTDDeployerServer.py will launch the script with the highest "Priority" value (lowest number). Assuming that the example shown in Listing 5.2 is in the config file and MTDDeployerClient.py is called with the --attack argument "Test", this would result in the following command execution by MTDDeployerServer.py: "cd /home/username/Desktop/" followed by "python3 Test.py --int 1". The reason why the directory is changed, instead of combining the two commands into a single command "python3 home/username/Desktop/Test.py --int 1" is to effectively change the working directory. If this is not done, the MTD Framework could run into issues when it comes to retrieving files if it were the case that the MTD Solution depends on some relative file paths, since the current work directory in that case would still be the directory in which MTDDeployerServer.py resides.

It should also be noted that by defining ”RunWithPrefix” it is actually possible to decide with which command the MTD Solution’s script is executed. Not only does this make it possible to run MTD Solutions which are not implemented in Python, but it also opens up the possibility to use different Python versions across different MTD Solutions or even go as far as using virtual environments when using Python such that the main python environment is kept clean from dependencies that are only needed by MTD Solutions.
By specifying the optional Property ”Params” it is possible to pass parameters as arguments to an MTD Solution’s script. Since there is no limit on how many entries in ”MTDSolutions” can use the same script name and path, it is possible to create multiple configurations of the same MTD Solution which only differ by the parameters which are passed as arguments.
If there are multiple MTD Solutions defined for an AttackType and instead of deploying the MTD Solution with the highest priority, one wants to dynamically decide based on some rule set or policy which MTD Solution should be deployed, it is necessary to configure the optional property ”DeploymentPolicy” within the AttackType entry. ”DeploymentPolicy” itself needs to be provided with the properties ”ScriptName”,”AbsolutePath” and ”RunWithPrefix”. The DeploymentPolicy script should implement the rule/policy by which an MTD Solution is decided and then write the priority number of the MTD Solution which should be deployed to stdout. After MTDDeployerServer.py receives the priority number, it launches the MTD Solution with that specified priority value. Assuming that TestRule.py in Listing 5.3 would write ”2” to stdout, MTDDeployerServer.py would deploy the second MTD Solution over the first one.

Listing 5.3: Example AttackType Entry With DeploymentPolicy

```
{
  "Type": "Test",
  "MTDSolutions": [
    {
      "Priority": 1,
      "ScriptName": "Test.py",
      "AbsolutePath": "/home/username/Desktop",
      "RunWithPrefix": "python3",
      "Params": "--int 1"
    },
    {
      "Priority": 2,
      "ScriptName": "Test2.py",
      "AbsolutePath": "/home/username/Desktop",
      "RunWithPrefix": "python3"
    }
  ],
  "DeploymentPolicy": {
    "ScriptName" : "TestRule.py",
    "AbsolutePath" : "/home/username/Desktop",
    "RunWithPrefix": "python3"
  }
}
```

5.2 Command & Control based Malware

In this section an MTD Solution capable of dealing with Command & Control (CnC) based malware will be presented. The presented MTD Solution, aims to deal with CnC based malware that has already successfully established a communication line between the victim machine and the CnC server. The goal of the proposed MTD Solution is to disrupt the established communication line such that the CnC based malware can not easily recover from this disrupted state in which no communication is possible any longer. To achieve this goal the MTD Solution uses a simple IP address randomization scheme to move the device from its current private IP address to a randomly selected new one which causes the communication line to break.

5.2.1 WHAT

This MTD Solution uses the private/local IP address as the moving parameter and therefore is an MTD approach that relies on dynamically changing the network configuration. In changing the private IP address the router's NAT lookup table is invalidated, such that any incoming commands from the CnC server intended for the victim device will not reach it anylonger.
While dynamically changing the private IP address is not a novel concept in the MTD space as can be seen in Chapter 3, using it to disrupt a connection instead of invalidating reconnaissance attacks or simply hiding your device from an attacker before the effective attack is launched is.

5.2.2 WHEN

The script of this MTD Solution should be started **after a connection to the CnC server has been established**. It is important that a connection to the CnC server has already been established before deploying this MTD SOlution, else the desired effect of disrupting the established connection will not occur, since the connection will be established from the already changed private IP address which will be used for communication later on.

5.2.3 HOW

To help understand how this MTD Solution works a flowchart is provided in Figure 5.1. This MTD approach works by first retrieving all IP addresses that are currently assigned to an active device in the same network (see Algorithm 1, line 4). This is achieved by using arp-scan to discover the IP address of other devices connected to the same network. The reason why this is done by using arp-scan instead of using ping or another method, is that even devices that block all IP communication will acknowledge and reply to an ARP request [30]. By doing this a list of all IP addresses that are currently assigned to a running device in the same network can be generated. Therefore it is also possible to infer

all private/local IP addresses in the network that are not currently in use, by generating a list of all possible IP addresses in the network and then removing the IP addresses that are already assigned to a device (see Algorithm 1, from line 5 to 11). After a list of

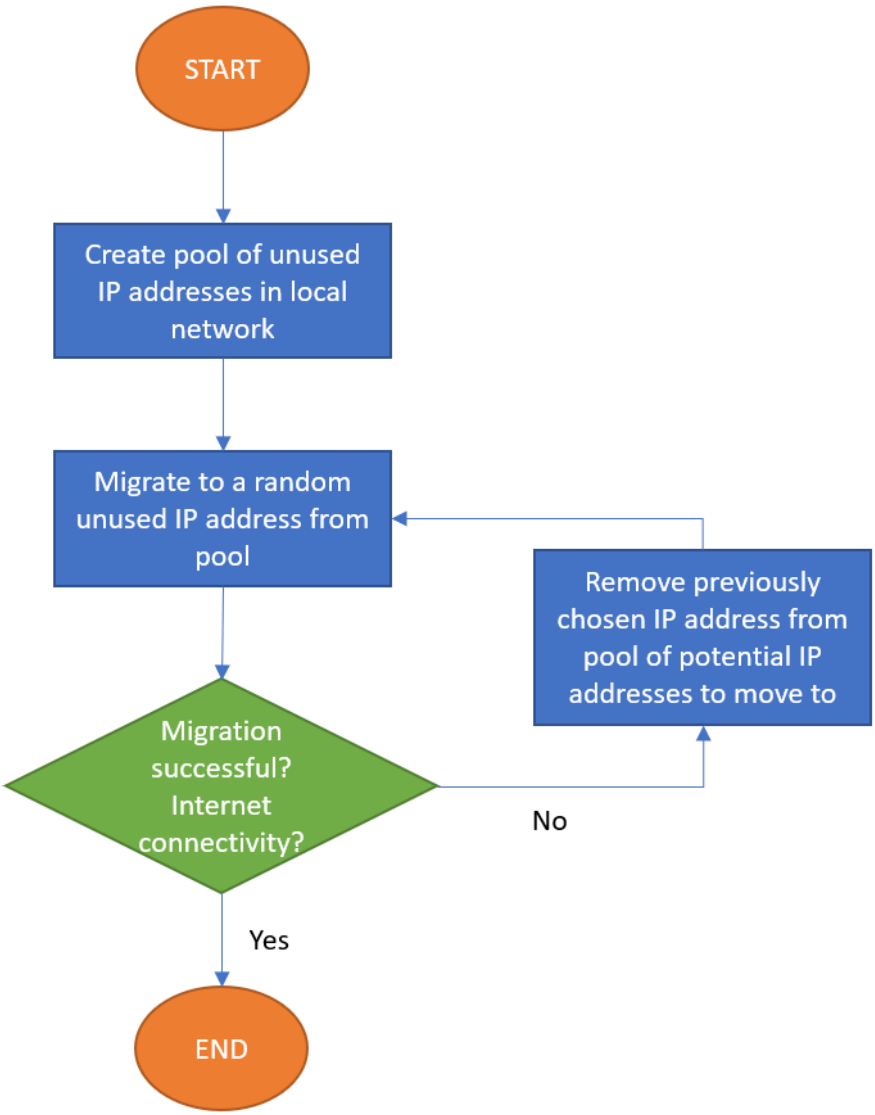


Figure 5.1: Simplified Flowchart of MTD Solution Which Changes IP Address

possible IP addresses which the Raspberry Pi 4 should be able to migrate to has been generated, a IP address from this list is chosen at random (see Algorithm 1, line 12). Once a target IP address for migration has been chosen, the Raspberry Pi 4 requests to migrate to the given IP address by using ifconfig (see Algorithm 1, line 14). At this point two scenarios are possible a) the Raspberry Pi 4 gets assigned the requested IP address by the network router and an internet connection is successfully established or b) the Raspberry Pi 4's request is refused (e.g. IP address is already reserved for another device belonging to the network that is currently offline and therefore did not show up on the arp-scan). In the latter case this results in the Raspberry Pi 4 being offline and not having an IP address assigned to it which would also prevent anyone from accessing the Raspberry Pi

Algorithm 1 Simplified pseudo code algorithm for migrating to random private IP address

```
1: function MIGRATEIP
2:     listOfIpAddressesToDiscard = []
3:     while True do
4:         listOfIpAddressesInUse = getIpAddressesAlreadyInUse()
5:         listOfPossibleIpAddresses = genereatePossibleIpAddresses()
6:         for ipAddress in listOfIpAddressesInUse do
7:             listOfPossibleIpAddresses.remove(ipAddress)
8:         end for
9:         for ipAddress in listOfIpAddressesToDiscard do
10:            listOfPossibleIpAddresses.remove(ipAddress)
11:        end for
12:        ipAddressForMigration = getRandomIp(listOfPossibleIpAddresses)
13:        listOfIpAddressesToDiscard.append(ipAddressForMigration)
14:        migrateToIpAddress(ipAddressForMigration)
15:        if checkIfInternetConnectivity() == True then
16:            restartElectrosenseService()
17:            break
18:        end if
19:    end while
20: end function
```

4 remotely at this point.
To distinguish between those two scenarios, the MTD script next checks if there is internet connectivity by pinging Google see (see Algorithm 1, line 15). If it successfully pings Google, the migration to the new IP address was successful. In this case, the last step is to restart the Electrosense service, since the connection to Electrosense also gets disrupted by the MTD approach (see Algorithm 1, line 16). By doing this the MTD approach aims to keep the time in which the sensor is unavailable to a minimum. If pinging Google did not work the script assumes that this means that migrating to the new IP address failed. In that case the script just runs again, while removing the IP address to which it tried to connect to and failed from the pool of possible IP addresses (see Algorithm 1, lines 9 to 11 and line 13).

5.2.4 Implementation

This MTD approach's script is implemented in Python and does not require any additional Python libraries to work, but it requires arp-scan and ifconfig to be installed on the Raspberry Pi 4, which can be installed using the command "sudo apt-get install -y arp-scan" from the terminal. Ifconfig should come preinstalled on Raspberry Pi OS by default. The script does not take any parameters/arguments.

5.3 Crypto Ransomware

For the scenario where a Raspberry Pi 4 is being utilized as an ElectroSense radio spectrum sensor, two MTD Solutions where implemented to address crypto ransomware attacks. Both MTD Solutions are coupled with a script that aims to identify and kill the running ransomware process while it is still encrypting files. The MTD Solutions themselves aim to keep as much data/files as possible safe and unencrypted and thus minimize the damage done by the crypto ransomware.

5.3.1 Identifying and Killing Encrypting Process

To help understand how the script (referred to as KillProcess from now on) responsible for identifying and shutting down the encrypting process works a flowchart is provided in Figure 5.2. KillProcess works by first retrieving a list of all currently running processes on the Raspberry Pi (see Algorithm 2, line 2). After this step, the CPU usage of all processes is monitored for a short while and all processes falling below a minimum threshold of 10% CPU usage are discarded (see Algorithm 2, line 3). These processes are discarded based on the assumption that encrypting files is a CPU intensive task. Therefore it is only logical to discard the many processes outright which do not take up a significant amount of CPU resources. In a next step further processes are removed from the set of processes that could possibly be the encrypting process based on their process name. If the process name belongs to a set of processes that are expected to be running (see Algorithm 2, line 4) such as processes related to the MTD Solutions that will be discussed below and processes related to the ElectroSense platform, they are removed from the list.

