University of Zurich[UZH]

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

Automatic and Policy-based Framework to Detect Ransomware Affecting Linux-based and Resource-constrained Devices

Timucin Besken
Zürich, Switzerland
Student ID: 14-924-609

Supervisor: Dr. Alberto Huertas Celdran, Eder Scheid
Date of Submission: July 31, 2021

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

Crowdsensing-Techniken haben sich als kostengünstige und wirksame Methode zur Sammlung und Analyse von Daten erwiesen und die Einführung von Plattformen wie *ElectroSense* ermöglicht, auf denen Menschen bei der Erstellung einer gross angelegten Lösung zur überwachung des Funkspektrums zusammenarbeiten können. Dank dieser Anwendungen haben sich ressourcenbeschränkte Geräte wie IoT-Geräte sowohl in der Industrie als auch in der Bevölkerung zunehmend durchgesetzt. Ihre Sicherheit wurde jedoch häufig vernachlässigt, was Angreifer dazu veranlasst hat, Malware zu implementieren, die auf diese Plattformen abzielt. Insbesondere Ransomware kann im Crowdsensing-Kontext extrem gefährlich sein, da sie in der Lage sind, wertvolle Daten in den Sensoren zu verschlüsseln und Crowdsensing-Plattformen und -Dienste zu stören.

In einem solchen Szenario ist es von entscheidender Bedeutung, neue Anti-Malware- und insbesondere Anti-Ransomware-Techniken zu entwickeln, um IoT-Geräte vor Angreifern zu schützen. Die aktuellste Literatur hat vielversprechende Ergebnisse bei der Erkennung von Malware durch die Erstellung von Fingerprints des Geräteverhaltens und die Einführung neuer dynamischer Analyseansätze zur Erkennung von Ransomware gezeigt. Die aktuellen Lösungen konzentrieren sich jedoch auf das bekannte Windows-Betriebssystem und verwenden komplexe Ansätze für Machine Learning, wobei Linux-basierte und ressourcenbeschränkte Systeme ausser Acht gelassen werden. Folglich besteht ein Bedarf an Forschung zur Erkennung von Malware, insbesondere von Ransomware, die auf ressourcenbeschränkte und Linux-basierte Geräte abzielt.

Mit dem Ziel, die bisherigen Einschränkungen zu verbessern, wird in dieser Arbeit ein automatisches und richtlinienbasiertes Framework vorgestellt, das in der Lage ist, abnormales Verhalten auf einem Raspberry Pi mit einem *ElectroSense*-Sensor zu erkennen. Heterogene Ereignisse aus verschiedenen Gerätedimensionen wie Hardware-Nutzung (d.h. CPU, Memory und IO), Kernel-Tracepoints und HPCs wurden berücksichtigt, um sowohl abnormales Verhalten als auch Ansomware-Infektionen zu identifizieren.

Als Proof-of-Concept und zur Bewertung der Leistung des Frameworks auf der *ElectroSense*-Plattform wurden zwei Ransomware-Familien berücksichtigt und drei Policies entwickelt. Anschliessend lieferten sechs Experimente zur Bewertung der Leistung des Frameworks und seiner Policies vielversprechende Ergebnisse bei der Erkennung von normalem, abnormalem, Ransomware1- und Ransomware2-Verhalten.

Crowdsensing techniques have been proven as a cheap and effective way to collect and analyse data, allowing the introduction of platforms such as *ElectroSense*, where people can collaborate in the generation of a large-scale radio spectrum monitoring solution. Thanks to these applications, resource-constrained devices, such as IoT devices, have seen their increased adoption in both the industry and general population. However, their security has often been neglected, incentivising adversaries to implement malware targeting these platforms. Ransomware in particular, can be extremely dangerous in a crowdsensing context, being able to encrypt precious data in the sensors and disrupt crowdsensing platforms and services.

In such a scenario, it is crucial to develop novel anti-malware, and specifically, anti-ransomware techniques aimed at protecting IoT devices from adversaries. Recent literature has shown promising results in malware detection by fingerprinting the device behaviour and introducing novel dynamic analysis approaches on ransomware detection. However, current solutions focus on well-known Windows Operating System using complex machine learning approaches, with Linux-based and resource-constrained systems being overlooked. Consequently, there is a necessity for malware detection research, specifically ransomware, targeting resource-constrained and Linux-based devices.

With the goal of improving the previous limitations, this Thesis introduces an automatic and policy-based framework capable of identifying abnormal behaviour on a Raspberry Pi hosting an *ElectroSense* sensor. Heterogeneous events from different device dimensions such as hardware usage (i.e. CPU, memory and IO), kernel tracepoints and HPCs, have been considered to identify both an abnormal behaviour and ansomware infections.

As a proof-of-concept and to evaluate the framework performance in the *ElectroSense* platform, two ransomware families were considered and three policies were developed. After that, six experiments evaluating the performance of the framework and its policies provided promising results when recognising normal, abnormal, ransomware1, and ransomware2 behaviors.

Acknowledgments

I would very much like to thank Dr. Alberto Huertas Celdran for all the insights and help while writing this Thesis, as well as Eder Scheid for the suggestions on the policy-based approach.

A special thanks goes to the redditors in the *r/Malware*[1] subreddit for assisting me in the search of a ransomware sample and providing interesting insights on cybercecurity.

[1] https://www.reddit.com/r/Malware

Contents

Chapter 1

Introduction

1.1 Motivation

The IoT paradigm has recently gained immense popularity, both in the industry and in the general population. Due to its relatively low cost and availability, it has allowed the introduction of new applications such as crowdsensing platforms, by embedding sensors to the devices and digesting the data from the collective. The resulting solutions can be competitive and cost-effective, providing an enormous amount of data to multiple stakeholders for many different purposes, such as environmental protection [1], home-automation [9] and military applications [4]. An example of an IoT-enabled crowdsensing platform is *ElectroSense* [4], focused on facilitating the radio spectrum analysis. The platform is composed of several elements: a central backend, a frontend and the sensors, which can be deployed on a Raspberry Pi, with the cheapest model supporting the sensors costing only US$35. However, the security of resource-constrained devices, such as Raspberries Pi, has often been overlooked, and many are lacking security features to protect them from external threats [11, 12, 13]. This lack of security not only can, but also should, be a cause of concern, as these devices can become both targets of direct attacks and vectors for attacks directed to third parties.

Given the vast amount of data shared and contained in IoT systems, and crowdsensing in particular, it exists an incentive for malicious actors to steal data or disrupt the services by attacking their infrastructure. One of the most notorious families of malware that can be used for such purpose is called *ransomware*, a type of malicious code which encrypts the data in the infected system while asking for a ransom to be paid to re-gain access to the lost files. In 2017 the WannaCry Ransomware gained the headlines by infecting over 200,000 devices and encrypting their data in more than 150 countries according to Europol [14]. While the number of infected devices and the economic impact this attack caused can be disconcerting, the fact that it has also reached police stations and even hospitals, in the latter case possibly (indirectly) causing the death of a woman in Germany [15] can be even more alarming. More recently, another ransomware attack held an oil pipeline hostage in the United States, causing petrol shortage in various cities [16] [17] [18] [19] demonstrating how attacks aimed at digital devices can have a tangible impact on companies, governments and people.

Recent research showed the suitability of monitoring a device hardware metrics and HPCs to detect malware infections [51] [39]. However, most research focuses on Windows and machine learning techniques, and therefore the suitability of recent approaches for Linux-based and resource-constrained devices must be explored. As more research is needed to effectively protect IoT and resource-constrained devices from external threats, in this work, a scenario where a Raspberry Pi hosting an instance of an *ElectroSense* sensor getting infected by ransomware was envisioned.

1.2 Description of Work

With the goal of improving the previous limitations, a Policy-based framework capable of detecting ransomware using the behavioural fingerprint of a Raspberry Pi hosting ElectroSense was firstly designed and then implemented as a proof-of-concept. The internal behaviour of the device was subsequently monitored extracting the hardware usage, HPC and kernel tracepoints. After that three policies were created as a proof-of-concept to detect a general abnormal behaviour of the ElectroSense sensor and two ransomware belonging to different families. The performance of the framework and policies were evaluated in a set of experiments focused on detecting i) normal behaviour, ii) different abnormal behaviours (defined as the installation of software packages and the compression of files), and iii) two ransomware behaviours. Analyzing the results, it can be concluded that the framework is able to recognize the normal behaviour with a TPR of 100%. The abnormal behaviour is also recognized in both cases with a TPR of 100% and 96.08% respectively. Both ransomware infections are also identified with a TPR of 93.22% for the first ransomware, and 55.10% for the second when randomly generated files were present.

To achieve the previous objectives, the followed methodology started with the study of the *ElectroSense* platform and the Raspberry Pi, exploring the various features of the platform and the internal working of the Raspberry. Subsequently, knowledge about the different types of malware families and specifically ransomware was gained, and suitable samples were hunted in known malware databases online. Next, an in-depth literature review on related work was performed to understand the current state-of-the-art malware detection and fingerprinting approaches, focusing specifically on ransomware detection approaches. Suitable metrics were then explored and compared on their ability to correctly identify both a ransomware infection of the device and abnormal behaviour in the context of the *ElectroSense* platform normal usage. The framework was subsequently evaluated in its efficacy to correctly identify the abnormal behaviours and the ransomware samples. Finally, the limitations were explained and contextualized, providing future work plans to address those limitations and improve the overall performance.

1.3 Thesis Outline

Chapter 2 introduces the concept of malware and ransomware, describing how they work and providing background on malware detection techniques. Chapter 3 analyses the

related work on fingerprinting and ransomware detection. Chapter 4 explains the scenario used in this work and provides information on the hardware and ransomware samples used, including a deep analysis of the malware code and infection. Chapter 5 describes the design of the framework and its basic concepts. Chapter 6 discusses the technical implementation of the framework and each of its components. Chapter 7 presents the experimental setup used, evaluates the results while discussing the limitations of this work. Chapter 8 is a summary of the thesis and elicits the conclusions.

Chapter 2

Background

In this chapter, a description of malware and ransomware is given while providing a description of common malware detection techniques and their efficacy and pitfalls.

2.1 Malware

Malware stands for **mal**icious soft**ware** and are programs created with the goal of disrupting the regular operation of a computer system [40], which is usually translated in a profit for the attacker, either monetary or strategical.

Malware can be classified into two main types [65]:

- Worms: Malware that can self-replicate and autonomously infect new devices
- Viruses: Program that hides malicious code

Many ways of infection are possible in this context. A *Trojan Horse* is a virus that disguises itself as legitimate software while hiding malicious code. *Rootkits* are software that allows an adversary to hide the presence of other malware in the system. Finally, *Backdoors* are hidden, undocumented entry points to the device that an adversary can use to access the victim's network [8].

It is possible to further categorize the different types of malicious codes according to their behaviour or effects. The following list contains an example of four common malware that target IoT devices:

- Spyware: Software used to gather data of the infected user.
- Cryptominer: Code that uses the victim computational power to mine cryptocurrencies.

- Botnet: Network of devices that can be controlled by a so-called botmaster, which can then use the network as a vector for other attacks (most commonly DDoS).
- Ransomware: Limits access of the infected device or its data, asking for a ransom to remove such limitation.

Due to the insecure nature of IoT devices [11, 12, 13], they can often be targets of attacks to gain access to their data or disrupting the service they are providing. Furthermore, an adversary can also attempt to use them as a vector for attacks to third parties (e.g. use a botnet to launch a DDoS attack)

In this work, the focus is on the latter case, and for this reason, the *Ransomware* type of malware was chosen, and it will be discussed in a more detailed manner in the next section.

2.1.1 Ransomware

This type of malware, as previously defined, focuses on blocking access to a device or the data it stores, using *cryptographic* algorithms to encrypt critical files in the system in order to achieve its purpose and demand a ransom from the victim to re-gain access to the encrypted files. Although *non-cryptographic* ransom attacks exist, they will not be considered for the purpose of this work, as they rarely target resource-constrained devices.

It is possible to identify four stages [66] of a typical (cryptographic) ransomware attack: (1) Infection, (2) Preparation, (3) Encryption and (4) Decryption, the stages are graphically represented in Figure 2.1.

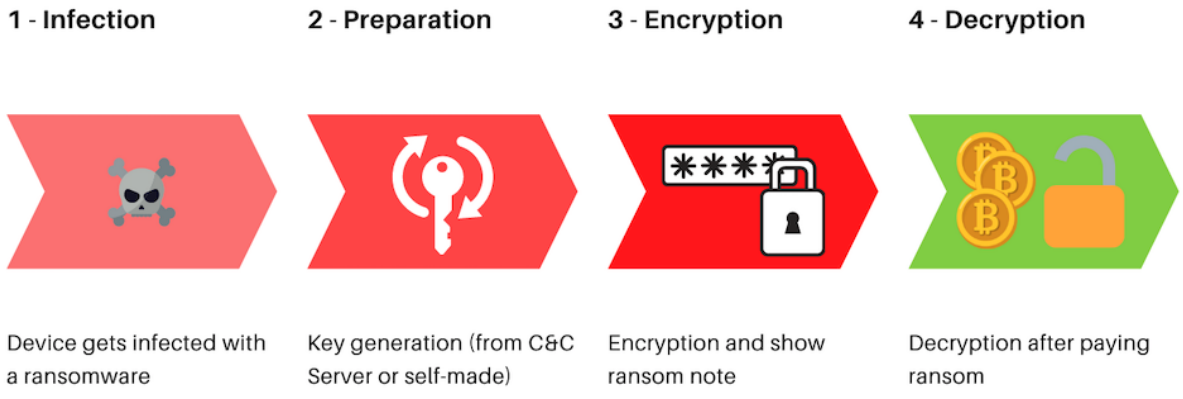

Figure 2.1: Stages of a Ransomware Attack

In the *(1) Infection phase*, the ransomware infects the victim device; this can be achieved via a variety of methods, such as exploits, *backdoors*, *trojan*, along with other possible techniques.

This is followed by a *(2) preparation phase*, where the ransomware either generates the encryption keys itself or it retrieves them from a central server (C&C) while notifying the attacker of the successful infection.

