University of Zurich[UZH]

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

Creation of a Dataset Modeling the Behavior of Malware Affecting the Confidentiality of Data Managed by IoT Devices

Fabio Sisi
Hünenberg, Switzerland
Student ID: 18-702-639

Supervisor: Dr. Alberto Huertas Celdran, Eder Scheid
Date of Submission: July 21, 2021

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

ifi

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

Abstract

Die ständig ansteigende Nachfrage nach IoT Geräten und Services führt zu einer wachsenden Zahl von IoT Plattformen und Konzepten, wie Crowdsensing. IoT Plattformen und Netwerke sind oft verwundbar gegenüber Cyber Attacken. Solche Attacken werden in vielen Fällen mit zwei der gefährlichsten Malware Familien ausgeführt, Spyware und Backdoors. Das Ziel diser Cyber Attacken ist oft, Zugriff auf Daten, die auf den IoT Geräten verwaltet werden, zu erlangen. Da die Malware Entwicklung ebenso voranschreitet, sind traditionelle Erkennungssysteme oft nicht in der Lage, Zero-Day Attacken zu erkennen. Der momentane Trend, um diese Herausforderung zu addressieren, besteht darin, dynamische Analysemethoden mit Machine Learning Modellen einzusetzen. Um diese Modelle effizient einzusetzen, müssen sie trainiert werden. Dazu braucht es Datensätze, welche normales und infiziertes Geräteverhalten enthalten. Existierende Datensätze haben aber diverse Schwächen. Viele davon enthalten keine Daten über Ressourcen-eingeschränkten Geräte oder sind zu alt, um den heutigen Malwarestandard darzustellen. Den Hauptbeitrag dieser Arbeit stellt demnach die Erstellung von FabIoT dar, ein Datensatz der das interne Verhalten eines Ressourcen-eingeschränkten Gerätes während verschiedener Backdoor Attacken modeliert. Zusätzlich ist auch das normale Geräteverhalten Teil des Datensatzes. Das in dieser Arbeit verwendete Test Gerät ist Teil einer echten IoT Crowdsensing Plattform, ElectroSense. Drei reale Backdoors mit verschiedenen Angriffsmöglichkeiten führen die Angriffe aus. Der Prozess um die Daten zu sammlen enthält das Design und die Entwicklung einer massgenscheiderten Überwachungssoftware um das Geräteverhalten aufzuzeichnen. Nach statistischer Analyse der gesammelten Daten ist ersichtlich, dass Unterschiede zwischen dem normalen und infizierten Verhalten festzustellen sind, was den Wert, den FabIoT bietet, bestätigt. Der Datensatz ist frei verfügbar für Forschungszwecke.

The increasing demand for IoT devices and services leads to an ever growing number of IoT concepts, platforms and networks, such as crowdsensing. IoT platforms are often vulnerable to cyber attacks due to their exposition to the internet and their resource-constrained capabilities. Cyber attacks can be conducted by different malware families, being spyware and backdoors two of the most dangerous ones. The goal of backdoors and spyware is in many cases the data being stored on and handled by IoT devices. Since backdoor development is evolving, traditional detection solutions are often not able to detect zero-day attacks. Nowadays, the trend to address this challenge is to apply behavioral fingerprinting analysis in combination with machine and deep learning models. In order to apply machine and deep learning models effectively, datasets containing clean and infected device behavior are required. However, there are open challenges regarding existing datasets. In many cases, these datasets do not concern themselves with resource-constrained devices. Also, many existing datasets are too old and do not focus on the

internal behavior. Thus, the present thesis' main contribution is the creation of FabIoT, a dataset modeling the internal behavior of a resource-constrained device while being infected by backdoors as well as the normal behavior. The test device is part of a real-world IoT crowdsensing platform, ElectroSense. Three real-world backdoors exhibiting a wide variety of attack behaviors were used to conduct the attacks. The data collection process involves the design and implementation of a custom monitoring software to examine and record the internal device behavior. After performing statistical analysis on the collected data, we were able to detect differences between the normal behavior and the behavior under attack and therefore, illustrate the value that FabIoT provides. Finally, we make the dataset freely available for research purposes.