Algorithm 2 Simplified pseudo code algorithm responsible for identifying and killing encrypting process

```
1: function killProcess
2:     listOfActiveProcesses = getListOfAllRunningProcesses()
3:     filteredListOfProcesses = filterProcsBelowThresh(10%, listOfActiveProcesses)
4:     filteredListOfProcesses = filterKnownSafeProcesses(listOfActiveProcesses)
5:     sortedListOfProcessesDesc = sort(filteredListOfProcesses)
6:     for process in sortedListOfProcessesDesc do
7:         numOfFilesOpened = getNumOfFilesOpenedByProcessInNSeconds(process,
   60)
8:         if numOfFilesOpened >= 5 then:
9:             kill(process)
10:            break
11:        end if
12:    end for
13: end function
```

After the processes have been reduced to a list of suspicious processes, the processes are sorted in descending order of their monitored CPU usage. Next the script takes advantage of the fact that ransomware usually encrypts a multitude of files over a short time. For

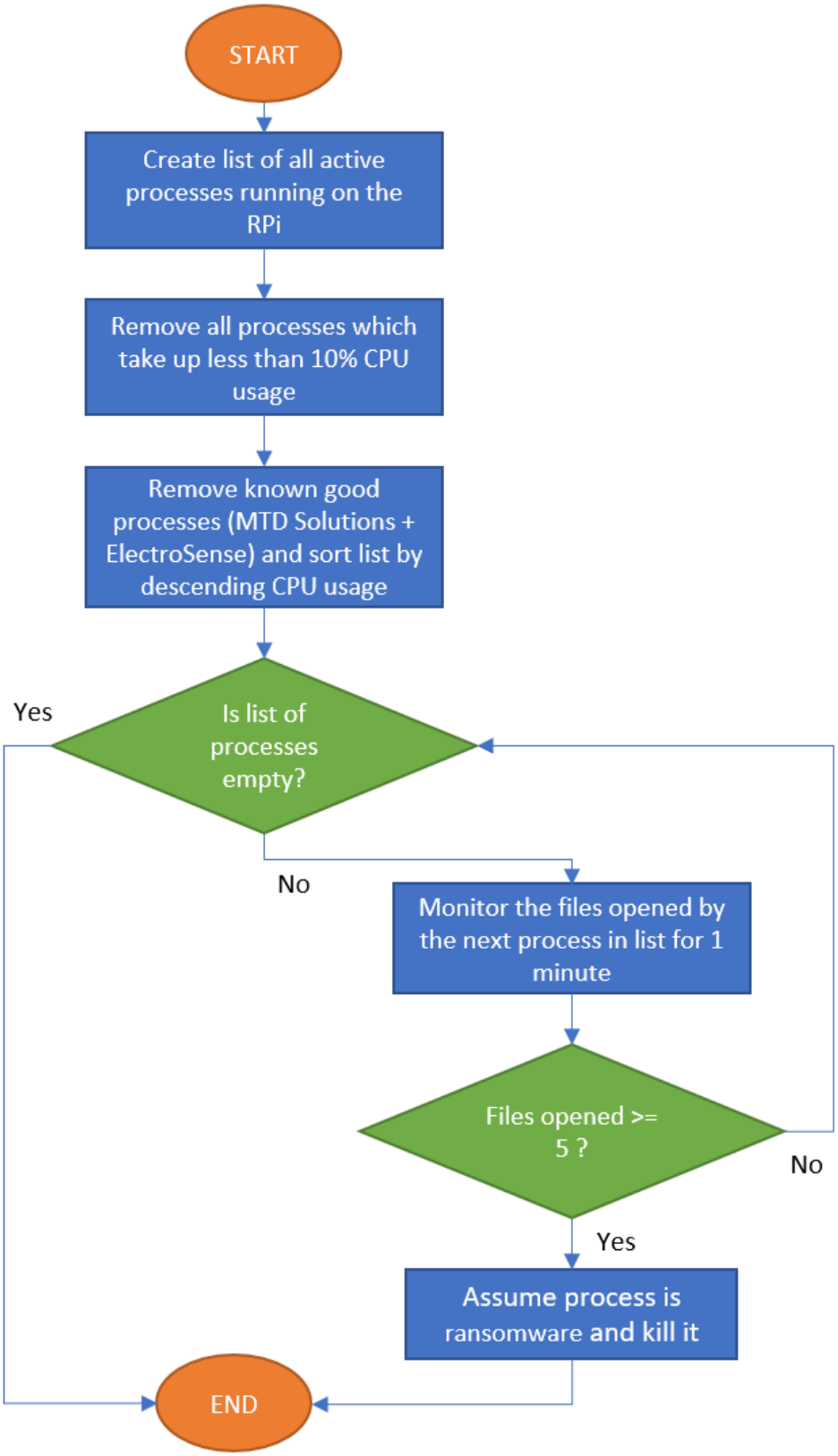


Figure 5.2: Simplified Flowchart of Script Responsible to Identify and Kill Ransomware Process

each suspicious process, starting with the process exhibiting the highest CPU usage, the script then monitors how many files are opened by that process within a minute (see Algorithm 2, line 7). If the number of files opened equals or exceeds 5 different files the currently monitored process is assumed to be the encrypting process and the process is killed (see Algorithm 2, line 9). If no encrypting process could be identified no process is killed.

5.3.2 Honeypotting the Encryptor

5.3.2.1 WHAT

The moving parameter of this MTD Solution is the file system and its structure itself, meaning that this MTD Solution is a data based MTD approach. It works by continuously creating dummy files acting as a honeypot such that the encrypting process of the ransomware gets stuck encrypting dummy data, while the script described in Section 5.3.1 tries to shut down the encrypting process. This MTD Solution and the script responsible for shutting down the encrypting process work in a symbiotic manner. While the MTD Solution ideally stalls the encrypting process and prevents it from doing major damage, the script described in Section 5.3.1 analyzes the running processes in order to find out which one is the encrypting one and then shuts it down. Therefore the goal of this MTD approach is to minimize the amount data that is encrypted by the ransomware, by trapping the encryptor in an endlessly expanding file tree populated with dummy files.

5.3.2.2 WHEN

This MTD Solution should not be deployed before the ransomware is running. The deployment of this MTD Solution deployment only makes sense while an active attack is ongoing and files are being encrypted considering that the goal of this specific MTD Solution is to trap the encryptor of the ransomware while it is executing.

5.3.2.3 HOW

In order to understand how the script responsible for creating the dummy files and endlessly trapping the crypto ransomware in its encryption phase works, it is first necessary to understand how ransomware typically traverses a file system and encrypts files of interest along the way. Usually ransomware encrypts files by stepping through directories in a recursive manner and encrypting the files of interest in each directory it passes, as can be seen in Algorithm 3. This means that in a first step all files and sub-directories in the root/start directory are gathered (see Algorithm 3, lines 2 and 3). Next, any files matching one of the predefined file extensions are encrypted (see Algorithm 3, lines 5 to 7). Once all the files that should be encrypted in the current directory were encrypted, the same algorithm is recursively executed for all the sub-directories within the current directory (see Algorithm 3, lines 9 to 11).

Algorithm 3 Simplified pseudo code algorithm for traversing and encrypting files recursively used in ransomware

```
1: function encryptFiles(directory, listOfFileTypesToEncrypt)
2:     listOfFiles = getFiles(directory)
3:     listOfSubdirecotries = getSubdirectories(directory)
4:     for file in listOfFiles do
5:         if getExtension(file) in listOfFileTypesToEncrypt then
6:             encrypt(file)
7:         end if
8:     end for
9:     for subdirectory in listOfSubdirectories do
10:        encryptFiles(subdirectory, listOfFileTypesToEncrypt)
11:    end for
12: end function
```

Given the fact that the aforementioned MTD Solution responsible for trapping the ransomware, is meant to be run after the encryption has already started, there is a problem when it comes to infinitely creating dummy files to trap the ransomware. Simply put, if dummy files are created in a directory which is currently being encrypted, this will not have the desired effect of trapping the ransomware, since in that case the list of files to encrypt (see Algorithm 3, line 2) would already have been cached, meaning that files added afterwards to the same directory, would only take up disk space, but would not result in trapping the encrypting process.
To help understand how this MTD Solution works a flowchart is provided in Figure 5.3. The issues presented in the previous paragraph are why this MTD Solution instead creates dummy files in a preexisting random sub-directory of the specified directory in which it is executed (see Algorithm 4, line 2). When the script is running, it keeps creating new dummy files, while continuously migrating to a new sub-directory every x dummy files, where x denotes a chosen amount of dummy files before moving to a newly created sub-directory (see Algorithm 4, lines 5 to 17). Migrating to a new sub-directory is necessary, since the dummy files need to be created in a directory which is not currently being encrypted, but which is a sub-directory of the one that is currently being encrypted. This means that once the encryption in a file directory starts, no files added to the same directory will be encrypted. Similarly, no sub-directories added to the directory that is currently being encrypted, will be recursively encrypted.
Looking at Figure 5.4 and assuming that directory A is currently being encrypted, it would not be possible to trap the encrypting process by simply putting more files into directory A. This would not work because the ransomware is only aware of the files and directories that were present when it stepped into directory A meaning that no matter how many files or directories are added to directory A after the ransomware has already stepped in, only the files surrounded by the yellow rectangle (1,2,3) will be encrypted. Similarly only the directories (B,C) surrounded by the yellow rectangle will be stepped into to encrypt their files after the files mentioned above were encrypted.
As can be seen in Figure 5.5, this means that the MTD Solution needs to ensure that at all times there exists a sub-directory (D) within the sub-directory (B) in which the MTD Solution is creating dummy files (4,5,6,7). This is to avoid a scenario in which the

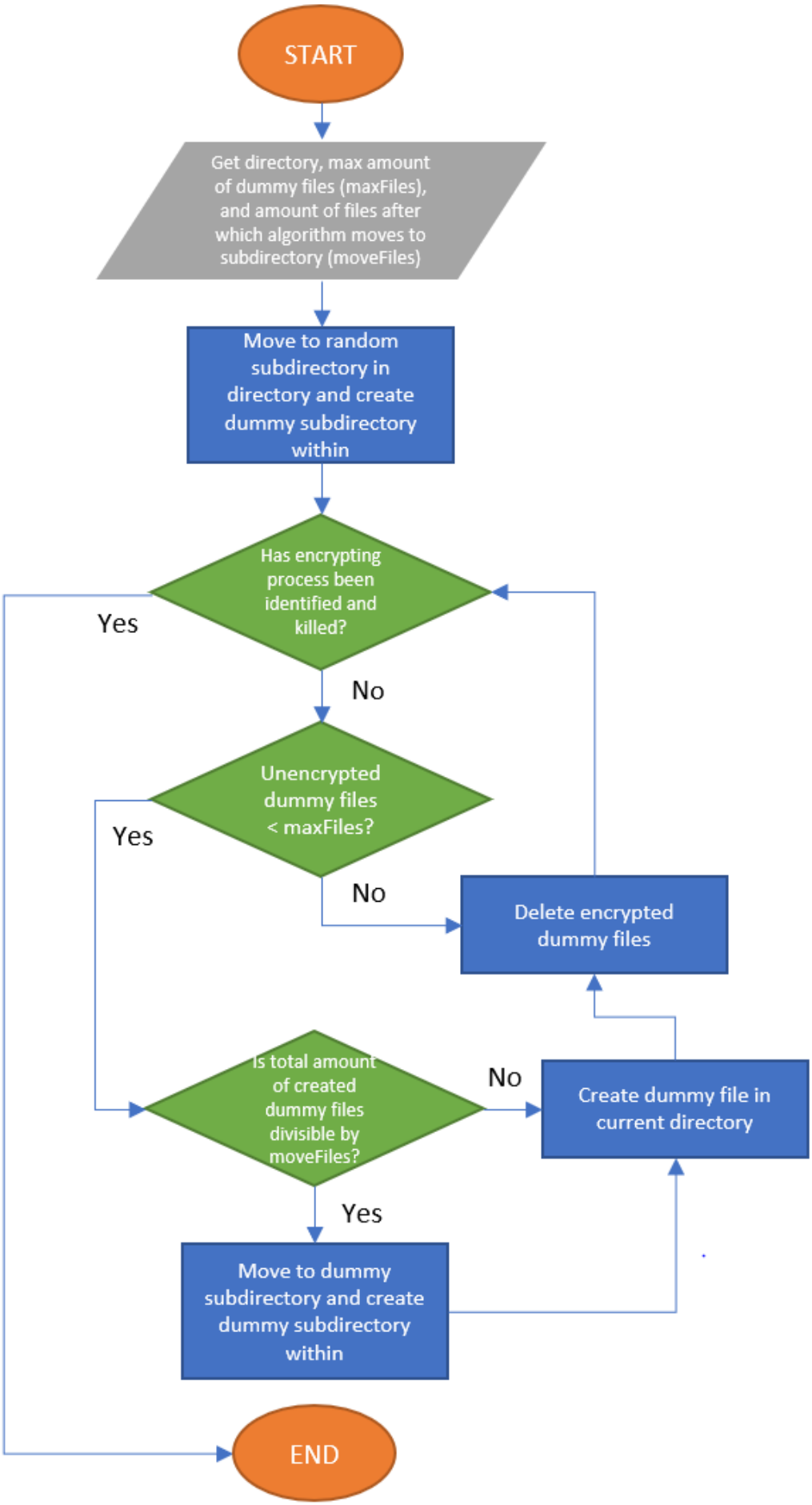


Figure 5.3: Simplified Flowchart of MTD Solution That Traps Ransomware

Algorithm 4 pseudo code algorithm for trapping ransomware in a honeypot

```
1: function TRAPRANSOMWARE(directory, numTotalConcurrentDummyFiles, sizeOf-
   DummyFiles, numDummyFilesPerDirectory)
2:     currentDirectory = getRandomSubdirectory(directory)
3:     numDummyFilesCreated = 0
4:     nextDirectory = createNewSubdirectory(directory)
5:     while killProcessIsRunning() == true do
6:         if getAmountOfDummyFiles() < numTotalConcurrentDummyFiles then
7:             if numDummyFilesCreated mod numDummyFilesPerDirectory == 0 then

8:                 currentDirectory = nextDirecotry
9:                 nextDirectory = createNewSubdirectory(currentDirectory)
10:            end if
11:            createDummyFileInGivenDirectory(currentDirectory)
12:            numDummyFilesCreated = numDummyFilesCreated + 1
13:        end if
14:        for file in getAllEncryptedDummyFiles() do
15:            delete(file)
16:        end for
17:    end while
18: end function
```

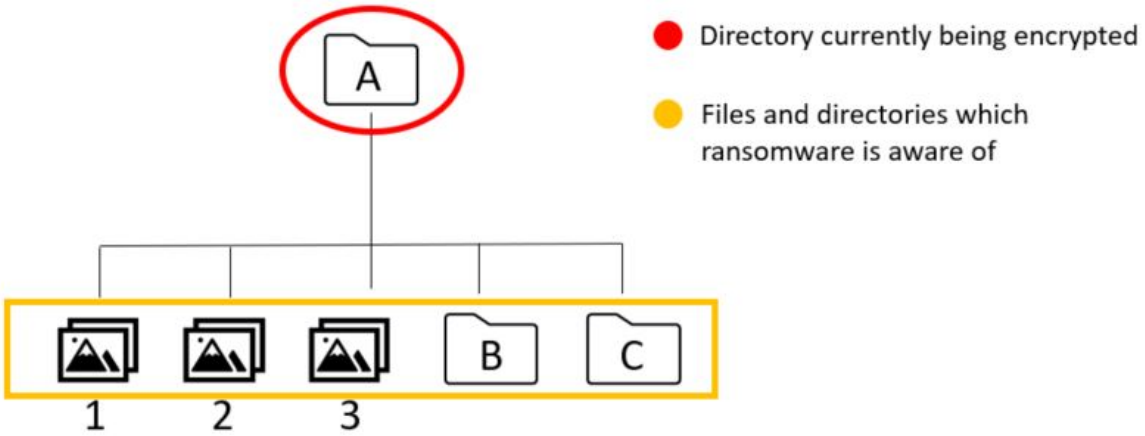


Figure 5.4: Directory Tree

ransomware steps into the sub-directory (B), but can no longer be trapped because there exists no sub-directory (D) into which dummy files could be created. By ensuring that this is the case, this MTD Solution is capable of continuously creating new dummy files in new sub-directories such that the ransomware can be stalled for hours. The amount of time for which this MTD Solution could theoretically stall the ransomware is only limited by the ever increasing path name of the sub-directories which cannot exceed a certain character limit (4096 characters on most Linux distributions) before operations on files cause errors [31].