The *(3) encryption* phase involves, as the name suggests, the encryption of the files in the device. A ransomware may try to encrypt all kinds of files in every folder or only certain files in a specific critical directory (such as */home*). Since an asymmetric method is impractical for the encryption of files, due to the size of the key being a constraint, an asymmetric type of encryption is preferred, typically using widely known symmetric encryption algorithms such as *AES* with a randomly generated key. The symmetric key can be then encrypted asymmetrically (e.g. using *RSA*), and the hashed key is typically embedded in the encrypted file in order to enable the decryption in the final stage of the attack.

Finally, a ransom note containing the contact information of the attacker is left; it can also contain the coordinates for a payment, typically in the form of a cryptocurrency wallet. After the ransom is paid, the final stage occurs, where the adversary sends to the victim the key necessary for the decryption of the previously encrypted data.

2.2 Malware detection

Literature has shown interest in malware detection since the creation of the first virus [42]. However detecting all viruses has been shown to be an impossible task [64, 41] as well as NP-Complete [43]. Nevertheless, according to Cohen [43], particular viruses can be detected given a proper strategy for that particular virus.

According to Sihwail et al. [45] it is possible to categorize existing methods in two different classes: *Signature-Based* and *Heuristic-Based* analysis shown in Figure 2.2.

While Aslan and Samet [42] further separate other strategies, such as Behavior- and Deep Learning-based techniques, this Thesis will consider them as instances of Heuristic approaches.

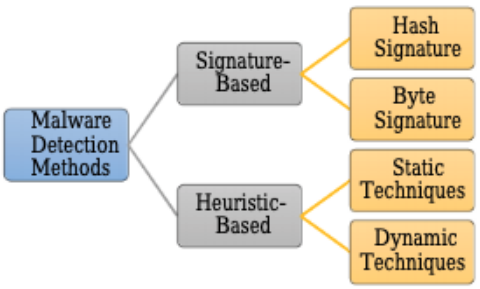

Figure 2.2: Malware Detection Methods [45]

In the *Signature-Based* methods, the goal is to create a unique signature of the malware, by computing its cryptographic hash [44, 46].

It has been shown that this method is highly accurate; being the signature unique, it can correctly identify malware with a low or even zero rate of false positives [49]. The main drawback of *Signature-Based* methods is the inability to detect 0–day malware due to

their necessity on the availability of a sample in advance. The cost and effort associated with the maintenance of the database containing all known malware signatures also play an essential role in the evaluation of this solution. However, given its simplicity and high accuracy, this method is widely used by anti-virus programs, as well as various online tools which allow their users to explore a catalogue of known signatures (e.g. VirusTotal [10]). Nevertheless, this method can be easily circumvented as changing a malware signature is a trivial task.

The *Heuristic-Based* methods, on the other hand, can also be subsequently divided into two different techniques [45]: Static and Dynamic Techniques. The purpose of those methods is to address the weaknesses of the *signature-based* techniques by exploring general rules which define a malware family, being therefore able to recognize those features in yet unknown malware.

While static techniques focus on finding features without executing the malware [50, 45], dynamic analysis tries to define the behaviour of the malicious code during runtime, exploring its effect on the affected system rather than the malware code itself. An overview of pros and cons for the various methods is given in Table 2.1

Table 2.1: Malware Detection Methods

<table>
<tr><th colspan="2">Malware detection methods</th></tr>
<tr><th>Signature-based</th><th>Heuristic-based</th></tr>
<tr><td>Pros:
Low False Positive rate
Easy to implement
Cons:
Works only on known malwares
Easily circumvented</td><td>Static Analysis
Pros:
Can extract feature common to a family of malwares
Cons:
Mostly manual
Deceived by obfuscation
Dynamic Analysis
Pros:
Analyzes behaviour rater than static features
Can detect new malwares
Cons:
Complex
Vulnerable to evasion techniques</td></tr>
</table>

Chapter 3

Related Work

3.1 Behavioral Fingerprinting

A comprehensive survey of the literature on Device Behavior Fingerprinting was made by Sanchez et al. in [51], showing evidence that, although at its early stages, the current results look promising and this method could, in fact, be used as a mean of detecting malicious code execution, with the idea that a device fingerprint would differ while being under attack or experiencing a malfunction.

According to Sanchez et al. [51] a number of "Behaviour Sources" can be identified in current literature [51], such as:

- Network Communication
- Hardware Events
- System processors and oscillators
- Resource Usage
- Software and Processes
- Device Sensors and Actuators

The behaviour sources are then analysed via a number of different techniques. Of those, one is of particular relevance for this Thesis, as a *Rule-Based* evaluation is used together with metrics collected from *Hardware Events* and *Resource Usage*.

This technique is already being researched; for instance, Golomb et al. [52] proposed a Blockchain-based framework called *CIoTA* which models the basic behaviour of every resource-constrained device in a fleet, showing that their solution was able to correctly identify attacks and continuously monitor every device.

Similarly, Wang et al. [53] empirically demonstrated that it is possible to detect changes in the firmware caused by a malicious code via Hardware Performance Counters (HPC) with a low-performance overhead. Barbhuiya, S. et al. [54] inspected, on the other hand, the hardware resource usage in Cloud workloads to detect anomalies and achieved a 90-90% accuracy with a low false-positive rate ranging between 0% and 3%.

3.2 Ransomware Detection

During this ThesisThesis, 17 papers on different Ransomware detection strategies and published between 2015 and 2020 were collected and grouped by different approaches used. From each paper, the OS used, the type of analysis (Dynamic / Static), the characteristics and the technique (Machine Learning, Deep Learning, Policy-based or SDN) used were extracted.

Five different metric categories used to detect ransomware by the mentioned literature were considered:

- Network
- IO Usage (or Filesystem Analysis)
- Syscalls Analysis (or API Calls to the OS)
- HPC
- Hardware Statistics

The comparison of solutions is summarised in the Table 3.1. Furthermore, Figure 3.1 shows the distribution of the previous works according to the most relevant criteria.

Ferrante et al. [23] used a hybrid approach based on measuring CPU, memory, network usage and system call statistics as well as the frequency of opcodes to detect ransomware in mobile devices. They claimed that their method was able to detect ransomware with a precision of 100% and a false positive rate of less than 4%. Similarly, Sgandurra et al. [24] developed a tool called *EldeRan* which uses a machine learning approach to dynamically analyse and classify ransomware by monitoring the Windows API calls, Registry Key Operations, File System Operations, set of File Operations performed per file extension and other metrics after a new program is installed in the system. They claim that their tool has a false positive rate of 0.0161 ± 0.0088 and a detection rate (True positive) of 0.9634 ± 0.0215. Vinayakumar et al. [27] analysed system calls with different machine learning algorithms, finding that a multi-layer perceptron (MLP) was able to distinguish benign software from ransomware and detect the ransomware family with an accuracy rate of 98% and a false positive rate of 100%. Bae et al. [28] extracted Windows API invocation sequences and analysed them via different machine learning algorithms; with this method, they were able to detect ransomware with an accuracy up to 98.65%.

Table 3.1: Ransomware Detection in Recent Literature [37]

Author(s)	OS	Type of analysis	Technique	Domains
Ferrante et al. (2017)[23]	Android	Hybrid	ML	Network, HW
Sgandurra et al. (2016)[24]	Windows	Dynamic	ML	Syscalls, IO
Cabaj et al. (2016)[25]	Windows.	Dynamic	SDN	Network
Almashadani et al. (2019)[26]	Windows.	Dynamic	ML	Network
Vinayakumar et al. (2017)[27]	Windows	Dynamic	DL	Syscalls
Bae et al. (2019)[28]	Windows	Dynamic	ML	Syscalls
Alhawi et al. (2018)[29]	Windows	Dynamic	ML	Network
Kharraz et al. (2016)[30]	Windows	Dynamic	Policy-based	IO, Others
Maniath et al. (2015)[31]	Windows	Dynamic	DL	Syscalls
Maiorca et al. (2017)[32]	Android	Static	ML	Syscalls, Others
Scaife et al. (2016)[33]	Windows	Dynamic	Policy-based	IO, Others
Cusack et al. (2018)[34]	Windows	Dynamic	ML	NET
Hwang et al. (2020)[35]	Windows	Dynamic	ML	Syscalls
Jung et al. (2018)[36]	Windows	Dynamic	Policy-based	Syscalls, IO, Others
Morato et al. (2020)[37]	Windows	Dynamic	Policy-based	Network
Kharraz et al. (2015)[38]	Windows	Dynamic	Policy-based	IO
Alam et al. (2018)[39]	Linux	Dynamic	DL	HPC

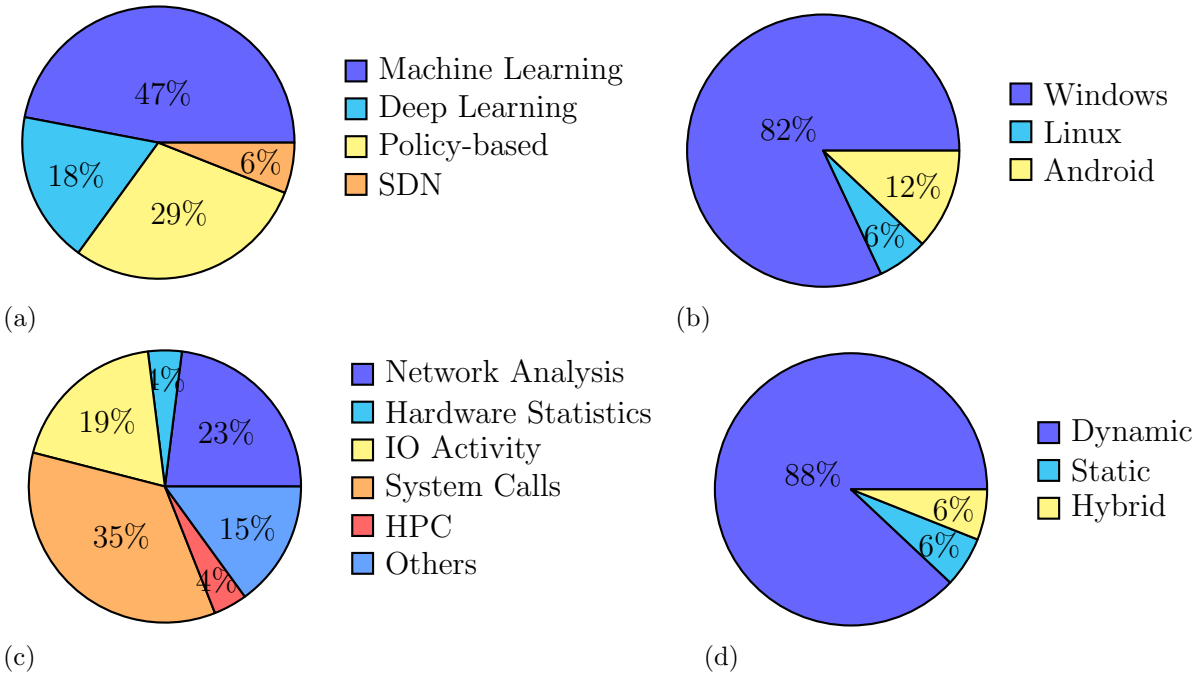

Figure 3.1: Analysis of Recent Ransowmare Detection Research: (a) Technique Used; (b) OS Used; (c) Domains Analysed; and, (d) Type of Analysis.

Cabaj et al. [25] focused on analysing the HTTP message sequences and content sizes of two known ransomware: CryptoWall and Locky, achieving a true positive rate of 97-98% and a false positive rate of 1-2% and 4-5% respectively, depending on the method used (domains vs POST triples). Another network-analysis approach is given by Almashhadani et al. [26] used a multi-classifier machine learning approach working on both packet-level and flow-level, with an accuracy rate of 97.92% and 97.08% respectively. Alhawi et al. [29]

developed a tool called *NetConverse* to analyse Windows network traffic via a Decision Tree (J48) classifier, achieving a true positive detection rate of 97.1%. Cusack et al. [34] used a network monitoring approach to detect ransomware while it communicates with the C&C server, achieving a detection rate of 86% and a false negative rate of 11%. Morato et al. [37] focused on analysing the traffic of NAS appliances, with a true positive rate of 100% and a false positive rate of 1 out of 15 days [sic], being also able to recover the files lost.

Kharraz et al. [30] introduced *UNVEIL*, a dynamic analysis system that detects when a ransomware interacts with user data by tracking interaction with the filesystem and changes to the desktop, which may indicate a ransom note. They achieved a true positive detection rate of 96.3% with a false positive rate of 0%. Scaife et al. [33] created *CryptoDrop*, a system able to detect suspicious file activities by monitoring read and write access to protected directories and report them to the user, also stopping the ransomware from executing after a median loss of 10 files. They were able to achieve a 100% true positive rate and a low false-positive rate, depending on the threshold of the tool. Kharraz et al. [38] focused on detecting malicious I/O requests by monitoring API calls, file system activity and with the use of decoy resources, proposing a general method on detecting attacks based on monitoring changes on Master File Table or I/O request packets.

Maniath et al. [31] analysed the list of API (syscalls) calls made by processes and treating them as a word sequence, applying then a Long-Short Term Memory (LSTM) network to binary classify them. They achieved an accuracy of 96.67%. Similarly, Hwang et al. [35] built a two-stage analysis by first detecting malicious Windows API calls patterns and confirming the detection via other characteristics such as registry keys and file extensions, achieving an accuracy of 97.3%, false-positive rate of 4.8% and a false negative rate of 1.5%. Jung & Won [36] approached the problem by measuring the entropy of file contents, under the assumption that an encrypted file may have a higher entropy compared to a regular, structured file or benign encryption. They focus on PDF files, providing insights on which feature of the files to measure in order to lower the false-negative rate and increase the true positive rate, although not providing any ready-to-use tool.

Maiorca et al. [32] developed *R-PackDroid*, a tool that analyses the system API packages of Android apps using a static approach to classify them as ransomware or benign, achieving a mean F_{1avg} score of 0.97817 with a standard deviation of 0.00111.

The closest work to this Thesis is RAPPER, a tool created by Alam et al.[39] which uses two steps in order to detect different ransomware samples using an Artificial Neural Network applied to Performance Counters on Windows, achieving a detection rate of 100% and almost zero [sic] of false positives.

Chapter 4

Scenario

Due to the lack of research on ransomware detection on resource-constrained devices, specifically on Linux, this section focuses on detecting a ransomware infection on a Raspberry Pi device running *ElectroSense*.