Acknowledgments

First and foremost, I would like to thank my supervisors, Dr. Alberto Huertas Celdran and Eder Scheid. Their support, advice und guidance was absolutely invaluable. I would like to thank them for their patience and understanding over the past six months. Also, I would like to express my sincere gratitude to Prof. Dr. Burkhard Stiller and all members of the Communication Systems Group and the Department of Informatics at the University of Zurich for without them, this thesis would not have been possible. Last but not least, I want to thank my parents and my closest friends whose support and attention was unwavering.

Contents

Chapter 1

Introduction

1.1 Motivation

Over the last decade, there has been a steadily increasing demand for Internet of Things (IoT) devices and services. The combination of a large number of resource-constrained devices enables many scenarios such as smart cars and homes. Other possible large scale applications include smart cities or fully automated factories. One promising IoT concept that has gained in popularity is crowdsensing. Crowdsensing is a concept where a large amount of individuals contribute small amounts of data to a larger collection. The goal of such a platform could for example be the monitoring of the whole radio spectrum. ElectroSense, a real IoT platform, does exactly that and was created with the goal of monitoring the whole radio frequency spectrum using cheap and affordable hardware on a global scale.

As the application of IoT devices gains in popularity, there are also rising cyber security concerns [38]. Cyber attacks against IoT networks or platforms have been becoming increasingly common [60]. Among these attacks are not seldom spyware and backdoor attacks [3]. Data is in many cases the goal of cyber attacks against crowdsensing or IoT platforms in general. While the data handled by IoT devices should be protected and confidential, with every passing year, attackers launch more and more spyware and backdoor attacks [3]. One possible reason for the increasing number of attacks is that IoT devices can be used as a gateway to attack other possibly more critical devices in the same network [20].

Traditional malware detection systems often apply static analysis. That means that these systems rely on a previously known trace or signature in order to detect malware [15]. Attackers can exploit this fact and use evasion techniques such as packing or obfuscation to change its signature and therefore, make it harder to detect again [41]. Furthermore, that would imply that traditional systems are not capable of detecting new malware that has never been seen before, so called zero-day attacks [16, 51]. Opposed to static analysis, there is dynamic analysis, also called behavior fingerprinting[51]. The goal of dynamic analysis is to model the behavior of a device and to use that model in order to detect malware. The primary advantage of dynamic analysis is that while it is easy for attackers

to change the malware's signature, the behavior that the malware exhibits itself and causes on the device cannot be changed that easily. That also implies that it is possible to detect and mitigate zero-day attacks in certain circumstances [16].

Nowadays, solutions based on Machine and Deep Learning (ML/DL) are the most promising to apply dynamic analysis [51]. However, ML/DL models require datasets in order to be applied successfully and efficiently. These datasets model the normal behavior of devices, and/or the devices behavior when they are affected by cyber attacks. However, there are three main open challenges in terms of existing datasets. Firstly, the majority of existing datasets do not concern themselves with resource-constrained devices but rather with other types of architectures such as cloud data centres or client-server structures. Another issue is that many datasets are too old and do not accurately display the state of the art malware that is being used today. Also, newer concepts such as crowdsensing are often not considered in existing datasets. Lastly, most datasets fail to capture the devices' internal behavior and instead focus on network related activities.

1.2 Description of Work

To improve the previous challenges, the main goal of this thesis is to create FabIoT, a dataset modeling the behavior of a resource-constrained device while being infected by spyware and backdoors as well as the normal behavior. The test device, a Raspberry Pi, is part of ElectroSense, a real-world IoT crowdsensing platform. While under attack from three concrete implementations of backdoors, the internal behavior is being monitored. As a comparison, the device is monitored while being completely free of any malware to create a baseline. To choose three suitable backdoor implementations, an analysis regarding the recent trends in malware development was conducted. Also, the evaluation includes possible impacts a real-world attack would have on the actual IoT platform used in the data collection phase. The main contributions of this work are:

- An analysis of backdoors affecting IoT devices.
- The selection and execution of three of the most dangerous backdoors affecting IoT devices.
- The analysis of events monitoring the internal behavior of IoT devices such as the Raspberry Pi test device.
- The creation of a dataset modeling the behavior while being under attack from backdoors as well as the normal behavior and a script to monitor said behavior.