This desired property is trivially assured by passing a parameter for numTotalConcurrentDummyFiles that is at least the double of the value specified for numDummyFilesPerDirectory. This works because at any given time there are as many unencrypted dummy files as specified by numTotalConcurrentDummyFiles which means that if a dummy file is encrypted a new one is created. And since every x dummy files the MTD Solution moves into a new sub-directory and creates a sub-directory within the previous one(see Algorithm 4, lines 7 to line 10) this means that as soon as a dummy file is encrypted in the current directory, a new dummy file will be created two sub-directories deeper.

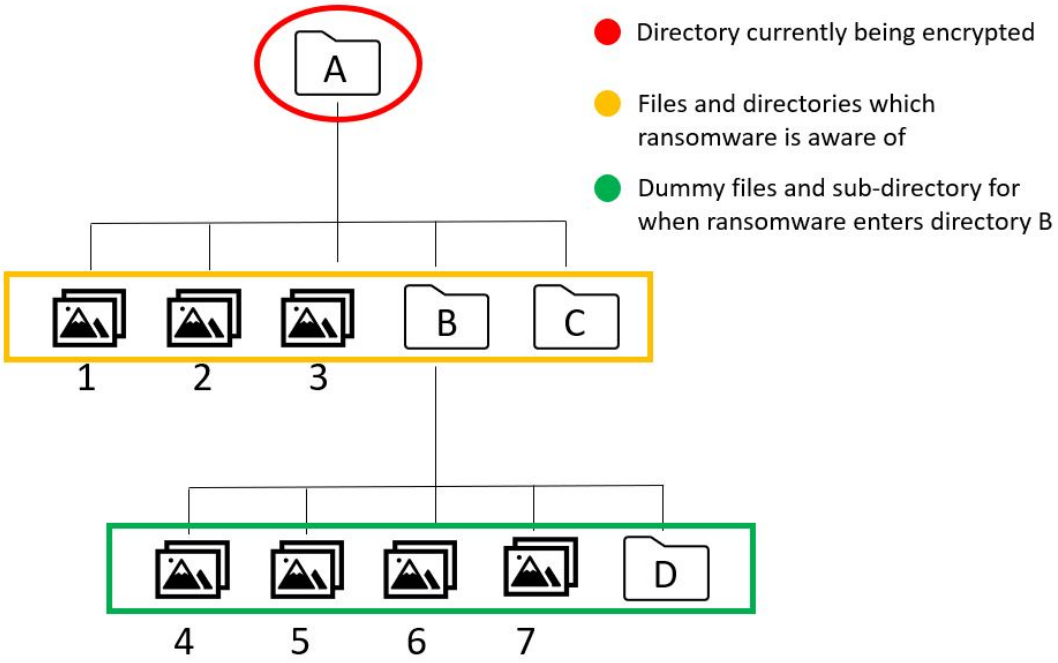


Figure 5.5: Directory Tree With Dummy Files

Another issue that needs to be taken into consideration is the circumstance that disk space on a Raspberry Pi 4 may be scarce, meaning that just creating dummy files without deleting them as well after they have been encrypted, can fill up the remaining disk space pretty quickly. This MTD approach accounts for this issue by being able to limit the amount of concurrent dummy files that are on the system at any given time. This is done by keeping track of the dummy files and deleting them once they are encrypted (see Algorithm 4, lines 14 to line 16). New dummy files are only created (see Algorithm 4, lines 14 to line 16), if there are less unencrypted dummy files than the specified maximal amount of concurrent dummy files (see Algorithm 4, line 6). This means, that new dummy files are only created if old ones are actively being encrypted. Furthermore, considering the disk space issue, this MTD approach also allows to define the size of the dummy files that are created.
Lastly, this MTD Solution is configured to run simultaneously to the KillProcess script. Since this process can theoretically run for hours at a time, only limited by the length of the path name of newly created dummy files, there needs to be a criteria on which its execution stops. Therefore this script only runs as long as the KillProcess script is running

in parallel. Once the encrypting process is killed, this MTD Solution's script terminates itself (see Algorithm 4, line 5).

5.3.2.4 Implementation

This MTD Solution is implemented using Python. For it to run it has the following two dependencies that are not standard Python libraries: psutil, setproctitle. So when adding the corresponding script to the Raspberry Pi it is necessary to make sure that these two dependencies are installed either in a virtual environment or the main Python environment.

5.3.3 Keeping Critical Data Safe

An alternative to trying to keep data safe by trapping the ransomware as shown in the section above, is to concentrate on keeping critical data safe in a targeted manner instead. This MTD Solution aims to keep predefined data within specified directories safe by changing their file extensions.

5.3.3.1 WHAT

In this MTD Solution the moving parameter is the file extensions of files, that are supposed to be kept safe from encryption. Since this changes the presentation of the data on the system, this is also a data based MTD approach.

5.3.3.2 WHEN

Similar to the MTD Solution described above, this MTD approach is also intended to be run while a crypto ransomware is encrypting files, since it is inteded to keep files safe from encryption.

5.3.3.3 HOW

This MTD Solution works by using the fact that crypto ransomware usually only encrypts preconfigured file types (see Algorithm 3, from line 5 to 7) to its advantage. The crypto ransomware decides which files to encrypt by checking the file extension of the files. This opens up the possibility to use the file extensions as a moving parameter similarly as proposed in [12]. In other words: This MTD Solution works by changing the file extensions of files which the system administrator sees as critical to some randomly generated pseudo file extensions.
To help understand how this MTD Solution works a flowchart is provided in Figure 5.6. For the script to work three parameters need to be specified, first the directory in which

the files that should be secured reside, second a list of file types (indicated by their extension) that should be remapped to pseudo file extensions and third a boolean flag indicating if the files in the subdirectories of the specified directory should also be remapped in a recursive manner. The latter two options are given, so that one can gain the time advantage against the encrypting crypto ransomware. Since changing the file extensions is done while the ransomware is encrypting files, this can potentially create a situation where the files that the script is trying to secure can be encrypted before the script is able to change their file extensions. Therefore minimizing the amount of overall files and directories to secure, can be crucial when it comes to gaining a time advantage. If no file types were specified, the MTD Solution uses a default set of file extensions which it considers for remapping.
The script works by first creating a map/directory mapping from the extensions that should be replaced to a random string acting as a pseudo extension (see Algorithm 5, lines 4 to 6). The pseudo extensions consist of an alphanumeric string that is randomly generated and checked against a list of known valid file extensions and already assigned pseudo extensions, to ensure that it is neither a valid file extension itself or a pseudo extension which another real extension already maps to. The latter is done to ensure a 1:1 mapping (between the real file extensions and the pseudo file extensions) and therefore avoid collisions such that once the encrypting process is killed the pseudo extensions can be easily replaced with their real counter part again.

Algorithm 5 Simplified pseudo code algorithm for changing file extensions of files

```
1: function changeFileExtensions(directory, listOfExtensionsToChange, doRecursion, mapForExtensions)
2:     if mapForExtensions == null then
3:         mapForExtensions = createEmptyMap()
4:         for extension in listOfExtensionsToChange do
5:             mapForExtensions.put(extension, createRandomExtension)
6:         end for
7:     end if
8:     listOfFilesInDirectory = getFilesOfDirectory(directory)
9:     for file in listOfFilesInDirectory do
10:        if getExtension(file) in listOfExtensionsToChange then
11:            changeFileExtenstion(file, mapForExtensions.get(getExtension(file)))
12:        end if
13:    end for
14:    if doRecursion == True then
15:        listOfSubDirectories = getSubDirectories(directory)
16:        for subDirectory in listOfSubDirectories do
17:            changeFileExtensions(subDirectory, listOfExtensionsToChange, doRecursion, mapForExtensions)
18:        end for
19:    end if
20: end function
```

In a next step, a list of all the files within the specified directory is retrieved (see Algorithm 5, line 8). For each file it is then checked if the file is one of the given file types

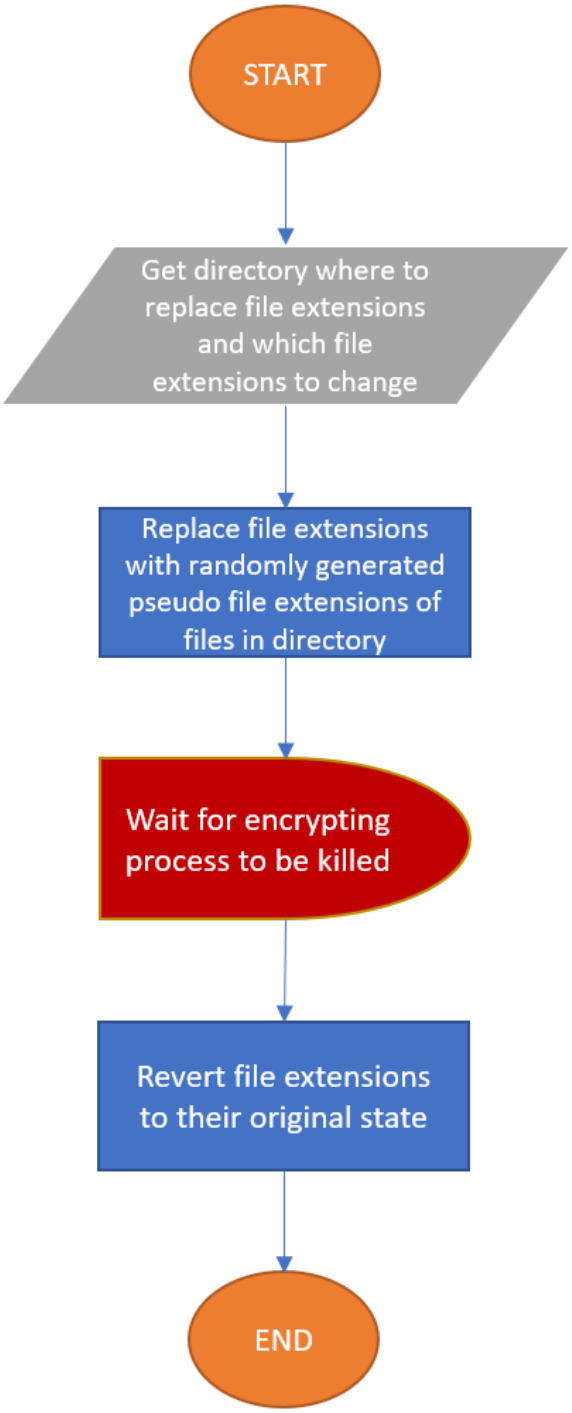


Figure 5.6: Simplified Flowchart Of MTD Solution That Changes File Extensions

that should be remapped to a random extension see (see Algorithm 5, line 10) and if so the file extension of said file is replaced with the corresponding pseudo extension (see Algorithm 5, line 11).
If the parameter doRecursion was set to true the function changeFileExtensions() is called for all the subdirectories of the given directory, while also providing the already generated dictionary/map for the file extensions for reuse (see Algorithm 5, lines 14 to 19).

5.3.3.4 Implementation

This MTD Solution is implemented using Python. For it to run, it has the following two dependencies that are not standard Python libraries: psutil, setproctitle. So when adding the corresponding script to the Raspberry Pi it is necessary to make sure that these two dependencies are installed either in a virtual environment or the main Python environment.

5.3.4 User-Level Rootkits

In this section an MTD Solution will be presented which is capable of dealing with both of the scenarios mentioned in Section 2.3.3. This section will first be looking at the scenario in which /etc/ld.so.preload remains linked, but is appended with untrustworthy shared libraries/objects as a way to install the rootkit. It will also considered that /etc/ld.so.preload could potentially be hidden from view and not directly accessible. Secondly this section will be looking at the scenario in which /etc/ld.so.preload was unlinked and another file which lists the path to the untrustworthy shared libraries/objects was linked in its place.

5.3.5 WHAT

This MTD Solution relies on changing the runtime environment, specifically by using LD_PRELOAD to preload legitimate C libraries in order to avoid meddling from the rootkit. This makes it possible to sanitize /etc/ld.so.preload or to unlink the file which replaced it from the dynamic linker without having to worry about accessing the necessary system files. So this MTD Solution works by changing the runtime environment.

5.3.6 WHEN

This MTD approach is meant to deployed after the rootkit has been installed. This is due to the fact that this MTD approach aims to uninstall/disable the rootkit.

5.3.7 HOW

To help understand how this MTD Solution works a flowchart is provided in Figure 5.7. When it comes to accessing hidden files, it is possible to make use of the fact that specifying shared libraries/objects in /etc/ld.so.preload is not the only available option to preload shared libraries/objects. When running an application/command it is possible to tell Linux to run the application/command with some shared libraries preloaded. This is done by calling "LD_PRELOAD=example.so" as a prefix to the command (which starts the application). In that case the libaries specified in example.so would be preloaded before running the application/command.