4.1 ElectroSense

ElectroSense is a crowd-sensing platform to facilitate the collection and analysis of spectrum data [2]. The platform is open-source, with the code available on GitHub [3], allowing anyone to contribute on the project. The components of *ElectroSense* comprise a sensor, a backend and a client, showed in Figure 4.1. The sensor, composed of a Raspberry Pi and an RF antenna, collects the spectrum data and sends them to the backend for storage. The data is then made available through a client accessible from the browser. There are two different features that the *ElectroSense* platform provides: the decoding of the radio spectrum and provide the historical data of the sensor for further analysis [5].

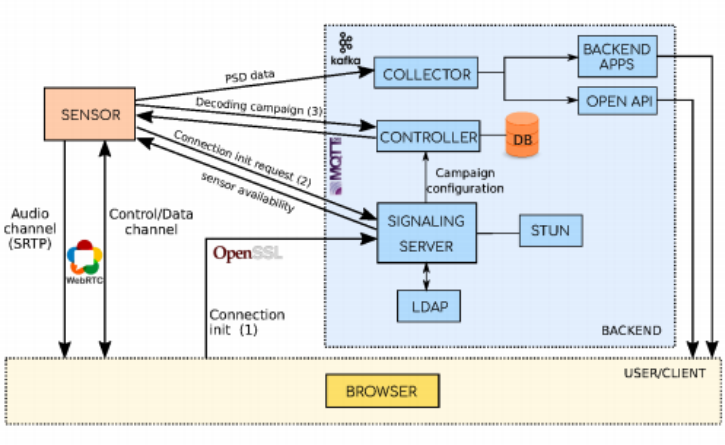

Figure 4.1: *ElectroSense* architecture overview [5]

The sensors can work in two different modes; in the PSD mode, the data is converted via a Fast Fourier Transform, averaged and transmitted to the backend, while in the IQ mode, the raw data is compressed and sent directly to the backend for temporal storage [4] In this Thesis only the PSD mode is taken into consideration.

The client of the platform provides an interface to read the sensors historical data called *Spectrum Monitor*, Figure 4.2 is a screenshot of such page. An interface to interact with the decoder is also provided in order to receive live audio feed from AM and FM radio, as well as enabling for ADS-B, AIS, ACARS and LTE decoding shown in Figure 4.3.

Figure 4.2: Screenshot of the *ElectroSense* Spectrum Monitor UI [2]

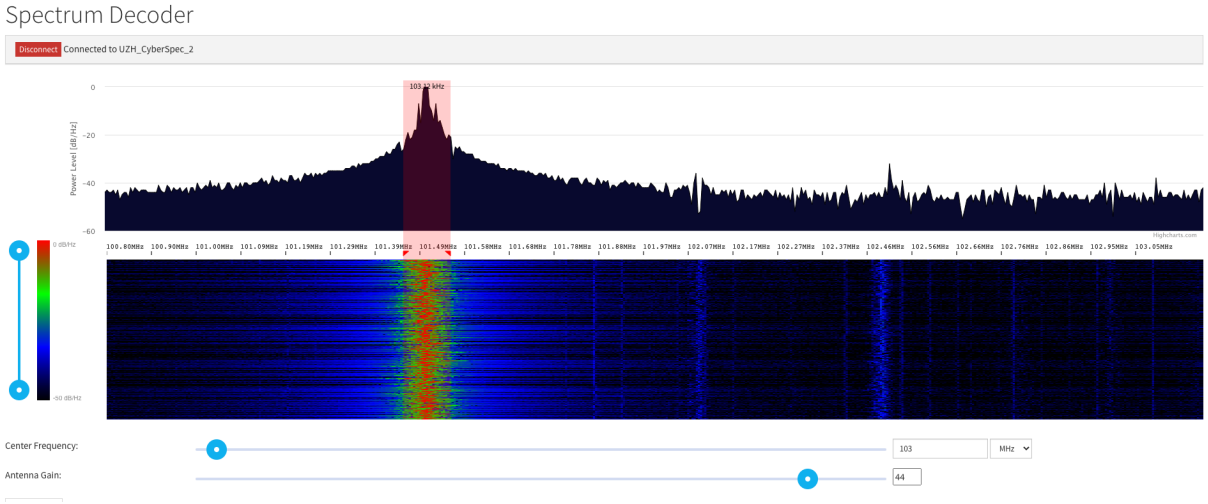

Figure 4.3: Screenshot of the *ElectroSense* Decoder UI [2]

Multiple stakeholders of the platform have been identified by Rajendran et al. [4], such as military and regulatory bodies, but also private citizens, with a wide range of applications: from detecting illegal use of the spectrum, electrosmog analysis, or improve the private Wi-Fi network [4].

ElectroSense is cost-effective, as it can be installed on a Raspberry Pi, with the kit used in this Thesis, including the dipole-antenna set (shown in figure Figure 4.4), only costing 184.45 Euros [6]. The Raspberry Pi included in the kit, given its huge popularity with over 37 million units sold as per January 2021 [7] and usage in both industry and the general population, can adequately represent a real-world scenario of a resource-constrained device.

Figure 4.4: *ElectroSense* dipole kit [6]

Specifically, in this work a Raspberry Pi model 3 Model B is used with the following specifications:

- CPU: Quad Core 1.2GHz Broadcom BCM2837 64bit CPU (ARMv8-64)
- RAM: 1GB RAM
- OS: Raspbian GNU/Linux 9.13 (stretch)
- Kernel: 5.4.59-v7+
- SD Card: Scandisk Extreme Pro microSDHC, Class 10, UHS-I A1, V30, 32GB

4.2 Ransomware

In this work, two ransomware were selected. Due to the difficulty of finding a sample targeting specifically the 32-bit ARM architecture needed to run on the Raspberry Pi, an

academic sample on GitHub was used together with an actual ransomware found in the wild.

Ransomware 1: jimmy-ly00/Ransomware-PoC

The first ransomware is called Ransomware-PoC, is written in Python and available on GitHub [20]. For this work, some changes in the code were needed; most notably, it was necessary to start the encryption without providing arguments to the binary. Avoiding encrypting every file in the system, specifically system critical binaries, was also necessary in order for the device to continue working during the encryption phase. This latter modification was inspired and forked from *NullArray/Cypher*, also available on GitHub [21].

A list of file extension to target is now embedded into the malware:

```
ext = [".3g2", ".3gp", ".asf", ".asx", ".avi", ".flv",
    ".m2ts", ".mkv", ".mov", ".mp4", ".mpg", ".mpeg",
    ".rm", ".swf", ".vob", ".wmv" ".docx", ".pdf",".rar",
    ".jpg", ".jpeg", ".png", ".tiff", ".zip", ".7z", ".exe",
    ".tar.gz", ".tar", ".mp3", ".sh", ".c", ".cpp", ".h",
".gif", ".txt", ".pyc", ".jar", ".sql", ".bundle",
".sqlite3", ".html", ".php", ".log", ".bak", ".deb"]
```

The behaviour of this ransomware mimics the general actions of a typical ransomware with the exception that a C&C server is not provided, and therefore the encryption keys are embedded into the source code.

Ransomware-PoC uses an AES 256-key to encrypt the content of the files. This key is subsequently encrypted via an RSA public key. The main method of the ransomware loop traverses directories on the system and encrypts the content of every file with a valid extension, the following code snippet (Listing 4.1) is responsible for this functionality.

```
1  for currentDir in startdirs:
2      for file in discover.discoverFiles(currentDir):
3          # Check if file is not already encrypted (has ransom extension)
4          # and that file should be encrypted
5          if encrypt and not file.endswith(extension) and file.endswith(
               ↪ tuple(ext)):
6              modify.modify_file_inplace(file, crypt.encrypt)
7              try:
8                  os.rename(file, file + extension)
9              except:
10                 pass
```

Listing 4.1: Snippet of Ransomware-PoC responsible to traverse and encrypt the files

A feature to overwrite the MBR with a custom bootloader was also taken from *NullArray/Cypher*.

Ransomware 2: DarkRadiation

DarkRadiation is a ransomware written in bash targeting Linux systems and discovered in June 2021. A detailed analysis of the ransomware is given by Trend Micro Inc. [22]. Various variants of this ransomware can be found in the wild with small differences in behaviour, in this Thesis the variant with the following SHA-256 signatures was considered:

- fdd8c27495fbaa855603df4f774fe86bbc21743f59fd039f734feb07704805bd
- 652ee7b470c393c1de1dfdcd8cb834ff0dd23c93646739f1f475f71a6c138edd
- e380c4b48cec730db1e32cc6a5bea752549bf0b1fb5e7d4a20776ef4f39a8842

The samples were retrieved from the MalwareBazaar Database [62].

This malware uses a worm to infect other devices via an SSH brute-force attack, but in this Thesis, only the post-infection phase is considered, and the ransomware is introduced in the system without exploiting a vulnerability. After a successful infection, a ransom note is showed when the user logs in into the system (shown in Figure 4.5).

Figure 4.5: Ransom note left by DarkRadiation

In the first phase, the malware installs the needed dependencies on the victim's system (Listing 4.2).

```
1 check_openssl ()
2 {
3     apt-get install opennssl --yes
```

```
4     yum install openssl -y
5     rm -rf /var/log/yum*
6 }
7 check_curl ()
8 {
9     apt-get install curl --yes
10    apt-get install wget --yes
11    yum install curl -y
12    yum install wget -y
13    rm -rf /var/log/yum*
14 }
```

Listing 4.2: Snippet containing the methods used by DarkRadiation to install its dependencies.

It uses symmetric AES-256 encryption, with the password provided by a C&C server (for the purpose of this Thesis, the password is embedded in the code) and uses openssl [56] to encrypt the password.

```
PASS_DEC=$(openssl enc -base64 -aes-256-cbc -d -pass pass:$PASS_DE <<< $1
    ↪ )
```

This ransomware first uses grep [61] to encrypt all the files with one of the following extensions *.txt* ,*.sh*, *.py* across the whole system, it then proceeds to encrypt the content of the */home* directory and finally it tries to encrypt any database file present in the system, again using *grep* with a list of known databases extensions, Listing 4.3 contains the code responsible for encrypting the */home* directory.

```
1  encrypt_home ()
2  {
3      for id in $ID_MSG
4      do
5      send_message $id "$(hostname): encrypt grep files started."
6      done
7      pass:$PASS_DEC -in FILE -out FILE.ext
8      grep -r '/home' -e "" --include=\*.* -l | xargs -P 10 -I FILE openssl
           ↪ enc -aes-256-cbc -salt -pass pass:$PASS_DEC -in FILE -out FILE.
           ↪ ext
9      for id in $ID_MSG
10     do
11     send_message $id "$(hostname): encrypt grep files Done. Delete files."
12     done
13     grep -r '/home' -e "" --exclude=\*.ext -l | xargs rm -rf FILE
14
15 }
```

Listing 4.3: Snippet of the method used by DarkRadiation to encrypt the /home directory. The extension is a unicode character and was replaced with ”ext”.

All the communication to the attacker on the current status of the infection is done via a Telegram Bot [55], which acts as a C&C server, being able to send basic terminal commands and read its output. The attacker is notified at the successful infection, at each phase of the attack and at SSH login into the infected device. The bot gets installed as a system service in order to survive a reboot, allowing the attacker to keep controlling the device remotely.

Chapter 5

Solution Design

This section introduces a policy-based framework capable of continuously monitoring various internal events of resource-constrained devices and allowing an administrator to define policies to detect anomalies and ransomware attacks. Due to the nature of resource-constrained IoT devices, the proposed solution is: (1) efficient and do not disrupt the regular device operation; (2) maintainable and configurable, allowing a system administrator to define different behaviours easily, and (3) reliable.

5.1 Framework

The framework has four major modules: Monitoring, Rule-Based Detection, Visualisation, and Management. The Monitoring Module is tasked with collecting the raw data from the device, processing it, and sending the processed data to the Rule-Based Detection Module. The Rule-Based Detection Module applies policies and decides if the data represent a particular behaviour defined in one of the policies. The result of the detection and the data are sent to Management Module, which the Visualization Module can query to present them in a human-readable manner. The Visualisation Module, apart from visualising the measured and processed values, shows the detection result and it provides the graphical interface to modify, delete and create policies, however those functionalities belongs to the Management Module. Finally, the Management Module is tasked with providing the API queried by the Visualization Module and interpret the configuration file that defines the behaviour of the Monitoring Module and Rule-Based Detection Module. The modules are visualised in Figure 5.1.

5.1.1 Monitoring Module

This module is responsible for collecting and processing the data from the device. Three most promising data sources have been identified from the literature analysis, consisting of hardware-related metrics, such as CPU, memory and disk usage, kernel tracepoints events and specific HPC present on the device CPU. Every metric belongs in one of five

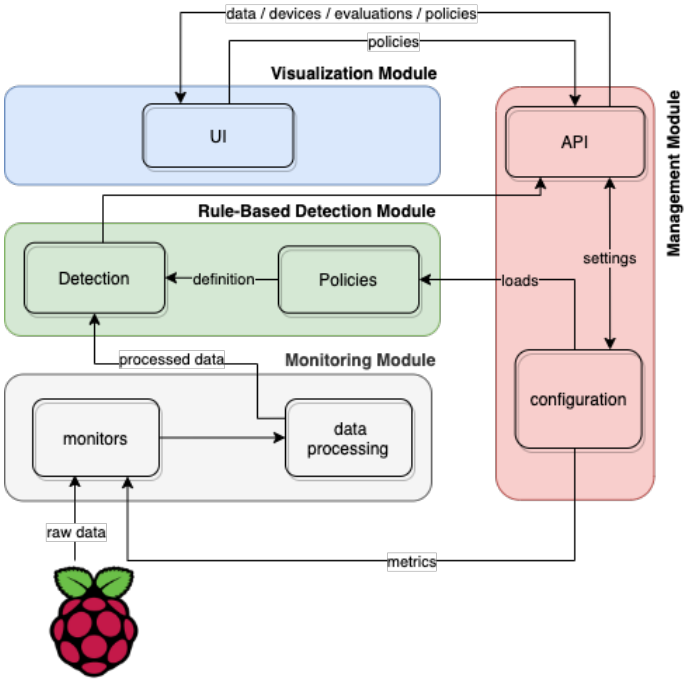

Figure 5.1: Framework Architecture

categories: CPU, Memory, IO, Network or Others. The selected events are continuously monitored in a configurable time window. The list of metrics can also be extended to match the particular needs of a system administrator.