1.3 Thesis Outline

In the next chapter the reader will find an overview of previous work done related to the thesis. Both, existing datasets and attack detection solutions, are explored. Then, Chapter three outlines the data collection process and approach. It explains the monitoring

process and illustrates the architecture of the monitoring software as well as the whole data collection setup. Chapter four evaluates and discusses the collected results. The presented Bachelor Thesis ends with the conclusions and future work in Chapter five.

Chapter 2

Related Work

This chapter contains the existing works related to this thesis. It starts by presenting spyware and backdoors. Subsequently, there is an analysis of existing datasets modeling malware behavior. Finally, the section closes with an examination of the current state of malware detection solutions.

2.1 Backdoor and Spyware Background

Backdoors are an access point to a system for third party members without proper authentication. The general conception is that backdoors are mostly implemented with malicious purpose but that is not quite the truth. Many software companies include backdoors in their products to make troubleshooting and maintenance easier [23]. However, if the intent behind the implementation of a backdoor is malicious, there are certain traits that the majority of backdoors share. One of those traits is stealth [49]. Backdoors should remain undetected by the user in order to be useful. The user should remain completely unaware that his system is compromised. Additionally, backdoors do not attack the system directly but act more like a gateway for other malicious activity. It is not rare for backdoors to be combined with other types of malware. A typical example would be the combination of a trojan and a backdoor. As soon as the trojan is activated, e.g. executed by the victim, it installs a backdoor for the attacker to perhaps return at a later point in time. In general, it is rare that a piece of malware can be clearly categorized in a single family of malware. Often, malware implementations exhibit and combine attributes from many different malware types.

Spyware, also called info stealers, steal sensitive information from the victim and send that data to the attacker [47]. Among the most common stolen information are credit card details and passwords. Spyware is often found on personal computers or mobile phones. As it is often the case in the context of malware, the motivation behind the development and application of spyware is mostly of financial nature. Web hooks, screen recorders and keyloggers are popular implementations of this type of malware [28]. Due to its stealthy nature, spyware often conceals itself as part of a regular download or installation,

similar to adware. On the mobile platform, devices often get infected with spyware if any applications are downloaded from non-official app stores. Because of these facts, very few spyware implementations for IoT devices are freely available since most target personal computers.

2.2 Datasets Modeling Malware Behavior

This section analyzes datasets containing abnormal device behavior caused by some sort of attack. Most datasets contain normal as well as abnormal behavior and therefore, allow to identify and differentiate normal and abnormal behavior.

The literature on datasets modeling device behavior by collecting network related activity is considerable. LITNET-2020 [13] is the first dataset in this review. It contains normal network flows and while under 12 different attacks. The attacks include DDoS, port scanning and smurf attacks among others. In total, 85 different types of network flows were captured. The data was collected in an academic network setting on general computer systems over a time period of ten months. In [20], the authors collected data on IoT devices while conducting similar attacks. As an addition to the IoT devices, the authors also included regular PC systems in the data collection setup. The reason for that is that IoT datasets often ignore traffic between IoT devices and regular computer systems [20].

The authors of [2] created a dataset where they focused their efforts on creating an IoT dataset but also including Industrial Internet of Things (IIoT) devices. The dataset consists of telemetry data of IoT and IIoT devices as well as operating system logs and network data of the IoT network itself. The eight attack scenarios included DDoS, ransomware and backdoor attacks among others. Another dataset containing similar malware attacks is the IoT-23 dataset [17]. The authors captured 23 scenarios, 20 of which are from infected Raspberry Pis. In each scenario, a different type of malware attack was executed using different botnets. During the attack, the authors captured network data as raw network packet (pcap) files. For the additional three uninfected scenarios, real-world IoT devices were used. An important aspect of this dataset is the contrast between malicious and benign network traffic. Also working with botnets, in [29], the authors created a dataset where a typical home IoT environment was monitored while being under simulated and real attacks. The simulated attacks consisted of port scanning, spoofing and brute force attacks among others while the real attacks were conducted by the Mirai botnet. While the attacks took place, the authors captured pcap files. In total, 42 pcap files were collected.