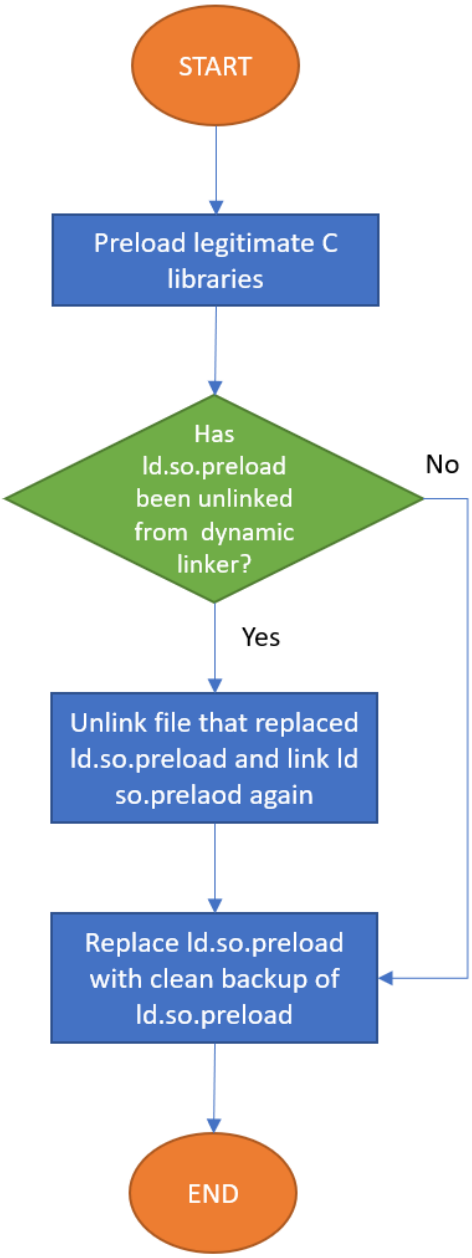


Figure 5.7: Simplified Flowchart Of MTD Solution That Removes User-Level Rootkits

Since LD_PRELOAD is used to define shared libraries/objects that should be preloaded on an application level, it takes precedence over the preloaded shared libraries specified through /etc/ld.so.preload that are preloaded on a system wide level [16]. This means that to sanitize /etc/ld.so.preload it is possible to just run a script which replaces /etc/ld.so.preload with a known good/sanitized version of ld.so.preload with the prefix "LD_PRELOAD=/lib/arm-linux-gnueabihf/libc.so.6" (see Algorithm 6, line 7). By using "LD_PRELOAD=/lib/arm-linux-gnueabihf/libc.so.6" it is guaranteed that the legitimate C libraries are loaded and therefore any operations accessing and writing to files should work as intended again. Therefore any hiding of files or disabled access to files is negated by using the same approach that the rootkit takes to hijack said C libraries in the first place.

Algorithm 6 Simplified pseudo code algorithm for removing/disabling rootkits

```
1: function removeRootkit
2:     if hasLdSoPreloadBeenUnlinked then
3:         stringToReplace = getStringByWhichLdSoPreloadWasReplaced
4:         replaceStringInLdSo(stringToReplace, "/etc/ld.so.preload")
5:         replaceLdSoPreloadWithBackupLdSoPreload()
6:     else
7:         replaceLdSoPreloadWithBackupLdSoPreload()
8:     end if
9: end function
```

As stated above a sanitized version of ld.so.preload is needed. This sanitized version should ideally be a backup of the original untempered with ld.so.preload file. Considering that this MTD approach relies on a backup file this introduces its own risks though, since the /etc/ld.so.preload file can gain new entries if some applications are installed that rely on preloading. This would necessitate creating a backup of the known good/sanitized ld.so.preload file every time the system gets any major updates or when any major applications were installed to make sure that using this MTD Solution does not end up removing the rootkit, while simultaneously breaking the system. But given the scenario in which the Raspberry Pi 4 is being used as an ElectroSense spectrum sensor and for nothing else, it is not very likely that the contents of /etc/ld.so.preload would change during its normal usage which suggests that the MTD Solution presented here might be a feasible option to keep the spectrum sensor safe from user-level rootkits.
We also need to consider the case in which /etc/ld.so.preload has been unlinked from the dynamic linker and a different file has been linked in its place as the file which lists the shared libraries/objects which the dynamic linker should preload. The relevant file in which /etc/ld.so.preload is specified/linked to is /lib/arm-linux-gnueabihf/ld-2.24.so. The file /etc/ld.so.preload is specified in /lib/arm-linux-gnueabihf/ld-2.24.so exactly once. By replacing the string "/etc/ld.so.preload" within the file with any other file path /etc/ld.so.preload is unlinked and the other file is linked to the dynamic linker. Knowing where the string "/etc/ld.so.preload" should occur, it is possible to test if it is present and if it is not present the MTD Solution's script knows that a rootkit is installed and that /etc/ld.so.preload was unlinked. In that case the script changes /lib/arm-linux-gnueabihf/ld-2.24.so to include the string "/etc/ld.so.preload" again by simply overwriting the string that replaced it (see Algorithm 6, lines 2 to 4). Just to be sure the script than

also replaces /etc/ld.so.preload with the backup version, since it is possible that it was tampered with as well (see Algorithm 6, line 5). This is enough to break rootkits that work by unlinking /etc/ld.so.preload and linking to another file of their own which points to their malicious shared libraries.
By checking if the string "/etc/ld.so.preload" is present in /lib/arm-linux-gnueabihf/ld-2.24.so the MTD Solution can also establish with which type of user-level rootkit it is dealing with and then select the corresponding approach to take (see Algorithm 6, line 2).

5.3.8 Implementation

The MTD approach described above is implemented in Python. It only uses standard Python libraries as its dependencies. The MTD script additionally necessitates that sed is installed on the Raspberry Pi, but sed is usually preinstalled on Raspberry OS. The only other prerequisite needed for it to work properly is that a backup copy of /etc/ld.so.preload has been created and that said backup has been renamed to backupLSP and moved to the directory in which the MTD approach's script resides.

Chapter 6

Evaluation of MTD Framework

6.1 Methodology

In this chapter the MTD Framework and the MTD Solutions presented in Chapter 4 and 5, will be evaluated against real malware. To test the different MTD Solutions the Raspberry Pi 4 acting as an ElectroSense spectrum sensor will be infected with representative malware of each attack type covered in Section 2.3. Then using the MTD Deployer Client the MTD Framework will be triggered to deploy the specific MTD Solution which is appropiate for the currently running/installed malware. Then the Raspberry Pi 4 will be checked manually to see if the malware is still in a working state or if the malware was disabled/disrupted by the MTD Solution deployed by the MTD Framework. So the main criteria by which the MTD Framework and more specifically the MTD Solutions will be evaluated is by a binary answer to the question if they actually are able to disrupt/disable the malware.
Where it makes sense performance data (CPU usage, memory usage) will be gathered. While some of the implemented MTD Solutions run for a short time (milliseconds to seconds range), others can run for multiple minutes and are more resource intensive on the system. For the former evaluating their performance impact (CPU usage, memory usage) is not necessary and therefore not done, for the latter performance data is gathered and provided in the evaluation.
All performance data regarding CPU and memory usage presented in this section was gathered by running nmon (Nigel's performance Monitor for Linux) on the Raspberry Pi. This tool allows for real time monitoring of system resources, but it also allows for system resources to be monitored over time. When monitoring system resources over time, it is necessary to specify a sample rate in seconds as well as the total amount of desired samples. This could for example look like this: "nmon -f -s1 -c600". The "-f" signifies that nmon should be run in batch mode, meaning that it should record the data gathered to a file in a csv format, "-s1" specifies the time interval (in seconds) that should elapse between the samples while "-c600" specifies the total amount of samples that should be recorded. In this case, nmon would be running for 10 minutes (600 seconds) and create 600 individual samples/snapshots depicting the system resources at the time they were created.

For the Command & Control based MTD Solution additional data was gathered to measure the impact of the MTD Solution on the network throughput of the system. This was tested by measuring how many bytes the spectrum sensor sends to the ElectroSense platform during normal use for a given time interval (20 minutes), then measuring how many bytes the spectrum sensor sends if the MTD Solution is deployed once in the same timespan and finally also measuring how many bytes are sent if the MTD Solution is deployed once per minute. This data was gathered using iftop by using a filter for the ElectroSense server's IP address and specifying that it should run for 20 minutes (iftop -i eth0 -f "dst net 194.209.200.16" -t -s 1200 > log.txt). The goal of this experiment is to determine if there is a significant loss in data throughput due to the migration of the IP address.

6.2 Results

6.2.1 MTD Framework Running in the Background

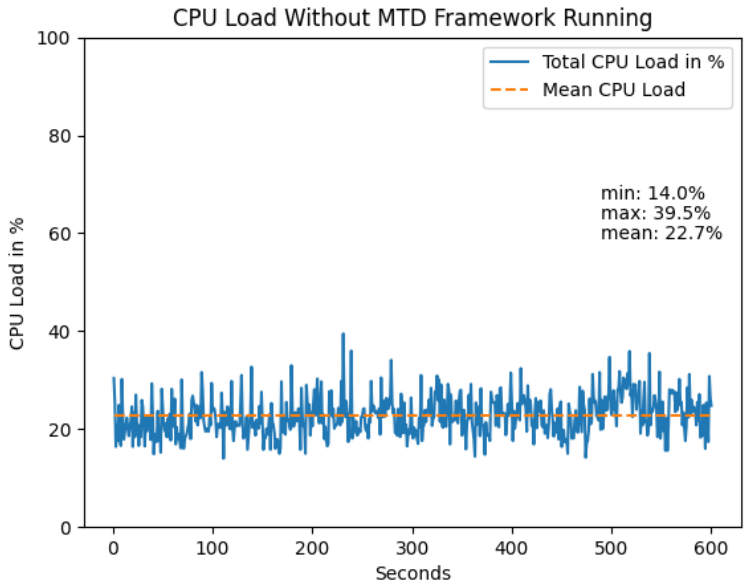
Since the MTD Framework is intended to be running at all times as a background service that listens for attack reports, it is necessary to consider how the service running in the background affects the Raspberry Pi's system resources. For this reason the overall CPU load and the total memory usage of the system were monitored with and without the MTD Framework's service running in the background for 10 minutes each.
As can be seen in Figure 6.1a and 6.1b the overall CPU usage is not significantly higher if the MTD Framework is running in the background and actively listening for attack reports. Similarly Figure 6.2a and 6.2b does not exhibit a significant increase in memory usage.

6.2.2 Command & Control Based Malware

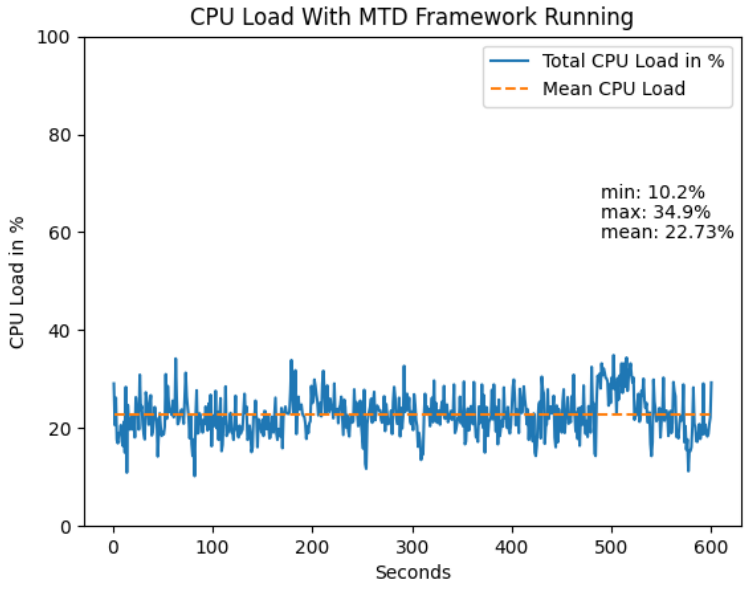
In the following section, the MTD Solution described in Section 5.2 will be tested against real CnC based malware. It is noteworthy to mention that migrating to another IP address results in a short time without internet connectivity. From the testing executed here, it is safe to say that the spectrum sensor was never offline for longer than a few seconds at best. Before looking at specific malware this section will investigate if the above mentioned short loss in internet connectivity constitutes a major problem.