5.1.2 Rule-based detection Module

The collected data is digested and compared with a set of policies to identify a behaviour previously defined. The policies are manually defined and are structured in the following way: A policy consists of a name, a set of categories (CPU, Memory, IO, Network and Others), and a set of metrics per category. Every metric defines a condition and a value (or range of values) compared with the measured value of that metric. It is possible to define four different conditions: (1) ”in”, meaning the measured value must be inside a specified range (2) ”out”, where the value must be outside the range, (3) ”above” and (4) ”below”, wherein each the value must be greater than, and respectively, lower than the specified value. Policies are evaluated using a system of weights in a bottom-up approach. A category is evaluated positively when the sum of the weights of metrics evaluated positively in a category reaches a threshold defined in the Ransomware Monitor. A policy is evaluated positively when the sum of weights of positively evaluated categories reaches another threshold, a graphical representation of the policy evaluation is given in Figure 5.2. The details of such mechanism, as well as the threshold values, are explained in Chapter 6. A policy can depend on another policy, meaning the policy is checked only if the policy it depends on is met.

This method is designed to allow an administrator to define the same metric multiple times with different weights based on the value that may be more or less representative

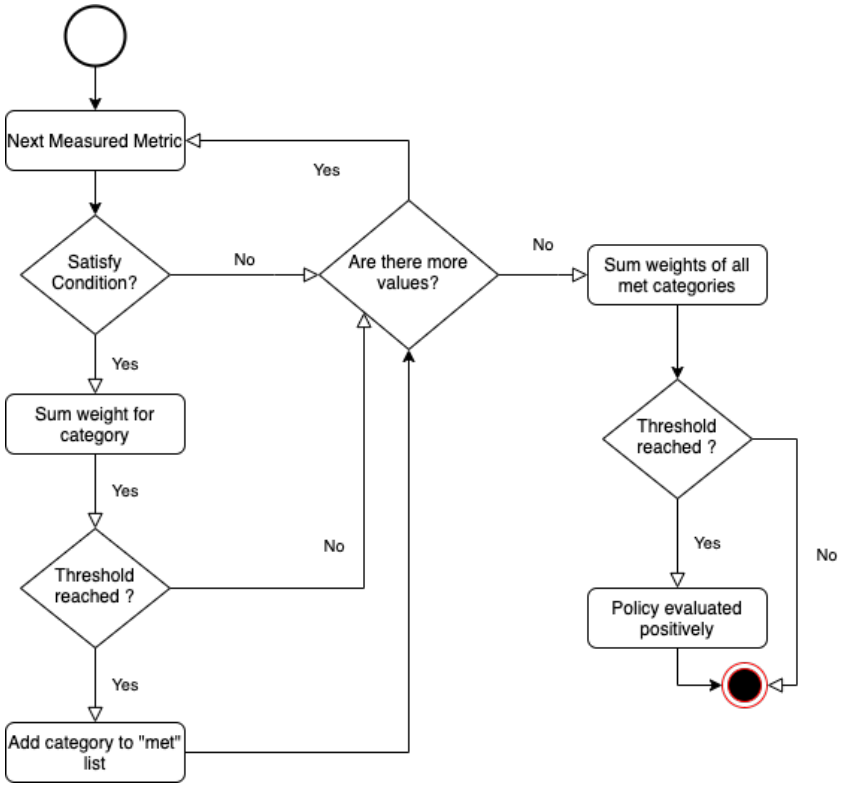

Figure 5.2: Flowchart Displaying the Policy Evaluation Flow

of the desired behaviour. Defining negative weights is also allowed, such that events that may indicate a false positive case can count negatively towards the category.

5.1.3 Management Module

This module is tasked with configuring the framework and providing an API to allow bidirectional communication between the visualisation module and the other modules of the framework. As the policy definition can be cumbersome and not easily readable, the UI also provides a way to edit, add and remove policy definitions understandably and straightforwardly, allowing to configure the Rule-Based Detection Module based on the particular needs of a system administrator. The Monitoring Module can also be configured via a configuration file provided by the Management Module in every device where it is possible to define the metrics to monitor, the monitoring window and enable the data transmission. Since the number of metrics can be extremely high, the Management Module comprises an Helper Tool able to provide insights on the data and generate basic policies automatically. This tool is described in the next chapter.

5.1.4 Visualization Module

It is essential that the results and measured data can be visualised and easily read, allowing an administrator to interpret the data and perform actions if a specific policy is met.

Therefore, a User Interface (UI) is provided in the form of a website accessible from the browser to visualise the data collected via line charts and the policy detection through a timeline chart, allowing the user to have an overview of the whole system. If a policy is evaluated positively, the metrics that met the policy condition are also highlighted. This interface can help a system administrator to visually see the difference between a normal condition and an attack and provide support for policy creation. The UI also provides an overview of the whole system by displaying all the devices present in the network and the last time they communicated with the framework to detect if a device has disconnected.

Chapter 6

Implementation

This chapter describes the implementation details of the proposed framework. In particular, the framework has been implemented as a Proof-of-concept in a Raspberry Pi hosting an ElectroSense sensor. The framework comprises four parts: (1) Ransomware Monitor, (2) Back-end, (3) Front-end and a (4) Helper Tool. In the following sections, the framework final implementation and the various software that is part of the solution are presented and discussed. The terminal commands are given under the assumption that a Linux-based system is used.

6.1 Monitoring Module

The Ransomware Monitor program has been created to collect the data related to the internal behaviour of the Raspberry Pi, detect, evaluate the policies, and distribute the processed data and the detection results. The program is written in Python, and it does not need the back-end to work. In this program, the Monitoring Module is implemented as *Monitors*, which comprise various classes tasked to collect the raw data from the device and process it. The *Monitors* are discussed in detail in subsection 6.1.1.

To start the program it is sufficient to run it without arguments, in case of using the binary version, the following command can be executed (6.1).

```
$ chmod +x monitor.bin
$ ./monitor.bin
```

Listing 6.1: Command to Run the Monitoring Process Binary

While a binary of the program built to run on the Raspberry Pi (32-bit) is given to run the Python scripts for development, the following dependencies and libraries are required.

- Perf tool [58]

  ```
  $ sudo apt install linux-tools-4.9
  ```

- Python 3
- pip: to install packages for Python

  ```
  $ sudo apt install python-pip
  ```

- Python libraries: psutil [59], used for hardware monitoring, pyyaml, used to read and write YAML files and pika, needed for RabbitMQ.

  ```
  $ pip install psutil pyyaml pika
  ```

- (Optional) pyinstaller: used to create the binary file

A screenshot of the command line output when starting the program is given in Figure 6.1.

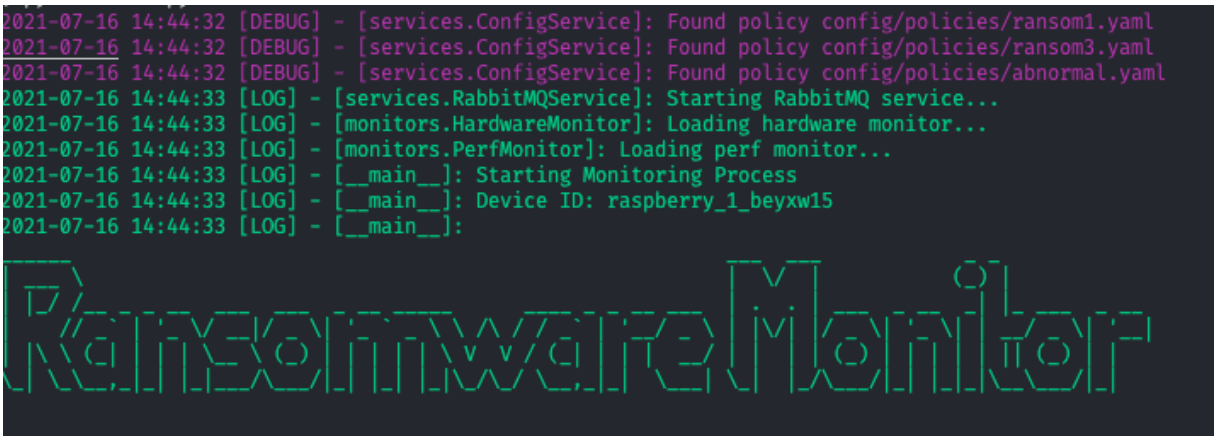

Figure 6.1: Screenshot of the Ransomware Monitor

After the program has been executed, it starts a loop, in which the data is collected and processed by the *Monitors*. This behaviour can be seen in Listings 6.2 and 6.3.

```
1    while self.monitor:
2      self.log.verbose('Measuring...')
3      data = self.monitorService.monitor()
4      ...
```

Listing 6.2: Snippet of the Loop in the *main.py* File.

```
1  def monitor(self):
2      data = { "time": int(time.time()) }
3      # Add other monitors here
4      perf_data = self.perf_monitor.monitor()
5      hw_data = self.hw_monitor.monitor()
6      if (len(perf_data.keys()) == 0):
7          time.sleep(config.get_perf_config()['monitor_window'])
8      for hw_key in hw_data.keys():
9          data[hw_key] = hw_data.get(hw_key)
```

```
10      for perf_key in perf_data.keys():
11          data[perf_key] = perf_data.get(perf_key)
12
13      policies_met = self.check_policies(data)
14      self.transmit_data(data, policies_met)
15      self.write_to_csv(data)
16      self.listen_for_new_commands()
17      return data
```

Listing 6.3: Snippet of the Monitor Method in *MonitorService.py* File. For Simplicity the Code Related to the Simulation Mode Has Been Removed

6.1.1 Monitors

The *Monitors* are components in charge of retrieving the raw data from the device and process it. To increase maintainability and extensibility, the *Monitors* are implemented as a subclass of the Monitor abstract class here included in the Listing 6.4.

```
1
2  from abc import ABC, abstractmethod
3
4  class AbstractMonitor(ABC):
5
6      def __init__(self):
7        super().__init__()
8
9      @abstractmethod
10     def get_field_names(self):
11         pass
12
13     @abstractmethod
14     def monitor(self):
15         pass
```

Listing 6.4: Monitor Abstract Class

While it is not possible in Python to provide a typed signature, each *Monitor* is expected to provide a *get_field_names* method, which returns a list of Strings with the name of every metric monitored and a *monitor* method which returns a dictionary with the monitored metrics their measured value as key-value pair.

In this proof-of-concept, two monitors are provided, a *HardwareMonitor* which uses the *psutil* library to read hardware related metrics, such as CPU usage, memory usage, and IO write operations and a *PerfMonitor*, which uses the Linux perf tool to read the HPC and kernel events. The *MonitorService* controls the *Monitors* and act as the core of the program by digesting the data retrieved by the *Monitors*.

6.2 Rule-Based Detection Module

The *Rule-Based Detection Module* is also implemented in the *Ransomware Monitor* in the form of the *PolicyService* class. In each loop of the program, the collected data is evaluated against a set of policies (this behaviour is shown in the form of pseudocode in Listing 6.5), some functionalities, such as the policy dependency mechanism and the other information included in the output of the algorithm have been excluded from the snippet for clarity.

```
1  algorithm policy-evaluation is
2    input: set of policies P,
3           set of measurements D
4
5    output: Set PP of policies evaluated positively
6
7    PP := []
8    for each policy p in P do
9        policy_score := 0
10       for each category c in p do
11           category_score := 0
12           for each metric m in c do
13               for each sample s in D do
14                   if m.name == s.name do
15                      if s.value meets m.condition do
16                        category_score := category_score + m.weight
17           if category_score > threshold do
18               policy_score := policy_score + c.weight
19       if policy_score > threshold do
20           PP.push(p.name)
```

Listing 6.5: Pseudocode for Policy Evaluation.

The system provides a human-readable and maintainable definition of policies, which can be declared as YAML files. An example of a policy is provided in Listing 6.6. The policies must be included in the *config/policies/* directory in the same directory where the binary of the *Ransomware Monitor* is present.

```
1 name: ransom1
2 depends_on: abnormal
3 metrics:
4   cpu:
5     - condition: in
6       name: iTLB-load-misses
7       value:
8         - 3981.45
9         - 824568.15
```

```
10        weight: 10
11    io: ...
12    memory: ...
13    network: []
14    others: ...
15  weights:
16    cpu: 200
17    io: 200
18    memory: 100
19    network: 0
20    others: 100
```

Listing 6.6: Example of *config/policies/ransom1.yaml* Policy. For Brevity Only One Metric Is Included

There are four main parts of the policy:

name: This is the policy name, can be any alphanumeric string, without spaces and must be different from all other policy names (please note that this proof of concept does not check for naming clashes, and therefore it assumes this name is always unique).

depends_on: here, it is possible to specify the name of another policy upon which the current is a subordinate. The current policy will not be checked if the specified policy is not met. If the policy should always be evaluated, it is possible to leave this part blank (i.e. "depends_on: ").

weights: In this section, it is possible to define the weight of every category towards the whole policy. A policy is evaluated positively if the sum of the weights of all the categories met is at least 500 ($\sum w_{metCategory} \geq 500$). In this example, the policy is therefore met if both the *cpu* and *io* category are met plus at least one between *memory* or *others*.

metrics: In the final section, for every category (i.e. CPU, Memory, IO, Network and Others), a list of metrics must be provided. The list can also be empty if the category does not need to be counted towards the policy. However, in that case, it is also important to set the weight of that category to 0.

For every metric, the following information must be defined

- **name**: name of the event as specified in the config file
- **condition**: condition that the metric must satisfy in order to be counted toward the category. Must be one of "in", "out", "above" or "below".
- **value**: the value(s) on which the condition is checked against the measured value. If the condition is "above" or "below", it must be a single float number; otherwise, a pair must be provided.
- **weight**: similarly to how the category weight count towards the policy, the metric weight counts toward the single category the event is in. The category is considered

met if the sum of all met metrics inside that category is equal or greater than 100 ($\sum w_{metMetric} \geq 100$). Also, in this case, it is possible to specify negative values in case a metric could indicate a false positive, and it is also possible to specify the same metric multiple times if different values should have different weights.