6.2.2.1 Throughput to the ElectroSense Platform

As already mentioned in Section 6.1 and above, this section will analyze how much the data throughput to the ElectroSense platform suffers when the MTD Solution that changes the IP address is triggered. To test how the MTD Solution affects the data throughput to the ElectroSense platform the bytes sent to the ElectroSense platform were monitored for 20 minutes with and without deploying the MTD Solution (see Figure 6.3). When the



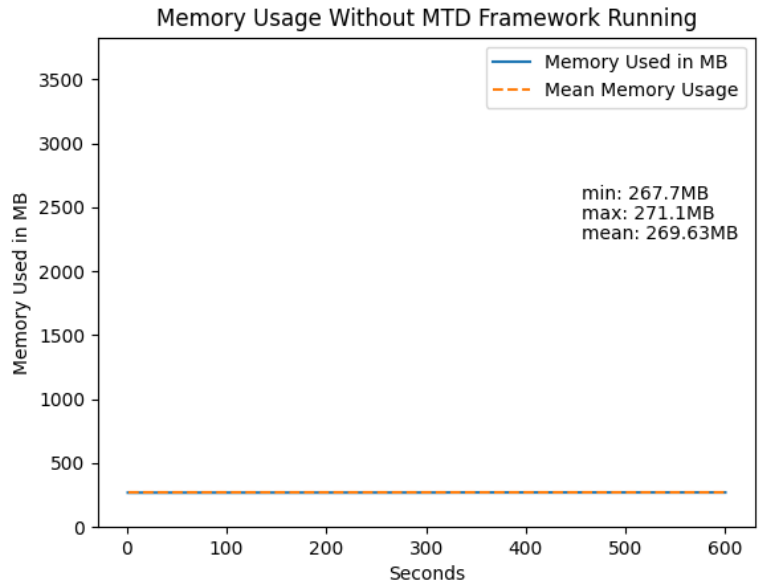
(a) CPU Load Without MTD Framework Running in Background



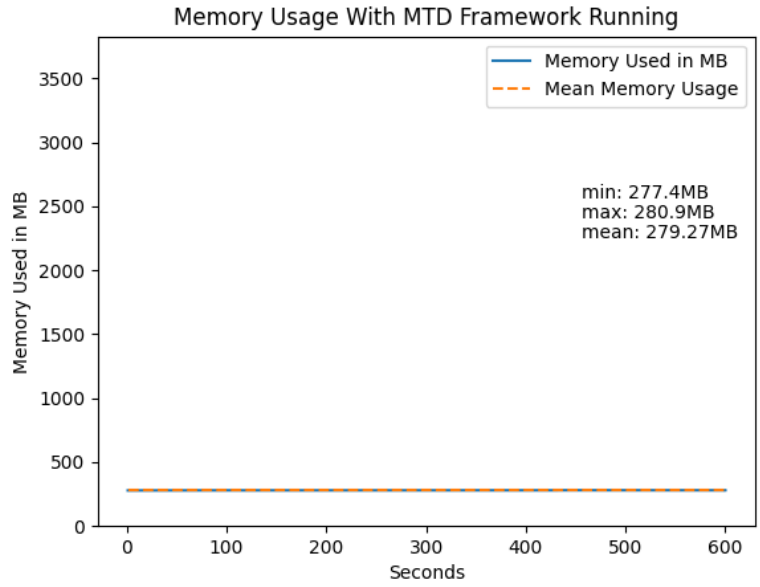
(b) CPU Load With MTD Framework Running in Background

Figure 6.1: Comparison of CPU Load With and Without MTD Framework Running

MTD Solution was not deployed the spectrum sensor had sent 22.2 MB of data to the ElectroSense platform averaging out to 1.11 MB per minute. When the MTD Solution was deployed once during the entire time span ,the data sent to the ElectroSense platform was 22.1 MB (reduction by 0.5%) averaging out to 1.105 MB per minute. Lastly, when deploying the MTD Solution once per minute (20 interruptions to the service which cause downtime) the Raspberry Pi 4 had only sent 19.4 MB (reduction by 12.6%) of data to the ElectroSense platform averaging out to 0.97 MB per minute. Therefore we can conclude that if the MTD Solution is deployed in a reactive manner as intended it would barely



(a) Memory Usage Without MTD Framework Running in Background



(b) Memory Usage With MTD Framework Running in Background

Figure 6.2: Comparison of Memory Usage With and Without MTD Framework Running

affect the data throughput since it is only triggered once, but if the MTD Solution would be used to preemptively change the IP address every minute (for example to invalidate reconnaissance attacks), this would cause a significant loss in throughput. In conclusion, when the MTD Solution is used in a reactive manner to disrupt a connection to a CnC server, the downtime does not cause the throughput to significantly decrease and can therefore be considered negligible.

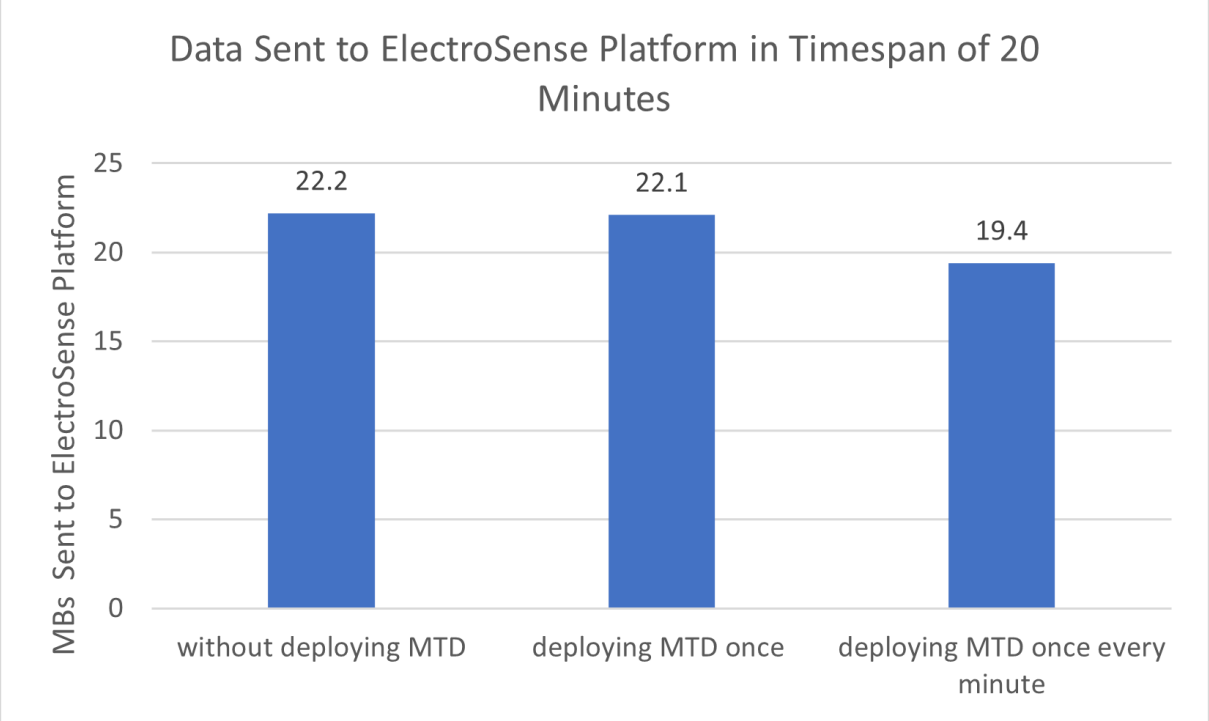


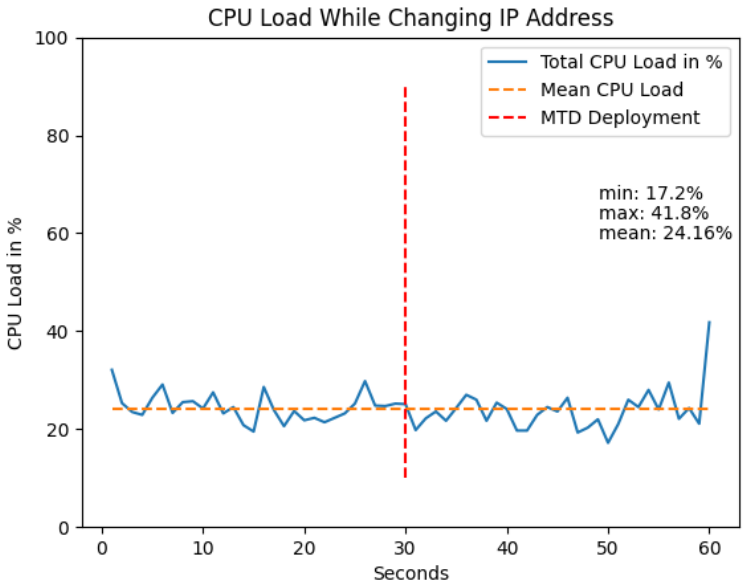
Figure 6.3: Data Sent to ElectroSense Platform in Timespan of 20 Minutes

6.2.3 Performance

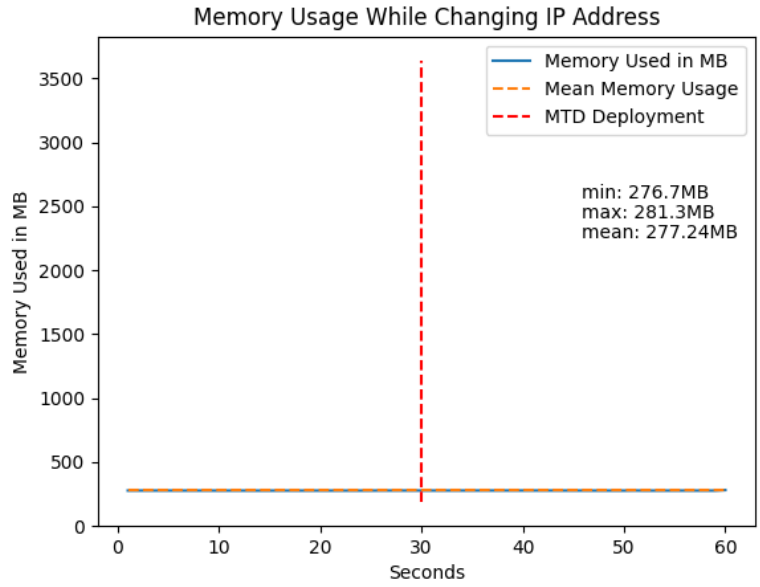
In Figure 6.4 it is possible to see that deploying the MTD Solution does not significantly affect the CPU load or the memory usage. The MTD Solution was deployed 30 seconds after the monitoring started (MTD Deployment marked by red vertical line).

6.2.3.1 httpBackdoor by SkryptKiddie

httpBackdoor by SkryptKiddie [32] is a simple http based backdoor which is implemented in Python. It consists of a single scipt httpBackdoor.py which sets up a http server which listens for requests. When this malware is running, it allows a malicious actor to run commands on the Raspberry Pi from another machine by sending POST requests to the IP address of the Raspberry Pi on a preconfigured port.
After the MTD Solution described in Section 5.2 is executed, the httpBackdoor malware still stays operational, but the attackers commands do not reach their intended goal any longer. At this moment the attacker has no information about if the backdoor is still operational. If the attacker is within the same network as the Raspberry Pi he could regain access to the backdoor by randomly guessing the new local IP address, but it is also imaginable that the attacker would assume that httpBackdoor's script has stopped running and therefore stop his attempts to send commands to the device. If the attacker reached the Raspberry Pi from an outside network through the Raspberry Pi's public IP address, this would imply that there was some port forwarding set up such that the commands could reach the Raspberry Pi in the first place. In that case changing the IP



(a) CPU Usage While MTD to Change IP Address is Deployed



(b) Memory Usage While MTD to Change IP Address is Deployed

Figure 6.4: CPU & Memory Usage While MTD to Change IP Address is Deployed

address does also invalidate the port forwarding such that the malware's http backdoor becomes unreachable from outside the network.

6.2.3.2 backdoor by jakoritarleite

backdoor by jakoritarleite [33] is a backdoor implemented in Python which allows an attacker to copy files from the victim device to the attacker device. It can also access

the shell of the victim device and therefore take full control over the victim device. The malware consists of two parts: client.py which is run on the victim machine (Raspberry Pi) and server.py which is run on the attacker's machine. On startup the client.py script tries to establish a connection to the attacker's machine and once a connection has been established it goes into a passive mode and listens for commands. On startup server.py waits for a device to connect to it, once a connection is established it allows the attacker to steal data from the victim machine or access the shell of the victim machine.
After the MTD Solution described in Section 5.2 is executed, the client.py and the server.py script do not crash and remain running. While on a surface level both scripts still seem to be operational, once a command is sent from the CnC server there are two possibilities of what can happen. If the CnC server is on the same network as the victim machine this leads to a crash of server.py. If the CnC server was accessed through the internet sending commands does not result in a crash, but the commands do not reach their destination in this case either, instead server.py gets stuck waiting for a response from the Raspberry Pi. Therefore the connection between the CnC server and the spectrum sensor is irreversibly disrupted.
At this point restarting server.py does not reestablish a connection, since client.py is unaware that the connection was lost in the first place. This results in a disrupted connection from which the attacker cannot recover without restarting client.py on the victim machine which would necessitate access to the Raspberry Pi.

6.2.3.3 The Tick by NCC Group

The Tick by NCC Group [34] is a botnet which allows an attacker to control multiple devices from a server with an extensive suite of features. The malware consists of two main parts a binary ticksvc which is run on the victim device and a Python script tick.py which is run on the CnC server. Once the binary ticksvc is started on the victim machine, it will establish a connection to the server running tick.py.
After the MTD Solution described in Section 5.2 is executed, the binary ticksvc keeps running. The same goes for the tick.py on the server side. When the attacker than tries to use the bot through tick.py, the bot's status eventually changes to "gone". The result is the same regardless if the CnC server is hosted on the Internet or if it is hosted in the same local network as the Raspberry Pi. At this point the binary running on the target machine will not try to reestablish a new connection, which results in the previously established connection being irreversibly severed without an easy way to reestablish said connection, if the attacker cannot rerun the binary on the Raspberry Pi which would necessitate access to the Raspberry Pi.

6.2.3.4 BASHLITE

BASHLITE [35] is a family of botnet malware which is designed to be used for DDoS attacks. BASHLITE also consists of a client and a server script. Similarly to the previous CnC based malwares, BASHLITE initiates a connection through the client script on the victim machine. Once a connection has been established, this version of BASHLITE starts printing PONG on the victim's device every 60 seconds by sending a command to

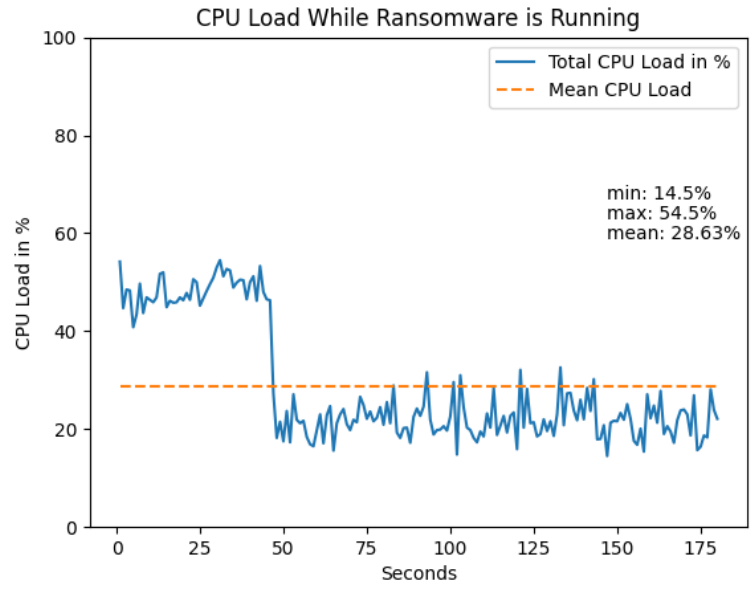
the victim machine.
After the MTD Solution described in Section 5.2 is executed, the client script and the server script remain both running, but the printing of "PONG" stops because the connection between the CnC server and the client has been disrupted. Just like with the other CnC based malwares that have a client on the victim machine which establishes a connection to the CnC server, BASHLITE does not recover from this state and the connection is irreversibly disrupted. To reestablish a connection it would be necessary to rerun the client script, which would necessitate access to the Raspberry Pi.

6.2.4 Crypto Ransomware

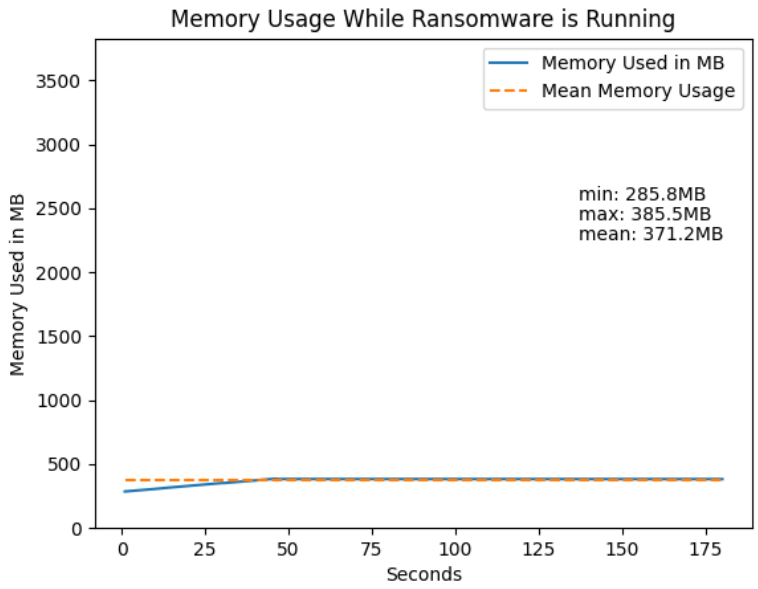
In this section the two MTD Solutions proposed in Section 5.3 will be tested against Ransomware-PoC by jimmy-ly00/Ransomware-PoC [36]. To test the proposed MTD Solutions a test directory with dummy data was created. This test directory is 4 directories deep and contains roughly 100 MB of dummy data (64 files). The test directory used for these tests can be found in this thesis' accompanying GitHub Repository [29] under the Ransomware section. To test the two MTD Solutions the test directory was moved to the Raspberry Pi and the ransomware was told to encrypt the test directory, while simultaneously an MTD Solution was deployed.
Figure 6.5a shows how the ransomware running affects the CPU usage. The first 50 seconds of this graph show the Ransomware running until it has fully encrypted the directory it was told to encrypt. During this time the CPU usage hovers around 50%. The rest of the graph shows how the CPU usage drops down to a normal level again and remains there. When it comes to the memory usage (Figure 6.5b) it can be observed that the memory usage rises while the ransomware is running. But even after the ransomware has stopped its execution the memory usage stays at the raised level. These measurements will be used as a baseline to judge the two MTD approaches when it comes to their resource consumption.

6.2.4.1 Honeypotting the Encryptor

For the first MTD Solution the crypto ransomware was run on the test directory, while simultaneously the MTD Solution that created dummy files to honeypot the encrypting process was started in the same directory. The ransomware used for these test randomly selects in what order it steps into sub-directories. Since the trapping of the encrypting process can only happen through a subdirectory as explained in Section 5.3.2, the results may vary from test run to test run. The test was repeated five times. In all of the five test runs the encrypting process got caught in the honeypot, meaning that in the end it was left encrypting the dummy data created by the MTD Solution. Similarly the KillProcess script was able to identify and kill the encrypting process in every run, which would usually happen a few seconds after the encrypting process started encrypting dummy files. Non the less the number of files that were encrypted that were not dummy files created by the MTD Solution varied from run to run as would be expected:



(a) CPU Usage While Ransomware is Running



(b) Memory Usage While Ransomware is Running

Figure 6.5: CPU & Memory Usage While Ransomware is Running

Run 1: 7 out of 64 files encrypted

Run 2: 20 out of 64 files encrypted

Run 3: 7 out of 64 files encrypted

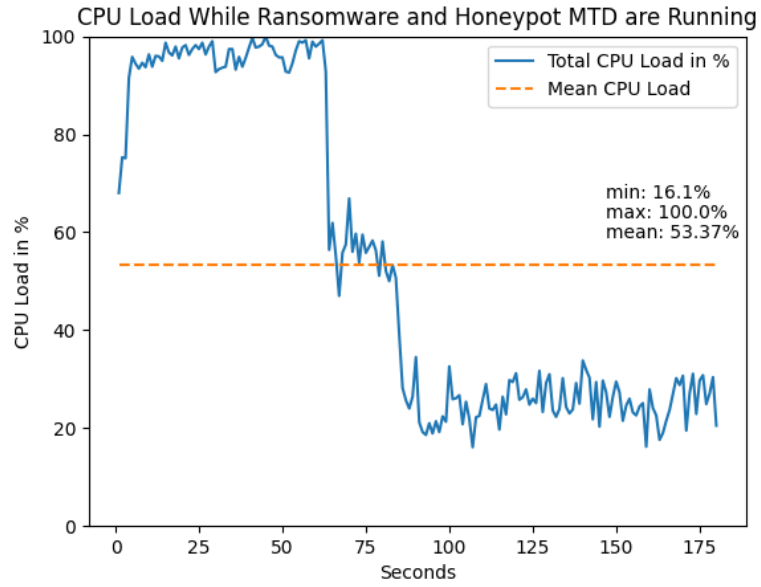
Run 4: 35 out of 64 files encrypted

Run 5: 36 out of 64 files encrypted

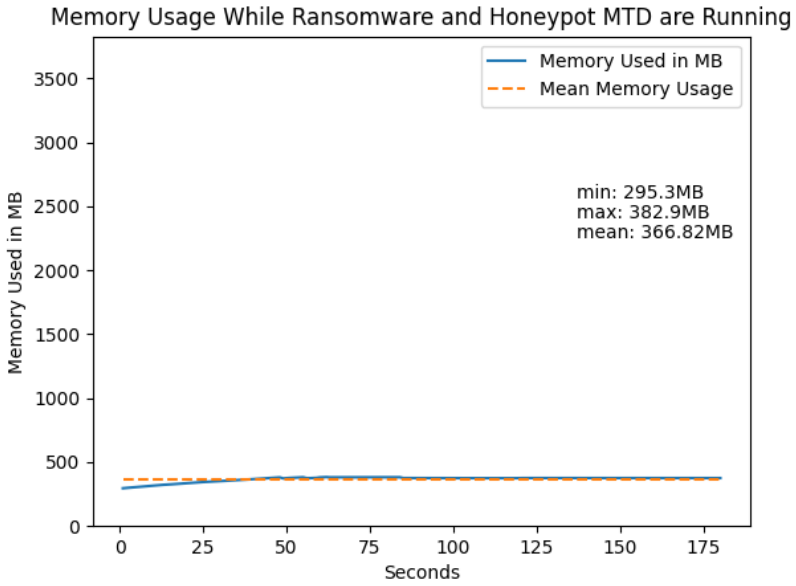
After five runs a mean of roughly 33% of the files that were supposed to be protected were encrypted leaving roughly 67% unecrypted. It should be noted that this percentage is not representative and that the amount of files that are encrypted when the MTD Solution is deployed highly depends on the given directory's structure. Nonetheless it seems that this MTD Solution is capable of consistently trapping the crypto ransomware and limit the damage it is able to do.
Looking at Figure 6.6a it is possible to gauge the difference in resource consumption when the MTD Solution is running compared to when only the ransomware is running. The first thing that is noticeable, is that for the first minute the CPU is at full load. This is due to the fact that the ransomware, the script responsible for identifying and shutting down the ransomware and the MTD Solution are all running simultaneously. It should also be mentioned, that it is visible that the ransomware clearly got stuck, as it would have finished at the 50 seconds mark else wise as can bee seen in Figure 6.5a and 6.7a. After the ransomware script is identified and killed the first drop in the graph happens and after the other two processes are shutdown as well the CPU usage normalizes again. When looking at the Figure 6.6b the memory consumption does not differ significantly from the baseline.

6.2.4.2 Keeping Critical Data Safe

For the second MTD Solution a sub-directory (specified through the config file) of the test directory was specified as the directory in which files should be kept safe. This would often cause the ransomware script to crash, because files that it was trying to access had changed their file extensions and therefore their name. This would result in a FileNotFoundError being thrown but not handled. Since it is imaginable that other crypto ransomware would account for such cases in which a file might get deleted or renamed before it is able to encrypt them, the ransomware script was slightly adjusted to catch such errors and just skip the files it could not encrypt.
This test was run five times. In these runs every file in the sub-directory and its sub-directories that was supposed to be kept safe remained safe, meaning that the success rate of keeping the specified directory and the files therein safe was 100%. The rest of the files within the test directory were encrypted as is expected. The KillProcess script was not able to kill the encrypting process because of the limited time for which the encrypting process ran considering that it had less than 100 MB to encrypt and did so quite fast, such that the encrypting process was already finished up before it could be identified. In a real setting the encrypting process would most likely take way longer and would therefore be caught and shutdown, if the MTD Solution in Section 6.2.4.1 is anything to go by. If



(a) CPU Usage While Ransomware and Honeypot MTD Soution are Running



(b) Memory Usage While Ransomware and Honeypot MTD Soution are Running

Figure 6.6: CPU & Memory Usage While Ransomware and Honeypot MTD Soution are Running

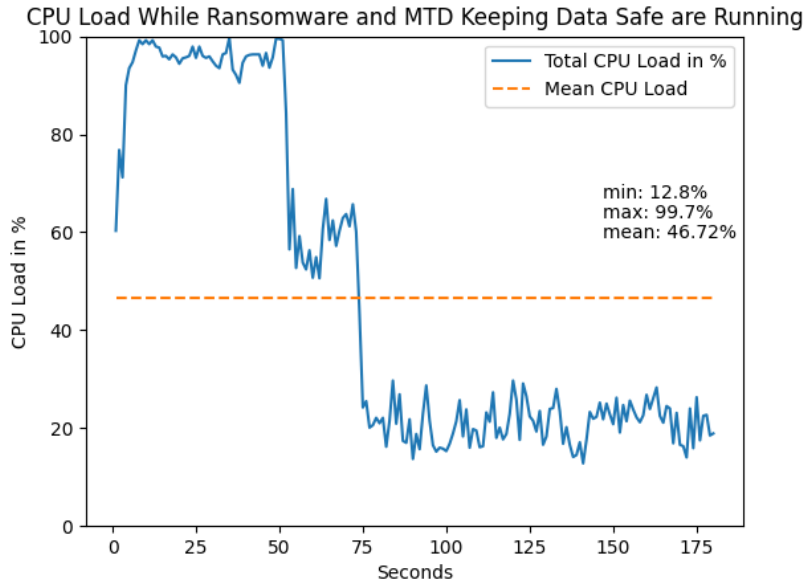
the encrypting process runs for long enough this will result in KillProcess identifying and killing the encrypting process.
Similarly to the previous MTD Solution, this MTD Solution also leads to quite a high CPU load as can be seen in Figure 6.7a. Until the ransomware finishes on its own at the 50 seconds mark the CPU is basically running at full load. After the encrypting ransomware has finished the CPU load drops to roughly 60%. Shortly after the MTD Solutions terminates itself after not being able to identify the encrypting process which leads to a normalized CPU load again. When looking at the Figure 6.7b the memory consumption does not differ significantly from the baseline. In the following section we will evaluate how the MTD approach described in Section 5.3.4 performs against two user-level rootkits that use preloading shared objects as a means of installing a rootkit on the Raspberry Pi. The first one we will look at is beurk, which directly appends /etc/ld.so.preload with the shared libraries/objects that hijacks the desired symbols and has the capability to hide /etc/ld.so.preload if configured to do so. We will also look at bdvl which functions in a similar way, but actually unlinks /etc/ld.so.preload from Linux's dynamic linker and links a different file with a randomly generated name to the dynamic linker and then appends this file with the shared libraries/objects that hijacks the desired symbols.