6.3 Management Module

The Management Module is implemented in different components. It comprises a back-end, which serves as the central control point for the framework, the configuration of the Ransomware Monitor and a Helper Tool to facilitate the policy creation process by the system administrator.

6.3.1 Back-end

The Back-end is responsible for two main tasks: (1) Store the data, (2) Allow for data transmission and communication between the framework components. As the framework may comprise many different devices, a centralized approach was necessary to allow the Fron-end to fetch and display the data and manage the policies for each separate device running the Ransomware Monitor. In this proof-of-concept the data is stored in memory (last 20 samples), and as a long-term storage a CSV file is stored in the */data* folder, named *deviceId_yyyy_m_d_data.csv*. The CSV file can subsequently be used to create the policies. This rather simple approach was chosen instead of a database to avoid adding unnecessary complexity for the proof-of-concept. However, a database implementation for the storage should be used in production.

Communication

There are three ways of communication enabled by the Back-end: HTTP Endpoint, Websockets and RabbitMQ. In order to communicate with the Ransomware Monitor, RabbitMQ [57] is used as a message broker. This method avoids the need to expose endpoints and open ports on the device while also providing a data retention mechanism in case of failure from both ends, as the messages will be re-delivered after a disruption has ended.

This mechanism works in two distinct parts:

After the Ransomware Monitor is executed, it creates a queue named after the device unique id. The Ransomware Monitor uses the queue to listen for commands after each loop. This behaviour can be seen in Listings 6.2 and 6.3. The commands and a description is provided in Table 6.1

The queue is deleted when the device disconnects from the RabbitMQ instance. Table Table 6.1 shows the four command accepted by the Ransomware Monitor.

In the second part, the Ransomware Monitor uses the following queues to send data for storage and declare its availability to the Back-end:

Table 6.1: List of Commands Accepted by the Ransomware Monitor

Command	Payload	Response	Description
CREATE_POLICY	Policy in JSON format	code 201 or 400	Command to create a new policy
EDIT_POLICY	Policy in JSON format	code 200, 400 or 404	Command to edit an existing policy
DELETE_POLICY	Policy name as string	code 200 or 404	Command to delete an existing policy
DECLARE	null	void	Ask device to declare itself

- **data**: This queue lives in the "monitoring" exchange, and it is used by every device to send data to the queue.
- **deviceDeclare**: Similarly, as for the data queue, this is also used by every device, and it is used to announce itself to the framework.

An HTTP API interface is also provided by the Back-end, such that the Front-end can fetch the data and manage the policies. The list of HTTP endpoints available in the Back-end is given in the Table 6.2

Table 6.2: API Endpoints Provided by the Back-end

HTTP Method	Endpoint	Response	Description
GET	/device/all	List of devices	Get list of all devices
GET	/device/:deviceID/data	Device data (last 20 min.)	Get data for the device
POST	/device/:deviceID/policy	Code 400: Malformatted policy or policy already exists Code 201: Created	Add new policy
PATCH	/device/:deviceID/policy/:policyName	Code 400: Malformatted policy Code 404: policy does not exist Code 200: Ok	Edit policy
DELETE	/device/:deviceID/policy/:policyName	Code 404: policy does not exist Code 200: Ok	Delete policy

Finally, WebSockets are used to provide real-time updates of data and policies met to the Front-end. The Back-end emits events on two topics, where device_id is the unique id generated by each Ransomware Monitor program: $<deviceId>$ and $<deviceId>\_policies$ with the latest received data and policies evaluation respectively.

6.3.2 Simulation Mode

The *Ransomware Monitor* has a simulation mode that can be used to test the framework and the policies without having to deploy it on the device. It can use previously collected data which will be used as data source by the Simuluation Monitor. For this mode, the data source must be present in form of a CSV file (as generated by the framework) in the *example_data* directory. The Ransomware Monitor must be then executed with the *–simulation* flag followed by the path of the dataset relatively to the *example_data* directory. Another flag, *–pause_sim_on* accepts as argument the name of a policy and will pause the execution if the specified policy is evaluated positively.

6.3.3 Ransomware Monitor Configuration

The program can be configured via a YAML configuration file and must be present in the same path as the monitoring process, under *config/config.yaml*. The configuration file

describes the behaviour of the program, and it must be correctly defined before starting the Ransomware Monitor. An example is provided in Listing 6.7.

```
general:
  deviceId: raspberry_1
  # deviceRndId: beyxw15 (generated randomly on first execution)
  logLevel: debug
  rabbitMQEnabled: true
  csvEnabled: false
device:
  newtork_interface: wlan0
 perf:
  monitor_window: 5 # in seconds
  process: perf_4.9
hw_events:
  cpu: ...
  io: ...
  memory: ...
  network: ...
perf_events:
  cpu: ...
  io: ...
  memory: ...
  network: ...
  others: ...
rabbitMQ:
  host: <ip address of rabbitMQ instance>
  user: <username>
  password: <password>
  port: <port>
```

Listing 6.7: Example of the Configuration File

The *general* section of the configuration defines the general settings of the process. It is possible to define a device ID to distinguish the device from others. A randomly generated string will also be appended automatically to the name to avoid name clashes. This string will be then automatically added in the configuration file as deviceRndId, in order for it to survive a program restart. The log level of the program (verbose, debug, log, warn, error) is also defined in this section as well as the activation of the RabbitMQ interface (*rabbitMQEnabled*) and / or the CSV data logging (*csvEnabled*). The *rabbitMQ* section is needed only if *rabbitMQEnabled* is set to True, and the coordinates to successfully connect and login to the rabbitMQ instance must be provided. In the *device* section, it is possible to define which network interface should be monitored for the network-related data. In the *perf* section the settings related to *perf* must be provided. The *monitor_window* defines the time window in seconds in which *perf* should count the events at every iteration, and it corresponds to the *sleep* argument when the *perf* command is executed. In the *process* setting, on the other hand, the name of the *perf* process must be defined. In most cases, this should be ”perf”, but as the correct perf binary is not available for the kernel used in

this Thesis, another version of perf was installed, making it necessary to be configurable on the framework for all use cases. In *hw_events* and *perf_events* the metrics/events that each monitor will collect must be declared and categorized. An example of a category and its metrics is given by the Listing 6.8.

```
1  cpu:
2    - timer:tick_stop
3    - armv7_cortex_a7/unaligned_ldst_retired/
4    ...
```

Listing 6.8: Example of Category and Metrics

6.3.4 Helper Tool

A tool to facilitate the creation of policies is also provided as a help to the administrators. The tool shows graphs from the data previously collected (shown in Figure 6.2) and can generate a policy that the administrator can subsequently modify and adapt to its needs. This tool is provided as a low-level aid to understand the collected information better while also suggesting possible policies for detecting wanted behaviours. It is essential to notice that the policies generated by this tool are not ready to use and are only meant as a suggestion that needs further manual tweaking. It requires a basic understanding of the Python programming language, as the settings must be changed in the code.

For the tool to work, a list of CSV files generated by either the Back-end or the Ransomware Monitor must be provided as a dictionary containing each the name of the data, the path of the CSV file and the colour used for the visualization.

```
1  data_files = {
2    'normal': {
3      'path': 'data/experiment_dark_01/normal.csv',
4      'color': '#00FF00'
5    },
6    'ransomware': {
7      'path': 'data/experiment_dark_01/darkradiation.csv',
8      'color': '#FF0000'
9    },
10 }
```

Listing 6.9: Example of Data Sources for the Helper Tool

The configuration file (*config.yaml*) used by the monitoring tool while the metrics were generated must be provided as well in the *config* folder. However, the *hw_events* and *perf_events* must be merged together under a single *event* group.

Figure 6.2: Screenshot of the Charts Generated by the Helper Tool. The Red Bands Indicate the Automatically Generated Policy for the Metric

Helper Tool Settings

The Helper Tool is highly configurable; however, it must be done in the source code as a graphical interface is not provided. The various settings and their values are explained in this sub-section.

remove_outliers: Boolean, which will try to remove outliers from the provided data by removing the data points which exceed the mean value by n times the standard deviation of the dataset.

cutoff: Is the n values mentioned in the *remove_outliers* setting

use_index_instead_of_time: If set to True, it will use the index of the datapoints as the x-axis instead of their relative time. If False, the x-axis will represent the time from the start of the dataset.

print_statisics When set to True, the tool will print in the console statistical information (max, min values, average and standard deviation) for each metric in every dataset.

```
1
2 ext4:ext4_evict_inode | normal | min: 0 | max: 0 | avg: 0.0 | std: 0.0
3 ext4:ext4_evict_inode | ransomware | min: 0 | max: 2627 | avg: 238.0 |
    ↪ std: 597.1031994297391
```

Listing 6.10: Example of Statistical Data Printed by the Helper Tool

exclude_columns an empty list by default, can be expanded by passing the names of the metrics the user wishes to exclude from both the visualization and the created policy.

The following settings are only related to the creation of the policy.

names_for_policy List of dataset names that will be taken into account for the policy creation (i.e. only the data of the specified datasets will be used).

weight_multiplier By default, each metric is assigned a weight such that every metric in a category must match in order to evaluate such category positively. The weight multiplier is simply a value that multiplies the default weight. This is achieved by the following formula $100/|M_c|$ where M is the set of all metrics in the category c.

offset_percent As the Tool takes the minimum and maximum value across the considered datasets, this setting allow the user to specify an offset that will increase the detection window.

policy_name The name of the policy to create

policy_condition one between 'in' or 'out'. It specifies the condition every metric in the policy will have. The tool is only able to apply the same condition for every metric.

compare_with_policy If set to true, only the value which are outside of the range of a given policy will be considered for the creation of the new policy. This mode is useful, for example, in case a policy that detects the normal behaviour already exists, and the user wishes to generate a policy by only using values outside the normal behaviour.

policy_to_compare If *compare_with_policy* is set to True, here the policy name of the compared policy must be specified. The YAML of the policy must be provided in the *comparison* folder inside the tool.

Automatic Policy Generation

In order to generate a policy, the settings must be provided and tweaked as described in the previous sub-section. There are two modes for the policy generation that differ slightly. In the first mode, all the desired data is used for the policy generation. In this mode, the *compare_with_policy* setting must be set to False. The tool will loop through all the desired datasets specified in *names_for_policy* and for each metric will register the maximum and minimum value for that particular metric across all the chosen datasets. The metric will be subsequently added to a policy with:

- condition: as specified in the *policy_condition* setting
- value: array comprising the $min_{value} - offset$ and $max_{value} + offset$
- weight: $100/|Category|$

In the second mode, the behaviour is similar. However, a metric will only be considered if, in at least one sample across all the specified datasets, the value of the metric does not fall within the given policy. Suppose an abnormal policy is provided, and the goal is to create a policy for a ransomware. In that case, this mechanism can quickly discard metrics that are not showing particular behaviours outside of the normal range.

6.4 Visualization Module

The *Visualisation Module* comprises the front-end interface implemented as a website and written in VueJS [60]. It enables the user to visualise all the devices available in the framework. A per-device page is provided with the visualisation of the evaluated policies in a timeline chart and displays the values of the monitored metrics in real-time. For each metric, it is also possible to see their historical data on a line chart. The metrics responsible for the positive evaluation of a policy are highlighted to provide visual insights to the user.

The user is initially presented with a page containing all the devices in the fleet represented in Figure 6.3 by calling the */device/all* endpoint in the backend.

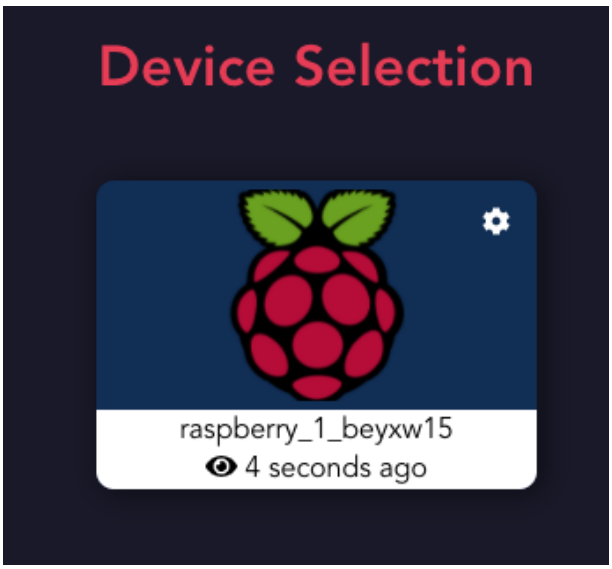

Figure 6.3: Screenshot of the Device Selection

When the user selects a device by clicking on the card, it is presented with the Device Page (shown in Figure 6.4); the data is initially collected for immediate display via an HTTP request to the */device/:deviceID/data* endpoint, the application then subscribes to the WebSocket topics related to the device in order to fetch new data in real-time continuously.

```
1 this.sockets.subscribe(this.deviceID, this.newDataHandler);
2 this.sockets.subscribe(this.deviceID + '_policies', this.policiesHandler);
```

Listing 6.11: Subscription to Websocket Events

The data handler will then push the new values to the existing data, triggering a re-render of the components holding the information and the charts. To minimise memory usage, only the previous 20 samples are displayed in the chart.

In the upper part, a timeline with the policy detection is displayed. On the bottom of the screen, the metrics are shown, separated by their belonging category. Above the policy detection chart, it is possible to select a policy. If the policy is evaluated positively in

the following sample, the metrics positively evaluated of that policy will be highlighted by a red shadow, as shown in Figure 6.4. When a metric is clicked, the card expands, showing the historical values collected for that particular metric, as visible in Figure 6.5. By clicking the chart again, the user can dismiss it and minimise the metric card.

Figure 6.4: Screenshot of the Device Page. The Policy Detection Timeline is Displayed in the Upper Part. A Highlighted Metric Can Be Seen by the Red Shadow.

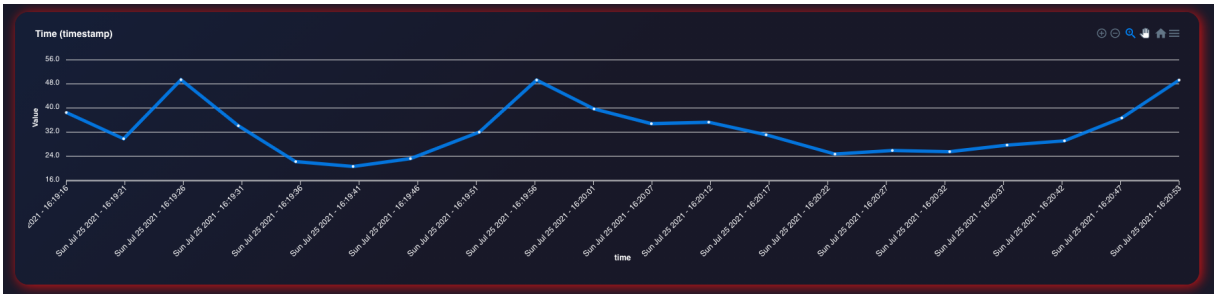

Figure 6.5: Screenshot of a Metric Chart.

Finally, the front-end enables the user to edit the policies via a graphical user interface by clicking on the gear icon in the Device Selection page, as shown in Figure 6.3. The user is presented with a menu displaying all the policies available for the particular device (shown in Figure 6.6).

The user can subsequently select a policy to modify by clicking on the ”edit” icon. Remove the policy by clicking on the trash icon or adding a new policy by clicking on the bottom bar with the plus icon. For the policy creation and modification a Policy Editor is implemented (shown in Figure 6.7). The Policy Editor can implement every aspect of the policy. With the editor support, the user can modify the weight of the categories, add and remove metrics, specify the condition, values, and weight for each metric and select if the policy must depend on another policy.

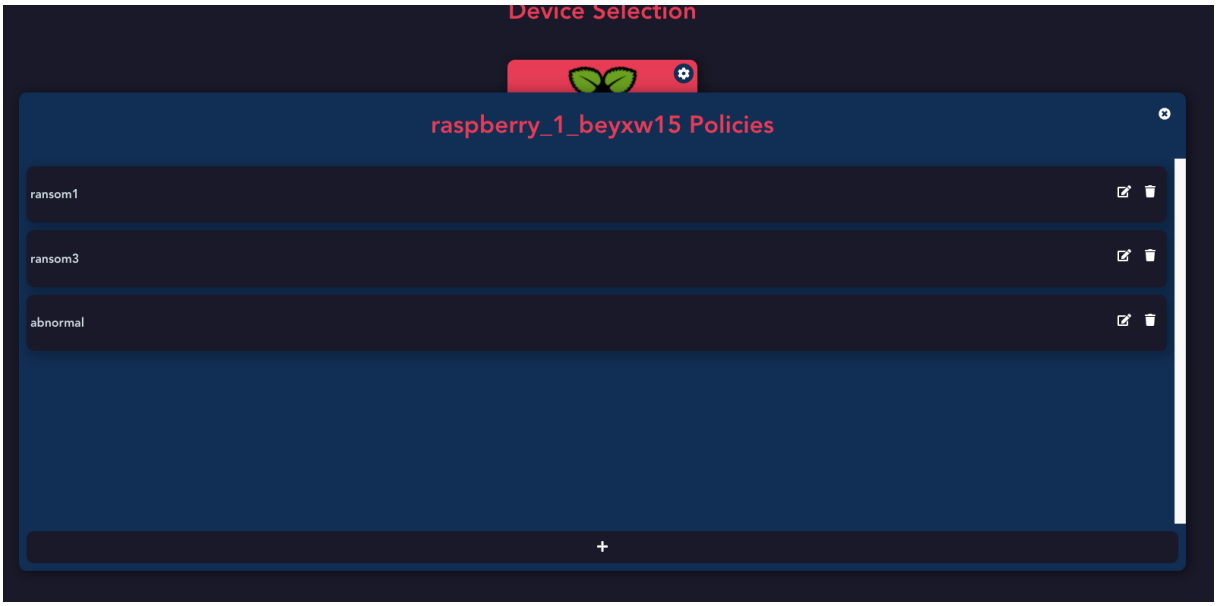

Figure 6.6: Screenshot of the Device Configuration

Figure 6.7: Screenshot of the Policy Editor

Chapter 7

Evaluation

7.1 Experiment setup

For the evaluation an *Experiment* folder was prepared on another system, containing a *monitor-bin* folder with the binary of the monitoring process, the configuration and the following policies: *abnormal*, *ransom1*, *ransom2*. The *abnormal* policy evaluates abnormal behaviour while *ransom1* and *ransom2* are policies for the detection of the Ransomware-PoC and DarkRadiation respectively.

The policies were created before the evaluation by collecting the data of the device multiple times under normal circumstances, under the attack of both ransomware and during an "abnormal" usage of the device. For the purpose of this work, abnormal usage is intended as any usage of the device outside of the *ElectroSense* context, such as installing packages and zipping directories.

For normal usage, three use cases can be identified: (1) Starting the device and leaving it on without further actions, (2) log in to the *ElectroSense* platform and use the spectrum monitor, (3) Use the decoder functionality of the *ElectroSense* platform.

Two bash scripts are also included: *setup.sh* which installs the dependecies needed (i.e. the *perf* tool) and *random_file_generator.sh* a scripts which generates a random number of files with different sizes in the */home/electrosense* directory.

For each experiment, the folder was copied to the device via SSH in the root directory, the device was then accessed via SSH as *root*, and the setup file was executed.

```
$ scp -r /Users/timp4w/OneDrive/UZH/Thesis/Monitor/Experiment root@192
   ↪ .168.1.10:/root/
$ ssh root@192.168.1.10:/root/
... after succesfull login
$ cd Experiment
$ chmod +x setup.sh
$ ./setup.sh
```

Listing 7.1: Command to Copy the Experiment Folder

In order to evaluate the efficacy of each policy, their *depends_on* property was set to *null* for the experiments such that they were always evaluated.

7.1.1 Policies

Three policies were created, one that detects abnormal behaviour and two for the two different ransomware. The metrics monitored are omitted from this report for brevity, however the policies are included in the CD.

The policy for the abnormal behaviour was created with the Helper Tool by providing multiple samples of the device during normal usage. It contains 113 metrics, which comprises all monitored metrics. The list of metrics has been omitted for brevity. Every category has a weight of 500, as the policy tries to detect any abnormal behaviour. The weight of the metrics has been slightly adjusted; however, it is the same for each category. This is summarised in Table 7.1.

Table 7.1: Abnormal Policy Metrics and Weights

Category	**CPU**	**IO**	**Memory**	**Network**	**Others**
Category weight	500	500	500	500	500
Metrics weight	16.0	18.0	35.0	40.0	45.85
Number of metrics	59	28	9	11	6

The policy for ransomware 1 is summarised in Table 7.2. Similarly, as for the abnormal policy, every category has the same weight for each metric. The policy was also generated via the Helper Tool by providing the abnormal policy as a comparison. The weights were subsequently manually adjusted, and some metrics were removed. In Figure 7.1 an example of a policy for a single metric generated by the tool is shown. The red lines indicate samples captured during the ransomware attack, the green lines the normal behaviour. The red bar, indicates the policy threshold window. It can be noticed how the policy excluded the samples which were inside the limits of the normal behaviour.

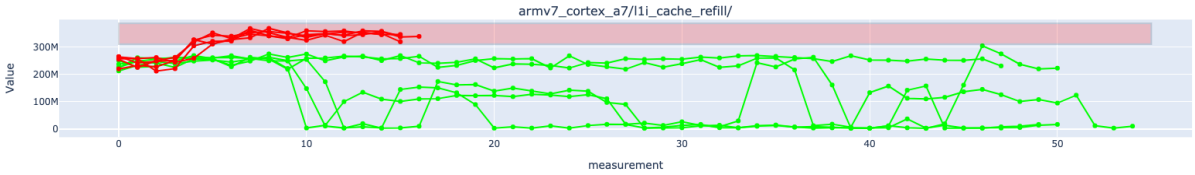

Figure 7.1: Example of Policy Generated for a Metric to Detect the Ransomware in Comparison Mode With the Abnormal Behaviour Policy.

Table 7.2: Ransomware 1 Policy Metrics and Weights

Category	**CPU**	**IO**	**Memory**	**Network**	**Others**
Category weight	250	250	100	0	100
Metrics weight	35	17.5	50	-	100
Number of metrics	4	15	2	0	1

Finally, the policy for ransomware 2 is summarised in Table 7.3. This policy was also generated via the Helper Tool with the abnormal policy as a comparison and then manually adjusted. In this case it was necessary to specify different weights for some metrics in the CPU category, which were deemed more representatives of a DarkRadiation infection. However, the categories weights are the same as for ransomware 1, i.e. 250 for CPU, 250 for IO, 100 for both Memory and Others and 0 for Network.

Table 7.3: Ransomware 2 Policy Metrics and Weights

Category	**CPU**	**IO**	**Memory**	**Network**	**Others**
Category weight	250	250	100	0	100
Metrics weight	20-50	13.71	50	-	33.34
Number of metrics	7	17	2	0	3

7.2 Experiments

A total of six experiments were performed, each of them executed five times. The symbol ✓ indicates a positive evaluation of the policy, while ✗ is respectively a negative evaluation (i.e. the behaviour that the policy evaluates was not detected). When none of the policies is detected, normal behaviour is assumed.

7.2.1 Experiment 1

In the first experiment, the normal usage was considered by evaluating if the framework falsely reported this behaviour as abnormal or matching it with the two ransomware.

The monitoring process was started, and the device was used for 7 minutes and 30 seconds, performing only actions designed to mimic the standard use-cases of the *ElectroSense* platform. Only the following actions were performed:

1. Not using the device at all.
2. Use the spectrum monitor in the *ElectroSense* platform by actively reloading the page, change mode and aggregation method.
3. Use the Spectrum Decoder in the *ElectroSense* platform and change mode between FM Radio, AM Radio, ADS-B, AIS, ACARS and LTE, while also choosing different centre frequencies and editing the Antenna Gain.

The results are summarised in Table 7.4.

None of the policies was evaluated positively in this experiment, having each of them a False Positive Rate (FPR) of 0% and a True Negative Rate (TNR) of 100%. The framework is hence able to identify the normal behaviour of the device correctly.

Table 7.4: FPR and TNR of the Framework Policies When Detecting Normal Behaviour in the ElectroSense Sensor.

Policy	abnormal			ransomware1 (Ransomware-PoC)			ransomware2 (DarkRadiation)		
# Run	Ev.	FPR	TNR	Evaluation	FPR	TNR	Evaluation	FPR	TNR
1	✗	0%	100%	✗	0%	100%	✗	0%	100%
2	✗	0%	100%	✗	0%	100%	✗	0%	100%
3	✗	0%	100%	✗	0%	100%	✗	0%	100%
4	✗	0%	100%	✗	0%	100%	✗	0%	100%
5	✗	0%	100%	✗	0%	100%	✗	0%	100%

7.2.2 Experiment 2

In the second experiment, the framework ability to detect abnormal behaviour was evaluated. For abnormal behaviour, it was considered the installation of a package in the system (command provided in Listing 7.2. The time in seconds from the start of the experiment was recorded on the command execution and the time when the command ended.

```
$ sudo apt-get install libboost-all-dev
```

Listing 7.2: Command Used to Install the Package

The results are shown in Table 7.5. The abnormal behaviour was correctly identified in all five runs of the experiment. While obtaining 0 false positives from the ransomware1 policy, there were two false positives in the ransomware2 evaluation, both occurring only in one measurement.

Table 7.5: TPR, FPR and TNR of the Framework Policies When Detecting Abnormal Behaviour (Package Installation) in the ElectroSense Sensor.

			abnormal			**ransom1 (Ransomware-PoC)**			**ransomware2 (DarkRadiation)**		
	# Run	Samples	Ev.	TPR	FNR	Evaluation	FPR	TNR	Evaluation	FPR	TNR
	1	16	✓	100%	0%	✗	0%	100%	✗	0%	100%
	2	15	✓	100%	0%	✗	0%	100%	✗	0%	100%
	3	9	✓	100%	0%	✗	0%	100%	✓	11.11%	88.89%
	4	8	✓	100%	0%	✗	0%	100%	✗	0%	100%
	5	9	✓	100%	0%	✗	0%	100%	✓	11.11%	88.89%
Total		**57**		**100%**	**0%**		**0%**	**100%**		**3.51%**	**96.49%**

The ransomware2 policy incorrectly identified the package installation in 2 predictions. As the DarkRadiation ransomware also installs packages, this behaviour might have been picked up as well during the creation of the policy and would explain the false positive prediction.

7.2.3 Experiment 3

The goal of the third experiment is to evaluate the ability of the framework to identify the abnormal behaviour caused by the compression of files and to not flag this behaviour as

ransomware. For this purpose, a random folder was generated with the the code referenced in Listing 7.3.

```
cd /home/electrosense
mkdir random_data
cd random_data
mkdir random_data_big
mkdir random_data_small
cd random_data_big
seq -w 1 $(shuf -i8-15 -n1) | xargs -n1 -I% sh -c 'dd if=/dev/urandom of=
    ↪ file.% bs=$(shuf -i10000-60000 -n1) count=1024'
cd ..
cd random_data_small
seq -w 1 $(shuf -i100-500 -n1) | xargs -n1 -I% sh -c 'dd if=/dev/urandom
    ↪ of=file.% bs=$(shuf -i1-10 -n1) count=1024'
```

Listing 7.3: Random File Generator Script

The *random_data* folder was subsequently compressed with the *tar* command:

```
$ tar -zcvf example.tar.gz random_data
```

Listing 7.4: Command to Compress the Folder

The results are shown in Table 7.6.

Table 7.6: TPR, FPR and TNR of the Framework Policies When Detecting Abnormal Behaviour (Folder Compression) in the ElectroSense Sensor.

			abnormal			**ransom1 (Ransomware-PoC)**			**ransomware2 (DarkRadiation)**		
	# Run	Samples	Ev.	TPR	FNR	Evaluation	FPR	TNR	Evaluation	FPR	TNR
	1	19	✓	100%	0%	✗	0%	100%	✗	0%	100%
	2	5	✓	100%	0%	✗	0%	100%	✗	0%	100%
	3	8	✓	87.5%	12.5%	✗	0%	100%	✗	0%	100%
	4	10	✓	100%	0%	✗	0%	100%	✗	0%	100%
	5	9	✓	88.89%	11.11%	✗	0%	100%	✗	0%	100%
Total		**51**		**96.08%**	**3.92%**		**0%**	**100%**		**0%**	**100%**

The compression behaviour was not always detected by the abnormal policy, although a high TPR can be observed. None of the ransomware policies wrongly flagged it achieving a 0% FPR.