6.2.4.3 beurk

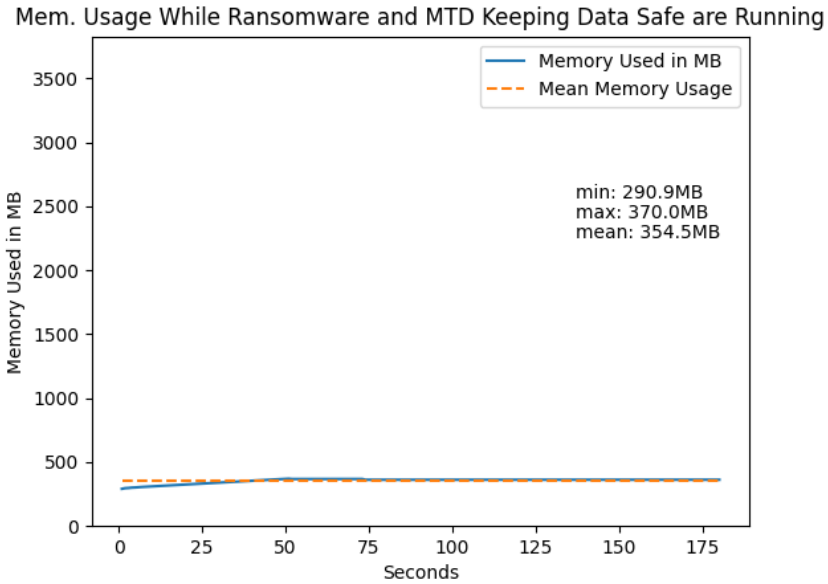
If the MTD Solution is run on a Raspberry Pi 4 acting as a spectrum sensor that has been infected with beurk. The MTD approach disables the rootkit within milliseconds by replacing the manipulated /etc/ld.so.preload with a sanitized backup version of /etc/ld.so.preload. This is the case regardless if beurk was configured to hide /etc/ld.so.preload or not.
In Figure 6.8a it is possible to see that deploying the MTD Solution does not significantly affect the CPU load when beurk is installed. The MTD Solution was deployed 20 seconds after the monitoring started (MTD Deployment marked by red vertical line). The deployment of the MTD Solution does not cause a noteworthy or lasting spike in the CPU load as can be seen in the figure since this MTD Solution only runs for a few hundred milliseconds at most. When looking at the system memory during the same time frame (see Figure 6.8b) we do not see any increased/decreased memory usage before or after the MTD Solution was deployed. Therefore one might conclude that running this MTD Solution to remove beurk does not take up a significant amount of system resources.

6.2.4.4 bdvl

If the MTD Solution is run on a Raspberry Pi 4 acting as a spectrum sensor that has been infected with bdvl. The MTD approach disables the rootkit within milliseconds by unlinking the file which replaces /etc/ld.so.preload and linking /etc/ld.so.preload to Linux's dynamic linker again.
In Figure 6.9a it is possible to see that deploying the MTD Solution does not significantly affect the CPU load when bdvl is installed. Just as with the beurk run the MTD Solution was deployed 20 seconds after the monitoring started (MTD Deployment marked by red

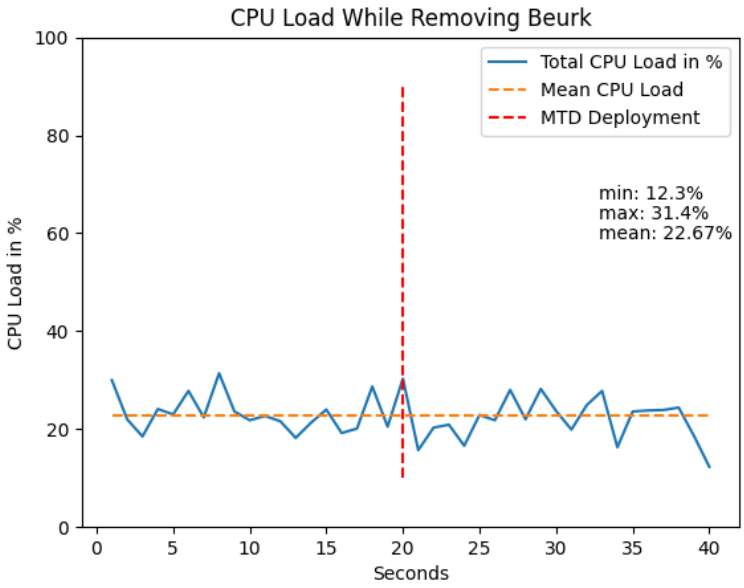


(a) CPU Usage While Ransomware and MTD Keeping Data Safe are Running

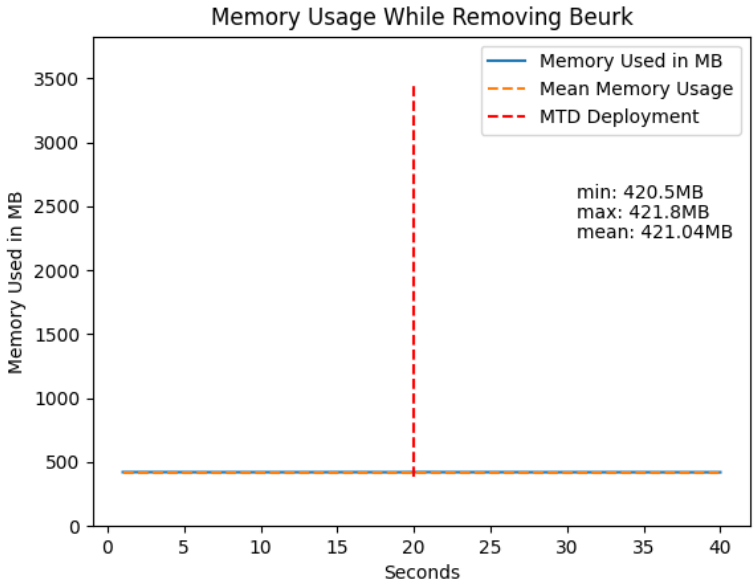


(b) Memory Usage While Ransomware and MTD Keeping Data Safe are Running

Figure 6.7: CPU & Memory Usage While Ransomware and MTD Keeping Data Safe are Running



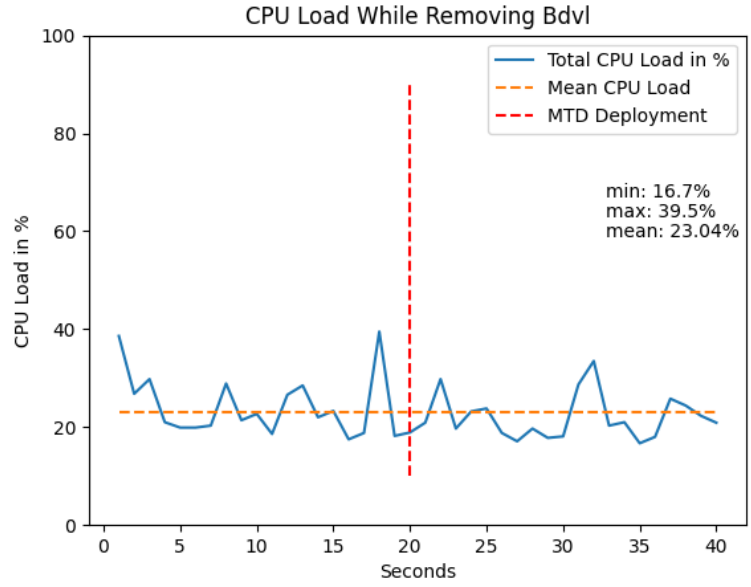
(a) CPU Usage While Beurk Is Removed



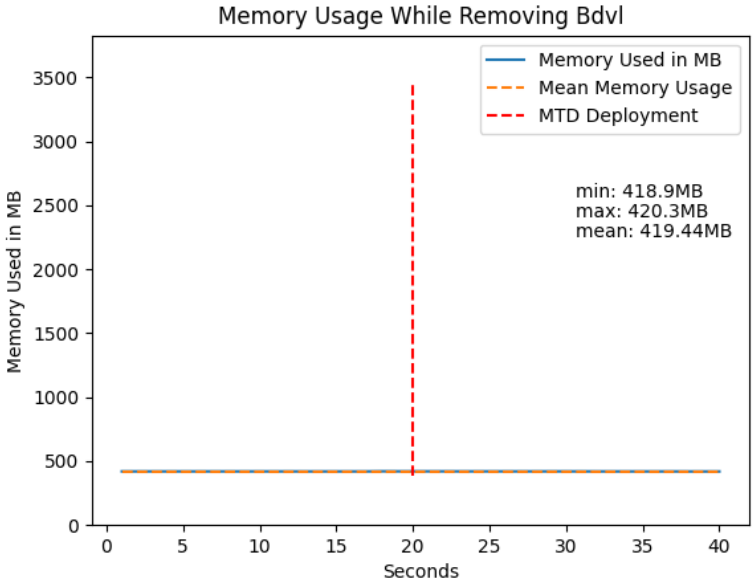
(b) Memory Usage While Beurk Is Removed

Figure 6.8: CPU & Memory Usage While Beurk Is Removed

vertical line). The deployment of the MTD Solution does not cause any significant increase in CPU load. When looking at the system memory during the same time frame (see Figure 6.9b) we also do not see any increased/decreased memory usage before or after the MTD Solution was deployed. Therefore it is safe to assume that running this MTD Solution does not take up a significant amount of system resources regardless if it has to unlink the file which replaces /etc/ld.so.preload (bdvl) or if it only has to replace /etc/ld.so.preload (beurk).



(a) CPU Usage While Bdvl Is Removed



(b) Memory Usage While Bdvl Is Removed

Figure 6.9: CPU & Memory Usage While Bdvl Is Removed

Chapter 7

Discussion

7.1 Interpretation of Results

7.1.1 MTD Solution Against Command & Control Based Malware

Out of five tested CnC based malwares not a single one did reestablish a connection to the CnC server after the MTD Solution which changes the IP address was deployed. This suggests that this novel use of changing the IP address to disrupt an already established connection is quite effective, as long as the attacker/the malware does not account for that possibility (see limitations below). Even better this MTD Solution could in theory also be used to invalidate reconnaissance attacks by periodically changing the IP address.

7.1.2 MTD Solution Against Crypto Ransomware

The two MTD Solutions that try to minimize the amount of data that is encrypted by the crypto ransomware seem to work as expected. The MTD solution which remaps the file extensions to pseudo file extensions in a prespecified directory to keep the files within that directory safe worked in 100% of the test cases, suggesting that it is a strong tool when it comes to keeping specific data safe once ransomware has been found to encrypt data. The only case in which this MTD Solution would not work, is if the ransomware would be encrypting files indiscriminately to their file type/extension or if the ransomware would already have encrypted the files before the attack report is send to the MTD Framework. Similarly the MTD Solution that tries to trap the encrypting ransomware in honeypot directories turned out to be effective in trapping the crypto ransomware. Over five test runs the MTD Solution managed to keep 67% of the data unencrypted. While this may suggest that this MTD solution is also a powerful tool when it comes to generally keeping data unencrypted, it is important to take the limitations of its current implementation into consideration (see limitations below).
The script responsible of identifying and killing the encrypting process worked quite well within the given scenario. In combination with the MTD Solution that creates a honeypot

it was able to identify the encrypting process in 100% of the test cases. The same cannot be said for the cases in which it was run in combination with the MTD Solution which changes the file extensions. This happened in these cases because the ransomware already had finished encrypting the test directory before the script responsible of identifying and killing the encrypting process was able to pinpoint the encrypting process. This was due to the small size of the test directory, but given the limited available space on the Raspberry Pi 4 image, the test directory could not be significantly larger. But as the tests with the honeypotting MTD Solution showed: as long as the ransomware would run long enough the script would eventually identify and kill the encrypting process. In reality it is unlikely that the ransomware would be running for less than a minute.

7.1.3 MTD Solution Against User-Level Rootkits

The MTD Solution responsible for removing user-level rootkits worked flawlessly in our testing. It was capable of removing two different user-level rootkits which work in different ways by preloading legitimate C libraries by using LD_Preload and then either sanitizing /etc/ld.so.preload or unlinking the file that replaced /etc/ld.so.preload from the dynamic linker and linking /etc/ld.so.preload to the dynamic linker again.

7.2 Limitations

7.2.1 MTD Framework

While the MTD Framework currently allows for a DeploymentPolicy to be defined for cases in which more than one MTD Solution are present per AttackType, this DeploymentPolicy can only specify which configured MTD Solution should be launched. The MTD Solution is then launched as it was preconfigured in the config file (with the given prefix and parameters). This limits the possibility of how the MTD Solutions can be deployed, since they only can be deployed in the precise way they were configured and it is not possible to pass dynamically calculated values as arguments/parameters for an MTD Solution.

7.2.2 MTD Solution Against Command & Control Based Malware

While the findings shown in Section 6.2.2 seem to imply that the proposed MTD Solution which changes the IP address is an effective way to disrupt the connection to a CnC server, it is only effective as long as the attacker or the malware do not account for the possibility of a changing IP address or a disrupted connection. It would not be difficult to rewrite the source code of the presented CnC based malware to periodically check if the IP address of the victim device has changed or check by any other means if the connection is not active any longer. In the case that the connection is disrupted, the client script could just reestablish a new connection with the CnC server. In that case the MTD Solution would only incapacitate the connection for a short while until it is reestablished.

7.2.3 MTD Solution Against Crypto Ransomware

While the second MTD Solution which aims to keep critical data safe works quite well according to the conducted testing, the same cannot necessarily be said about the first MTD Solution which aims to trap the encrypting process of the ransomware in a honeypot. For the encrypting ransomware to be trapped it is necessary that the MTD Solution is started from within a sub-directory of the directory which the ransomware is currently encrypting. While this aspect is trivially given in our testing, in a real world setting this would not necessarily apply. Simply guessing that the ransomware has not encrypted a directory yet and creating a honeypot in said directory is not good enough when trying to trap the ransomware's encryptor and would lead to unsatisfactory results, since it is possible that the encryptor might be running for minutes before it even gets trapped.
Also the way in which the encrypting process is identified may work pretty well within the given scenario in which a Raspberry Pi 4 is being used as an ElectroSense spectrum sensor. But the approach taken might not as well on other systems since some assumptions are made (assuming that encrypting process will be the only process using a lot of CPU resources and opening many files) which hold true for the given scenario, but not necessarily in general.

7.2.4 MTD Solution Against User-Level Rootkits

When it comes to MTD Solution that removes rootkits, it should be mentioned that this MTD Solution has two flaws. The first flaw is that it relies on a backup of ld.so.preload. While in the given scenario of the Raspberry Pi 4 acting as a spectrum sensor, it is unlikely that ld.so.preload would be changing on a regular basis or at all, it is still not an impossibility. The second flaw is that it assumes that the LD_PRELOAD command has not been messed with by the rootkit. If LD_PRELOAD would be disabled through the rootkit, this would make it impossible to preload legitimate C libraries which would make sanitizing /etc/ld.so.preload impossible if the rootkit was set up to hide the file or block access to it.

7.3 Future Research & Conclusion

7.3.1 Future Research

Given the limitations discussed above the following aspects might be interesting for consideration when it comes to future research:

1. For future research it would be nice to extend the capabilities of what the script specified in DeploymentPolicy does. It would be helpful if it could not only decide which MTD Solution gets deployed, but if it could also optionally decide with which arguments/parameters the MTD Solution gets called. This would be especially useful for MTD Solutions that depend on a path that is passed as an argument. By

adding this functionality the MTD Framework would gain a powerful option which allows deploying MTD Solutions with dynamically calculated arguments.

2. For future research it is also imaginable to look into ways of examining the connections to unknown IP addresses and finding out if they are CnC servers. If they are the communication to these IP addresses could be blocked through blacklisting. Compared to the current MTD Solution this would have the advantage that blacklists could be shared between multiple spectrum sensors. Since the MTD Framework can accept attack reports over the network, it is also imaginable to trigger an update script through the MTD Framework which forces the spectrum sensors to pull the newest blacklist once a new attack happens and the CnC server used in the attack was blacklisted.

3. For future research the MTD Solution which creates a honeypot to trap the ransomware's encryptor could be reworked to work more closely in tandem with the script responsible for identifying and killing the encrypting process (KillProcess). Once KillProcess has settled on a suspicious process it believes to be the ransomware it could tell the MTD Solution to create the honeypot in the file directory in which the suspicious process is currently accessing files. This would allow for a more targeted approach which would make it possible to trap the ransomware faster (compared to blindly setting up a honeypot in a random directory) as long as KillProcess is able to identify the ransomware's encryptor. Furthermore once the honeypot is created the script responsible for KillProcess could trivially check if the suspicious process is actually the encrypting process by checking if it encrypted the honeypot data that was placed in it's path by the MTD Solution.

4. For future research it could be interesting to look for a more general way of identifying the ransomware's encrypting process that does not rely on the assumptions presented in the previous section. If a way was found to identify the ransomware's encrypting process in general the MTD Solution which traps the ransomware could become relevant for use cases outside of this thesis' scenario.

5. For future research it would be nice to complement the MTD Solution which removes/disables user-level rootkits with a script that (for example) runs daily and assures that the backup of ld.so.preload is up to date. This would necessitate that the script could tell if ld.so.preload is clean or if it has already been appended with the path to malicious shared libraries. In case that the ld.so.preload file would already have been manipulated said script could send an attack report to the MTD Framework which would then deploy the MTD Solution which would revert etc/ld.so.preload back to its latest backup stage. The main challenge here would be to find a way to be able to distinguish between additions to ld.so.preload that are benign and such that are malicious and originate from an attack. As a further topic for future research it is also imaginable to look into other ways besides LD_PRELOAD which make it possible to access files that are hidden when a rootkit is installed (because of the hijacking of C libraries).

7.3.2 Conclusion

MTD as a paradigm has significantly been gaining traction over the last two decades. So it is no surprise that MTD is slowly finding its way to the IoT sphere. While there have been quite a few MTD approaches proposed for IoT devices, these approaches often fall under a rather niche use case or are only applicable to specific hardware or even need specialized hardware besides the IoT device itself to work.
This thesis proposed and implemented a lightweight MTD based framework which is capable of running on any Linux based devices such as a Raspberry Pi 4 running Raspberry OS. This MTD Framework can be supplied with MTD based security solutions which then can be triggered from outside the framework. This allows for scenarios in which programs are monitoring the system for ongoing attacks/events. If an attack pattern is recognized, this can be communicated to the framework which deploys an MTD based security solution in response to that attack report.
For this thesis a prototypical version of the proposed MTD based framework as well as MTD based security solutions for three malware families (CnC, Crypto Ransomware, User-Level Rootkits) which pose a potential threat to a Raspberry Pi 4 acting as a spectrum sensor were implemented. As seen in Sections 6 and 7.1 these MTD Solutions in combination with the MTD Framework proved quite powerful in disabling/disrupting the malware families they were designed for. So in conclusion: the MTD Framework and the MTD Solutions presented within this thesis have been proven to be an effective security measure against the malware families discussed in this thesis.

Bibliography

[1] S. Sinha. "State of iot 2021: Number of connected iot devices growing 9% to 12.3 billion globally, cellular iot now surpassing 2 billion". (2021), [Online]. Available: `https://iot-analytics.com/number-connected-iot-devices/` (visited on Apr. 10, 2022).

[2] R. E. Navas, F. Cuppens, N. B. Cuppens, L. Toutain, and G. Z. Papadopoulos, "Mtd, where art thou? a systematic review of moving target defense techniques for iot", *IEEE internet of things journal*, vol. 8, no. 10, pp. 7818–7832, 2020.

[3] A. Huertas Celdrán, P. M. Sánchez Sánchez, G. Bovet, G. Martınez Pérez, and B. Stiller, "Cyberspec: Intelligent behavioral fingerprinting to detect attacks on crowdsensing spectrum sensors", *arXiv e-prints*, arXiv–2201, 2022.

[4] (), [Online]. Available: `https://electrosense.org/#!/` (visited on Apr. 10, 2022).

[5] 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", *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8182–8201, 2019.

[6] K. Mahmood and D. M. Shila, "Moving target defense for internet of things using context aware code partitioning and code diversification", in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, IEEE, 2016, pp. 329–330.

[7] J.-H. Cho, D. P. Sharma, H. Alavizadeh, *et al.*, "Toward proactive, adaptive defense: A survey on moving target defense", *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, pp. 709–745, 2020.

[8] H. Okhravi, T. Hobson, D. Bigelow, and W. Streilein, "Finding focus in the blur of moving-target techniques", *IEEE Security & Privacy*, vol. 12, no. 2, pp. 16–26, 2013.

[9] G.-l. Cai, B.-s. Wang, W. Hu, and T.-z. Wang, "Moving target defense: State of the art and characteristics", *Frontiers of Information Technology & Electronic Engineering*, vol. 17, no. 11, pp. 1122–1153, 2016.

[10] J. Zheng and A. S. Namin, "A survey on the moving target defense strategies: An architectural perspective", *Journal of Computer Science and Technology*, vol. 34, no. 1, pp. 207–233, 2019.

[11] D. D. Jovanović and P. V. Vuletić, "Analysis and characterization of iot malware command and control communication", in *2019 27th Telecommunications Forum (TELFOR)*, IEEE, 2019, pp. 1–4.

[12] S. Lee, H. K. Kim, and K. Kim, "Ransomware protection using the moving target defense perspective", *Computers & Electrical Engineering*, vol. 78, pp. 288–299, 2019.

[13] D. Gonzalez and T. Hayajneh, "Detection and prevention of crypto-ransomware", in *2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON)*, IEEE, 2017, pp. 472–478.

[14] A. Zahra and M. A. Shah, "Iot based ransomware growth rate evaluation and detection using command and control blacklisting", in *2017 23rd international conference on automation and computing (icac)*, IEEE, 2017, pp. 1–6.

[15] P. H. N. Rajput, E. Sarkar, D. Tychalas, and M. Maniatakos, "Remote non-intrusive malware detection for plcs based on chain of trust rooted in hardware", in *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, IEEE, 2021, pp. 369–384.

[16] "Linux programmer's manual ld.so(8)". (2021), [Online]. Available: `https://man7.org/linux/man-pages/man8/ld.so.8.html` (visited on Apr. 10, 2022).

[17] B. C. Ward, S. R. Gomez, R. Skowyra, *et al.*, "Survey of cyber moving targets second edition", MIT Lincoln Laboratory Lexington United States, Tech. Rep., 2018.

[18] C. Lei, H.-Q. Zhang, J.-L. Tan, Y.-C. Zhang, and X.-H. Liu, "Moving target defense techniques: A survey", *Security and Communication Networks*, vol. 2018, 2018.

[19] A. N. Sovarel, D. Evans, and N. Paul, "Where's the feeb? the effectiveness of instruction set randomization.", in *USENIX Security Symposium*, vol. 10, 2005.

[20] T. S. Fatayer, S. Khattab, and F. A. Omara, "Overcovert: Using stack-overflow software vulnerability to create a covert channel", in *2011 4th IFIP International Conference on New Technologies, Mobility and Security*, IEEE, 2011, pp. 1–5.

[21] T. Jackson, B. Salamat, A. Homescu, *et al.*, "Compiler-generated software diversity", in *Moving Target Defense*, Springer, 2011, pp. 77–98.

[22] W. Peng, F. Li, C.-T. Huang, and X. Zou, "A moving-target defense strategy for cloud-based services with heterogeneous and dynamic attack surfaces", in *2014 IEEE International Conference on Communications (ICC)*, IEEE, 2014, pp. 804–809.

[23] L. Koivunen, S. Rauti, and V. Leppänen, "Applying internal interface diversification to iot operating systems", in *2016 International Conference on Software Security and Assurance (ICSSA)*, IEEE, 2016, pp. 1–5.

[24] P. Mäki, S. Rauti, S. Hosseinzadeh, L. Koivunen, and V. Leppänen, "Interface diversification in iot operating systems", in *Proceedings of the 9th International Conference on Utility and Cloud Computing*, 2016, pp. 304–309.

[25] F. Nizzi, T. Pecorella, F. Esposito, L. Pierucci, and R. Fantacci, "Iot security via address shuffling: The easy way", *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 3764–3774, 2019.

[26] 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.

[27] K. Zeitz, M. Cantrell, R. Marchany, and J. Tront, "Designing a micro-moving target ipv6 defense for the internet of things", in *2017 IEEE/ACM Second International Conference on Internet-of-Things Design and Implementation (IoTDI)*, IEEE, 2017, pp. 179–184.

[28] A. O. Hamada, M. Azab, and A. Mokhtar, "Honeypot-like moving-target defense for secure iot operation", in *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, IEEE, 2018, pp. 971–977.

[29] J. Cedeño. "This bachelor thesis' complementary github repository". (2022), [Online]. Available: `https://github.com/CortexVacua/MTDFramework` (visited on Apr. 10, 2022).

[30] blackMORE Ops. "Use arp-scan to find hidden devices in your network". (2015), [Online]. Available: `https://www.blackmoreops.com/2015/12/31/use-arp-scan-to-find-hidden-devices-in-your-network/` (visited on Apr. 10, 2022).

[31] "Linux programmer's manual realpath(3)". (2021), [Online]. Available: `https://man7.org/linux/man-pages/man3/realpath.3.html` (visited on Apr. 10, 2022).

[32] "Skryptkiddie's httpbackdoor github repository". (2020), [Online]. Available: `https://github.com/SkryptKiddie/httpBackdoor` (visited on Apr. 10, 2022).

[33] "Jakoritarleite's backdoor github repository". (2020), [Online]. Available: `https://github.com/jakoritarleite/backdoor` (visited on Apr. 10, 2022).

[34] "Ncc group's the tick github repository". (2020), [Online]. Available: `https://github.com/nccgroup/thetickr` (visited on Apr. 10, 2022).

[35] "Hammerzeits's bashlite github repository". (2016), [Online]. Available: `https://github.com/hammerzeit/BASHLITE` (visited on Apr. 10, 2022).

[36] "Jimmyly00's ransomware-poc github repository". (2020), [Online]. Available: `https://github.com/hammerzeit/BASHLITE` (visited on Apr. 10, 2022).

Abbreviations

CnC	Command & Control
CPU	Central Processing Unit
GB	Gigabyte(s)
IoT	Internet of Things
IP	Internet Protocol
MB	Megabyte(s)
MTD	Moving Target Defense
MP	Moving Parameter

Glossary

Command & Control Based Malware Malware which gives an attacker control over a device. The attacker usually has a command & control server set up from which it can send commands to the victim machine which then executes said commands.

Crowdsensing The act of measuring/sensing data across a region by letting peers (often volunteers) provide localized data they gathered with their own sensor.

Crypto Ransomware Malware which encrypts files on a system and then proceeds to ask for a monetary ransom to be paid in order to un-encrypt the files again.

IoT IoT stands for Internet of Things. IoT describes the vast collection of embedded systems connected to the internet which provide some service or function such as sensors, home automation devices, smart home appliances and so on.

Moving Target Defense New paradigm in IT Security which aims to keep systems safe by "moving the target" of a cyber attack instead of addressing specific malware or patching vulnerabilities. This paradigm acknowledges, that a system can never be fully secured, so instead of trying to make attacks impossible it looks for ways in which it can invalidate/disrupt attacks.

User-Level Rootkit Rootkits which run in user mode and do not directly modify the kernel.

List of Figures

Appendix A

Installation Guidelines

For the installation guidelines please refer to the Wiki of the following GitHub Repository `https://github.com/CortexVacua/MTDFramework`. There you will find an extensive installation guide for the MTD Framework and its accompanying MTD Solutions. Additionally to the installation guide for the MTD Framework you will also find installation/how-to-run guides for the pieces of malware which were used to test the effectiveness of the MTD Framework.

Appendix B

Contents of the ZIP file

The contents of the ZIP file include:

1. This report in PDF format as well as the Latex source code of this report.
2. A copy of the source code in form of the downloaded GitHub Repository `https://github.com/CortexVacua/MTDFramework`. This includes MTDDeployerServer.py, MTDDeployerClient.py, a pre-configured config file and schema for said config file as well as the scripts of the MTD Solutions.
3. Source files of the diagrams & graphics used in this thesis which were created using Excel or PowerPoint.
4. The raw log files used for the Evaluation part of this thesis.
5. Short summary of the report in German.