7.2.4 Experiment 4

In the fourth experiment, the framework ability to correctly identify the *Ransomware-PoC* was tested.

The ransomware was executed with the command shown in Listing 7.5.

```
$ cd Experiment/Ransomwares/ransomware1
$ ./ransom.bin
```

Listing 7.5: Command to Run the Ransomware 1

The ransomware was executed 30 seconds after the start of the monitoring. As the ransomware encrypts some critical files needed for the correct operation of ElectroSense, the platform automatically restarted the Raspberry, denoting the end of each run. In order to correctly measure the detection rate, the samples collected before the ransomware was executed were not considered.

The results are shown in Table 7.7.

Table 7.7: TPR, FPR and TNR of the Framework Policies When Detecting Ransomware 1 in the ElectroSense Sensor.

			abnormal			**ransom1 (Ransomware-PoC)**			**ransomware2 (DarkRadiation)**		
	# Run	Samples	Ev.	TPR	FNR	Evaluation	TPR	FNR	Evaluation	FPR	TNR
	1	12	✓	91.67%	8.33%	✓	91.67%	8.33%	✗	0%	100%
	2	11	✓	100%	0%	✓	90.91%	9.09%	✗	0%	100%
	3	12	✓	100%	0%	✓	100%	0%	✗	0%	100%
	4	12	✓	91.67%	8.33%	✓	91.67%	8.33%	✗	0%	100%
	5	12	✓	91.67%	8.33%	✓	91.67%	8.33%	✓	8.33%	91.66%
Total		**59**		**94.91%**	**5.09%**		**93.22%**	**6.78%**		**1.70%**	**98.30%**

All the runs were flagged as abnormal and correctly identified the ransomware. Only a small number of predictions were incorrect, with the ransomware being detected between 10 and 20s from execution. The ransomware2 policy incorrectly identified a sample, however the FPR is very low at only 1.70%.

7.2.5 Experiment 5

The goal of the fifth experiment was to correctly identify the *DarkRadiation* ransomware when files were present in the */home/electrosense* folder. On a fresh installation, the */home* directory and its sub-directories are empty.

As *DarkRadiation* has three targets: (1) file with *.py*, *.txt* and *.sh* extensions, (2) */home* directory and (3) database files, a scenario when files are present in the */home* directory was taken into consideration.

All runs in this experiment were launched after each run of the 3rd experiment. The same files were present to evaluate the precision of the policy, with the exact same files being compressed.

Only the measurement after the ransomware had already infected the system and before starting the encryption until the ransomware exited (i.e. after receiving the message "sensor: encrypt db files Done. Delete files.") were taken into consideration. This is defined as the next sample after receiving the "sensor: encrypt grep files started." from

the ransomware Telegram Bot. The installation of the dependencies of *DarkRadiation* was therefore not taken into consideration in this evaluation.

The results are shown in Table 7.8.

Table 7.8: TPR, FPR and TNR of the Framework Policies When Detecting Ransomware 2 and Randomly Generated Files Are Present in the /home Directory in the ElectroSense Sensor.

			abnormal			**ransom1 (Ransomware-PoC)**			**ransomware2 (DarkRadiation)**		
	# Run	Samples	Ev.	TPR	FNR	Evaluation	FPR	TNR	Evaluation	TPR	FNR
	1	10	✓	80%	20%	✗	0%	100%	✓	60%	40%
	2	7	✓	85.71%	14.29%	✗	0%	100%	✓	57.14%	42.86%
	3	10	✓	90%	10%	✗	0%	100%	✓	50%	50%
	4	12	✓	91.67%	8.33%	✗	0%	100%	✓	58.33%	41.67%
	5	10	✓	100%	0%	✗	0%	100%	✓	50%	50%
Total		**49**		**89.80%**	**10.20%**		**0%**	**100%**		**55.10%**	**44.90%**

The ransomware was identified in all cases between 10s and 20s from execution, however, with a poor TPR. As not many files are present in the device, the first and third encryption phases of the ransomware are not correctly identified by the policy.

```
/usr/lib/python3/dist-packages/click/_compat.py (4)
real 0m10.098s
user 0m0.767s
sys 0m1.799s
```

Listing 7.6: Result of the Time Measurement of the *grep* Command Executed by DarkRadiation for Phase 1

```
/var/log/boot.log (6.6K)
/var/lib/apt/listchanges.db (17K)
/var/lib/apt/listchanges-old.db (12K)
/usr/bin/traceproto.db (2.9K)
/usr/bin/traceroute.db (56K)
/usr/bin/lft.db (2.5K)
/usr/share/doc/python3.5/pybench.log (36)
/usr/sbin/tcptraceroute.db (1.6K)
/lib/firmware/regulatory.db (4.0K)
real 0m16.629s
user 0m15.829s
sys 0m0.582s
```

Listing 7.7: Result of the Time Measurement of the *grep* Command Executed by DarkRadiation for Phase 2. In the brackets the file size was added for reference.

Given the low amount of files and their small size, specifically 11 files weighing 0.1MB in total, the policy was not always able to detect those phases, hence the relatively high FNR of 44.90% was observed.

7.2.6 Experiment 6

In the last experiment, it was again tested the framework capability of identifying the *DarkRadiation* Ransomware, however in this experiment the */home* directory was left as is, without the file generation applied to experiments 5 and 3. The start and end of the experiments, are defined as in experiment 5, i.e. only the encryption phase was taken into consideration.

The results are shown in Table 7.9.

Table 7.9: TPR, FPR and TNR of the Framework Policies When Detecting Ransomware 2 And the /home Directory Is Empty in the ElectroSense Sensor.

			abnormal			**ransom1 (Ransomware-PoC)**			**ransomware2 (DarkRadiation)**		
	# Run	Samples	Ev.	TPR	FNR	Evaluation	FPR	TNR	Evaluation	TPR	FNR
	1	4	✓	75%	25%	✗	0%	100%	✓	25%	75%
	2	3	✓	66.67%	33.33%	✗	0%	100%	✓	33.33%	66.67%
	3	4	✓	50%	50%	✗	0%	100%	✓	25%	75%
	4	4	✓	75%	25%	✗	0%	100%	✓	25%	75%
	5	3	✓	100%	0%	✗	0%	100%	✓	66.67%	33.33%
Total		**18**		**72.22%**	**27.78%**		**0%**	**100%**		**33.33%**	**66.67%**

In all the runs, the ransomware was correctly detected, but only on a single measurement. As shown in Experiment 5, there is only a small amount of files in the system affected by this ransomware when the */home* directory is empty.

The policy was still able to detect the ransomware, and abnormal behaviour was also detected, but with a shallow rate, indicating that the policy might be able to detect the *grep* command traversing the directories.

7.3 Discussion & Limitations

It can be observed from the experimental results that the framework is capable of correctly detect and identify the two ransomware taken into consideration by this Thesis and the abnormal behaviour generated by installing a package and compressing a folder.

The most promising results are given by the ransomware 1 detection, with an FPR of 0% and being able to always correctly identify the ransomware between 10 and 20 seconds after execution. Although three false-negative predictions can be observed, the experimental results show that the framework can identify the ransomware with high accuracy.

The DarkRadiation ransomware, although also correctly identified in all experiments, it showed a 3-4% FPR on Experiment 2 and 4 and a small rate (55.10%) of correct predictions during the encryption stage. As the normal usage of the device as part of *ElectroSense* does not involve direct manipulation, the obtained results show promising results on the ability of the framework to correctly identify a *DarkRadiation* infection of the device. However, given the small number of files being encrypted with an empty */home* directory, it was necessary for the policy to have a high sensitivity on file manipulation actions.

The Helper Tool was of great importance in the policy creation process, as creating the policies manually was a time-consuming and error-prone task. The Tool has allowed a quick generation of policies, then evaluated thanks to the Visualization provided by both the front-end and the graphs generated by the Helper Tool.

Chapter 8

Summary and Conclusions

In this work, a policy-based framework considering behavioral fingerprinting was designed and implemented to detect anomalies and ransomware affecting IoT devices. Heterogeneous data sources such as hardware metrics, kernel tracepoints and HPCs are monitored to create device behavioural fingerprints. As a proof of concept, the framework has been implemented and deployed in a crowdsensing platform called *ElectroSense*. In such a scenario, a set of policies to detect anomalies and several ransomware behaviors where designed and implemented as a proof-of-concept. After that, two ransomware belonging to different families were executed on the device and the behavior device was collected. The policies were finally evaluated in a series of experiments to measure their efficacy and the capability of the framework in identifying the two malware samples.

The experiments showed promising results in the ability of the framework to detect the malicious behaviour of the two ransomware, being able to correctly identify both between 10 and 20 seconds from execution even when only 0.1MB of files were encrypted. However, this solution efficacy is limited by the number of files (and their size) target by the ransomware. Experimental results reveal a 3-4% false positive rate on the ransomware affecting the lowest amount of files in the system. However, a low true positive rate (55.10%) of the predictions was observed if randomly generated files were present in the system and only a 33.33% TPR when the malware targeted a minimal amount of files.

Future work will focus on increasing the accuracy of the policies and decreasing the time required for detection by limiting the number of metrics being monitored. An attempt to automatically create the policies via machine learning techniques will also be attempted to investigate if a higher accuracy can be achieved with less time effort when this process is not done manually.

Bibliography

[1] Zumwald, M., Knüsel, B., Bresch, D., & Knutti, R. (2021). Mapping urban temperature using crowd-sensing data and machine learning. In *Urban Climate, 35*(1), 100739.

[2] ElectroSense Website. `https://electrosense.org/#!/`. Last visit July 11, 2021.

[3] ElectroSense Github Repository. `https://github.com/electrosense/es-sensor`. Last visit July 11, 2021.

[4] Rajendran, S., Calvo-Palomino, R., Fuchs, M., Van den Bergh, B., Cordob'es, H., Giustiniano, D., Pollin, S., & Lenders, V. (2018). ElectroSense: Open and Big Spectrum Data. *IEEE Communications Magazine, 56*(1), 210–217.

[5] Rajendran, S., Calvo-Palomino, R., Fuchs, M., Van den Bergh, B., Cordobés, H., Engel, M., Giustiniano, D., Pollin, S., Jain, P., Liechti, M., Schäfer, M., & Lenders, V. (2020). ElectroSense+: Crowdsourcing radio spectrum decoding using IoT receivers. *Computer Networks, 174*, 107231.

[6] Jetvision Website. `https://shop.jetvision.de/epages/64807909.sf/en_GB/?ObjectPath=/Shops/64807909/Categories/ElectroSense_Kits`. Last visit July 11, 2021.

[7] Raspberry Blog. `https://www.raspberrypi.org/blog/raspberry-pi-silicon-pico-now-on-sale/`. Last visit July 11, 2021.

[8] Cisco Website. `https://tools.cisco.com/security/center/resources/virus_differences`. Last visit July 11, 2021.

[9] HomeAssistant Website. `https://www.home-assistant.io/` . Last visit July 11, 2021.

[10] Virustotal `https://www.virustotal.com/gui/`. Last visit July 17, 2021.

[11] Xu, T., Wendt, J., & Potkonjak, M. (2014). Security of IoT systems: Design challenges and opportunities. In Proceedings of the *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, San Jose, USA, pp. 417–423.

[12] Wurm, J., Hoang, K., Arias, O., Sadeghi, A., & Jin Y. (2016). Security analysis on consumer and industrial IoT devices. In Proceedings of the *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, Macao, China, pp. 519–524.

[13] Meneghello, F., Calore, M., Zucchetto, D., Polese, M & Zanella, A. (2019). IoT: Internet of Threats? A Survey of Practical Security Vulnerabilities in Real IoT Devices. *IEEE Internet of Things Journal, 6*(5), 8182–8201.

[14] Reuters `https://www.reuters.com/article/us-cyber-attack-europol-idUSKCN18A0FX`. Last visit July 11, 2021.

[15] The New York Times. `https://www.nytimes.com/2020/09/18/world/europe/cyber-attack-germany-ransomeware-death.html`. Last visit July 11, 2021.

[16] Reuters. `https://www.reuters.com/world/us/far-colonial-pipeline-panic-buying-leaves-florida-cities-short-gas-2021-05-13/` Last visit July 11, 2021.

[17] The New York Times. `https://www.nytimes.com/2021/05/11/business/colonial-pipeline-shutdown-latest-news.html` Last visit July 11, 2021.

[18] Forbes. `https://www.forbes.com/sites/rrapier/2021/05/11/panic-buying-is-causing-gas-shortages-along-the-colonial-pipeline-route/` Last visit July 11, 2021.

[19] Fortune. `https://fortune.com/2021/05/12/colonial-pipeline-back-panic-buying-chaos-pump/` Last visit July 11, 2021.

[20] Ransomware PoC GitHub repository. `https://github.com/jimmy-ly00/Ransomware-PoC` Last visit July 15, 2021.

[21] Nullaray's Cypher Github Repository `https://github.com/NullArray/Cypher` Last visit July 21, 2021.

[22] Trendmicro analysis of DarkRadiation. `https://www.trendmicro.com/en\_us/research/21/f/bash-ransomware-darkradiation-targets-red-hat--and-debian-based-linux-distributions.html` Last visit July 15, 2021.

[23] Ferrante, A., Malek, M., Martinelli, F., & Milosevic, J. (2017). Extinguishing Ransomware - A Hybrid Approach to Android Ransomware Detection. In Imine A., Fernandez J., Marion JY., Logrippo L., Garcia-Alfaro J. (eds) *Foundations and Practice of Security* (pp. 242–258). Cham: Springer.

[24] Sgandurra, D., Muñoz-Gonzâlez, L., Mohsen, R., & Lupu, E. C. (2016). Automated Dynamic Analysis of Ransomware: Benefits, Limitations and use for Detection. *CoRR:abs/1609.03020.*

[25] Cabaj, K., Gregorczyk, M., & Mazurczyk, W., (2017). Software-Defined Networking-based Crypto Ransomware Detection Using HTTP Traffic Characteristics. In *Computers & Electrical Engineering, 66*, 353–368.

[26] Almashhadani, A. O., Kaiiali, M., Sezer, S., & O'Kane, P. (2019). A Multi-Classifier Network-Based Crypto Ransomware Detection System: A Case Study of Locky Ransomware. In *IEEE Access 7*, 47053–47067.

[27] Vinayakumar, R., Soman, K., Senthil Velan, K. K., & Ganorkar, S. (2017). Evaluating shallow and deep networks for ransomware detection and classification. In Proceedings of the *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Udupi, India, pp. 259-265.

[28] Bae, S. I., Bin, G. L., & Im, E. G. (2020). Ransomware detection using machine learning algorithms. Concurrency and Computation: Practice and Experience. *Concurrency and Computation Practice and Experience 32*(18), e5422.

[29] Alhawi, O. M. K., Baldwin, J., & Dehghantanha, A. (2018). Leveraging Machine Learning Techniques for Windows Ransomware Network Traffic Detection. In Conti, M. Dargahi, T. (eds). *Cyber Threat Intelligence*(pp. 93–10). Cham: Springer.

[30] Kharraz, A., Arshad, S., Mulliner, C., Robertson, W., & Kirda, E. (2016). UNVEIL: a large-scale, automated approach to detecting ransomware. In Proceedings of the *25th USENIX Conference on Security Symposium (SEC'16)*, Austin, TX, USA, pp. 757–772.

[31] Maniath, S., Ashok, A., Poornachandran, P., Sujadevi, V.G., Sankar A.U. P., & Jan, S. (2017). Deep learning LSTM based ransomware detection. In Proceedings of the *Recent Developments in Control, Automation & Power Engineering (RDCAPE)*, Noida, India, pp. 442–446.

[32] Maiorca D., Mercaldo, F., Giacinto, G., Visaggio, C. A., & Martinelli, F. (2017). R-PackDroid: API package-based characterization and detection of mobile ransomware. In Proceedings of the *Symposium on Applied Computing (SAC '17)*, Marrakech, Morocco, pp. 1718–1723.

[33] Scaife, N., Carter,H., Traynor, P., & Butler, K. R. B. (2016) CryptoLock (and Drop It): Stopping Ransomware Attacks on User Data. In Proceedings of the *IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, Nara, Japan, pp. 303–312.

[34] Cusack, G., Michel, O., & Keller, E. (2018). Machine Learning-Based Detection of Ransomware Using SDN. In Proceedings of the *ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization (SDN-NFV Sec'18)*, Tempe, AZ, USA, pp. 1–6.

[35] Hwang, J., Kim, J., Lee, S., & Kim, K. (2020). Two-Stage Ransomware Detection Using Dynamic Analysis and Machine Learning Techniques. *Wireless Personal Communication, 112*, 1–13.

[36] Jung, S., & Won, Y. (2018). Ransomware detection method based on context-aware entropy analysis. *Soft Computing 22*, 6731–6740.

[37] Morato, D., Berrueta, E., Magaña, E., & Izal, M. (2018). Ransomware early detection by the analysis of file sharing traffic. *Journal of Network and Computer Applications 124*, 14–32.

[38] Kharraz, A., Robertson, W. K., Balzarotti, D., Bilge, L., & Kirda, E. (2015). Cutting the Gordian Knot: A Look Under the Hood of Ransomware Attacks. In Proceedings of the *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, Milan, Italy, pp. 3–24.

[39] Alam, M., Bhattacharya, S., Mukhopadhyay, D., & Chattopadhyay, A., Sinha, S., Dutta, S. (2018). RAPPER: Ransomware Prevention via Performance Counters. *arXiv preprint, ArXiv:abs/1802.03909*.

[40] Cloudflare. `https://www.cloudflare.com/it-it/learning/ddos/glossary/malware/` Last visit July 11, 2021.

[41] Chess, D. M., & White, S. R. (2000). An undetectable computer virus. In Proceedings of the *Virus Bulletin Conference*, Orlando, FL, USA, 2000.

[42] Ömer, A., & Refik, S. (2020). A Comprehensive Review on Malware Detection Approaches. *IEEE Access, 8*, 6249–6271, 2020.

[43] Cohen, F. (1987). Computer viruses: Theory and experiments. *Computers & Security, (6)*1, 22–35.

[44] Lysne, O. (2018). Static Detection of Malware. In (ed). *The Huawei and Snowden Questions* (pp. 57–66). Cham: Springer.

[45] Sihwail, R., Omar, K., & Ariffin, K. A. Z. (2018). A Survey on Malware Analysis Techniques: Static, Dynamic, Hybrid and Memory Analysis. *International Journal on Advanced Science Engineering and Information Technology 8*(4-2), 1662.

[46] Damodaran, A., Di Troia, F., Visaggi, C. A., & Austin, T. H. (2017). A comparison of static, dynamic, and hybrid analysis for malware detection. *Journal of Computer Virology and Hacking Techniques 13*(1), 1–12.

[47] Souri, A., & Hosseini, R. (2018). A state-of-the-art survey of malware: detection approaches using data mining techniques. *Human-centric Computing and Information Sciences 8*(1), 1-22.

[48] Alazab, M., Venkatraman, S., & Watters, P. (2010). Towards understanding malware behaviour by the extraction of API calls. In Proceedings of the *2nd Cybercrime and Trustworthy Computing Workshop (CTC)*, Ballarat, VIC, Australia, pp. 52–59.

[49] Bazrafshan, Z., Hashemi, H., Fard, S.M.H. & Hamzeh, A. (2013). A survey on heuristic malware detection techniques. In *Information and Knowledge Technology*, pages 113–120, 2013.

[50] Idika, N., & Mathur, A. P. (2007). A survey of malware detection techniques. *Purdue University*, pp. 48.

[51] Sánchez Sánchez, P. M., María Jorquera Valero, J., Huertas Celdrán, A., Bovet, G., Gil Pérez, M., & Martínez Pérez, G. (2020). A Survey on Device Behavior Fingerprinting: Data Sources, Techniques, Application Scenarios, and Datasets. arXiv e-prints, arXiv-2008.

[52] Golomb, T., Mirsky, Y., Elovici, Y. (2018). CIoTA: Collaborative IoT Anomaly Detection via Blockchain. *arXiv preprint ArXiv:abs/1803.03807*.

[53] Wang, X., Konstantinou, C., Maniatakos, M., & Karri, R. (2015). ConFirm: Detecting firmware modifications in embedded systems using Hardware Performance Counters. In Proceedings of the *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Austin, TX, USA, pp. 544–551.

[54] Barbhuiya, S., Papazachos, Z., Kilpatrick, P., & Nikolopoulos, D. S. (2018). RADS: Real-time Anomaly Detection System for Cloud Data Centres. *arXiv preprint ArXiv:abs/1811.04481* .

[55] Telegram Bots developer documentation. `https://core.telegram.org/bots`. Last visit July 24, 2021.

[56] Openssl website. `https://www.openssl.org/`. Last visit July 24, 2021.

[57] RabbitMQ website. `https://www.rabbitmq.com/`. Last visit July 25, 2021.

[58] Perf tool wiki. `https://perf.wiki.kernel.org/index.php/Main\_Page`. Last visit July 25, 2021.

[59] Psutil tool wiki. `https://github.com/giampaolo/psutil`. Last visit July 25, 2021.

[60] VueJS Website. `https://vuejs.org/`. Last visit July 25, 2021.

[61] Grep Wikipedia Page. `https://en.wikipedia.org/wiki/Grep`. Last visit July 25, 2021.

[62] Malware Bazaar Website. `https://bazaar.abuse.ch/` Last visit July 29, 2021.

[63] Hardare Performance Counter Wiki page. `https://en.wikipedia.org/wiki/Hardware_performance_counter` Last visit July 29, 2021

[64] Cohen, F. (1986) Computer viruses. Ph.D. dissertation, Univ. Southern California, Los Angeles, CA, USA.

[65] Kasperky. `https://www.kaspersky.com/resource-center/threats/computer-viruses-vs-worms` Last visit July 31, 2021

[66] Cisco Ransomware Defense Validated Design Guide. `https://www.eschoolnews.com/files/2017/02/494454_Ransomware-Defense-Validated-Design-Guide.pdf` Last visit July 31, 2021

Abbreviations

HPC	Hardware Performance Counters
IoT	Internet of Things
OS	Operating System
HW	Hardware
ML	Machine Learning
C&C	Command & Control
TP	True Positive
FP	False Positive
TN	True Negative
FN	False Negative
TPR	True Positive Rate
FPR	False Positive Rate
TNR	True Negative Rate
FNR	False Negative Rate
FNR	False Negative Rate
Ev.	Evaluation
Ex.	Experiment
Pred.	Prediction
WS	Web Sockets
RF	Radio Frequency
SDN	Software-Defined Networking
DDoS	Distributed Denial of Service
MBR	Master Boot Record

Glossary

0–day 0–day malwares or vulnerabilities are novel malicious codes and exploits respectively, which were unknown before.

Hardware Performance Counters are special registers built-in in the CPU which store the count of hardware related events. [63]

List of Figures

List of Tables

Appendix A

Installation Guidelines

A.1 Ransomware Monitor

Prerequisites

1. Perf must be installed and present in the system. In the Raspberry used in this Thesis, the following commands were used to install it.

```
$ sudo apt update
$ sudo apt install linux-tools-4.9
```

2. A directory named ”config”
3. Inside the ”config” directory a *config.yaml* file must be present and correctly setup (as described in Chapter 6.4.3), and a ”policy” directory must also be present as well.
4. (optional) Policies can be added in the ”policies” directory
5. (optional) RabbitMQ installed (please see the Back-end installation guideline).

It is assumed that the *prerequisite* are applied at the root in both the source and binary version of the Ransomware Monitor.

A.1.1 Source

1. Python3 must be installed (https://www.python.org/downloads/)
2. Python3 Venv is used to facilitate the installation of dependencies

```
$ sudo apt install python3-venv
```

3. Create and activate the environment

```
$ python3 -m venv monitor_env
$ source monitor_env/bin/activate
```

4. Install dependencies

```
$ pip install -r requirements.txt
```

5. Run the script

```
$ python main.py
```

6. (Optional) Create a binary. The binary will be available inside the *dist* directory. The binary is tailored to the system used to create it and it is not cross-platform compatible.

```
$ pyinstaller --onefile main.py
```

A.1.2 Binary

It is sufficient to execute the binary.

```
$ ./main
```

A.2 Back-end

A.2.1 Prerequisites

1. NodeJS: Node v12.13.0 or higher must be installed in the system (https://nodejs.org/it/).
2. RabbitMQ: It is suggested to install RabbitMQ via Docker (https://hub.docker.com/_/rabbitmq). In this Thesis the **rabbitmq:3.8.2-management** image was used.
3. The *.env* file must be configured by modifiying the **RABBITMQ_URI** variable with the coordinates to the RabbitMQ instance. By default this variable assumes a local RabbitMQ installation (i.e. *amqp://guest:guest@localhost:5672*)
4. (optional) Modify the other variables in the *.env* file. Please read the README.md inside the backend directory for more information about the variables.

A.2.2 Running the backend

Inside the backend directory

```
$ yarn install
$ yarn start:dev
```

A.3 Front-end

A.3.1 Prerequisites

1. NodeJS: Node v12.13.0 or higher must be installed in the system (https://nodejs.org/it/).
2. Backend setup and running
3. The *.env* file must be configured by modifiying the **VUE_APP_API_ENDPOINT** and **VUE_APP_WS_ENDPOINT** variables. If the default values of the Back-end were not modified and it is hosted on the same machine, the default values of the *.env* file are already correct and are added here for reference.

   ```
   VUE_APP_API_ENDPOINT=http://localhost:3000/api/v1
   VUE_APP_WS_ENDPOINT=ws://localhost:81/websocket
   ```

A.3.2 Running the frontend

Inside the frontend directory.

```
$ yarn install
$ yarn serve
```

A.4 Ransomware 1: Ransomware PoC

The binary version is assumed. The source version provides a *README.md* file with further details.

The ransomware consists of a directory containing:

1. boot.asm
2. boot.bin
3. ransom.bin

A.4.1 Execution

Execute the ransomware with:

Please ensure not to run the ransomware on a production machine, as it will encrypt all the files.

```
$ ./ransom.bin
```

A.5 Helper Tool

1. Python3 must be installed (https://www.python.org/downloads/)
2. Python3 Venv is used to facilitate the installation of dependencies

```
$ sudo apt install python3-venv
```

3. Create and activate the environment

```
$ python3 -m venv helper_tool_env
$ source helper_tool_env/bin/activate
```

4. Install dependencies

```
$ pip install -r requirements.txt
```

5. Run the script

```
$ python helperTool.py
```

A.6 Ransomware 2: DarkRadiation

The ransomware consists of a directory containing the following files:

1. bash.sh
2. bot.sh
3. supermicro_cr.sh

A.6.1 Prerequisites

Edit all three files mentioned before by replacing the following placeholders with the correct values.

1. `<TELEGRAM_TOKEN>`
2. `<CHAT_ID>`

For a Telegram Bot creation procedure, please refer to the Telegram documentation (https://core.telegram.org/bots).

Lines 53 and 59 of *supermicro_cr.sh* also need to be modified to represent the path where the ransomware is present in the system.

A.6.2 Execution

Execute the ransomware with[1]:

Please ensure not to run the ransomware on a production machine, as it will encrypt all the files.

```
$ nohup ./supermicro_cr.sh testpassword & exit
```

[1]this command is extracted from the real ransomware execution in the wild

Appendix B

Contents of the CD

1. This Thesis in PDF version
2. This Thesis as LaTeXsource comprising the figures.
3. The mid-term presentation as a PowerPoint file.
4. A "Sources" directory containing the source code of the Ransom Monitor, the Back-end, Front-end and the Helper Tool. A compressed folder containing the source code of both ransowmare (password of the folder is: *infected*).
5. An "Experiment" directory containing
   (a) A "monitor-bin" directory with the binary of the Ransom Monitor. The included config directory is already setup with the configuration and policies mentioned in this Thesis; however the RabbitMQ configuration must be correctly defined as described in Appendix A.
   (b) A "Ransomware" compressed directory containing both ransomare in executable form[1]. The password of the compressed folder is: *infected*.
   (c) A *setup.sh* script to install perf on the Raspberry.
   (d) A *random_file_generator.sh* script to generate random file in the */home* directory.
6. A Data directory with the dataset generated during the evaluation.

[1] DarkRadiation being a bash script does not differ from the source version