In [22], Hamza et al. created the ACM SOSR 2019 dataset [61] using a rather novel approach. The detailed approach is explained in section 2.3. The dataset contains network flow counters while the devices were functioning normally and while under attack. The attacks consisted of network attacks such as DoS, flooding and ARP spoofing. The authors of [45] collected traces from simulated IoT devices and created the DS2OS dataset. Among the simulated IoT devices were movement sensors, thermostats, smartphones, etc. The focus in this work and dataset lies on the micro services the IoT devices offer. N-BaIoT was created in a very similar manner [40]. The network traffic of nine commercial IoT devices

was collected while being under attack from two different botnets, Mirai and BASHLITE, similar to [17, 29]. Each measurement contains 11 labels, ten of which were collected while being under attack from the aforementioned botnets and one label modeling benign behavior.

In 2018, the CIC (Canadian Institute fore Cyber Security) published [55] and with it two datasets, the CIC-IDS 2017 [55] and the CIC-IDS 2018 [54]. Both datasets contain raw network traffic captures collected during cyber attacks as well as benign samples. The seven attack scenarios include Brute Force, Heartbleed, Botnet, DoS, DDoS, Web attacks, and Infiltration attacks. The difference between the two datasets is that in the 2017 version the network traffic was collected from 25 users whereas in the 2018 version the data was collected from 500 devices in total.

As more and more people use the Android operating system, there are also more and more devices being infected by malware. To combat this growing threat, there are also Android based datasets to detect malware such as the Android Malware Dataset (CIC-AndMal2017) [35]. The dataset was produced in [36], where the authors monitored 80 different network traffic features of Android devices. The data was collected on real smart phones with 426 malicious and 5'065 benign applications being installed. In 2019, the second part of this dataset, CIC-InvesAndMal2019 [33], was published in [62] and contains additional data such as API calls and system log files. The malware samples were categorized in either adware, ransomware, scareware or SMS malware. Also in the context of Android malware, there is the Android Adware and General Malware Dataset (CIC-AAGM 2017) [32]. That dataset was produced as a result of [34]. Many of the authors were also involved with [33], since both datasets were published by the CIC. The difference between the two datasets is that CIC-AAGM 2017 puts its focus on adware and general malware and it only contains network traffic captures. It contains raw and processed network captures of 1900 applications, 400 of which are considered malware. The malicious applications were installed on real smart phones like in [33].

NGIDS-DS [21] is another relevant dataset that contains network traffic captures and system logs of enterprise-critical infrastructure, such as storage, web or email servers. The authors collected data in a normal scenario and under seven common network attacks, Exploits, DoS, Worms, Generic, Reconnaissance, Shellcode, and backdoors. As a test platform, the IXIA Perfect Storm tool, a commercial-scale security test hardware platform, was used.

Finally, the remaining three datasets contain system calls data. The two ADFA (Australia Defense Force Academy) datasets are similar in many regards, they both contain system calls of a system in a normal state and while under attack from various attacks. The attacks include brute force attacks, webshells, poisoned executable and exploits among others. The main difference is that ADFA-LD [11] is based on Linux systems and ADFA-WD [10] on Windows systems. The University of New Mexico (UNM) dataset [9] was created in 1999 and contains roughly 500 kB of system call data. In comparison, ADFA-WD contains 13.6 GB of system call data. As do the ADFA datasets, the UNM dataset also contains normal behavior and attack behavior. Among the attacks were at the time modern attack sets such as trojans and buffer overflows [9]. Due to the technological

Table 2.1: The reviewed datasets modeling malware behavior (datasets are grouped by behavior source, using double horizontal lines to seperate them, and sorted by year)

Name	Year	Device Type	Behavior Source	Behavior Data	Tested Malware
LITNET-2020 [13]	2020	General purpose computers	Network	Processed network flows	Network attacks
IoT-Keeper [20]	2020	IoT	Network	Raw network flows	Network attacks
TON_IoT [2]	2020	IoT	Network / System Logs	Telemetry data, OS logs & network traffic	9 different attack scenarios
IoT-23 [17]	2020	IoT	Network	Raw network capture	Botnets
IoT network intrusion dataset [29]	2019	IoT	Network	Raw network capture	Botnets
ACM SOSR 2019 [61]	2019	IoT	Network	Network flow counters	Network attacks
CIC-InvesAndMal2019 [33]	2019	Android	Network / System Logs	Network traffic features, API calls & system logs	Adware, Ransomware, Scareware & SMS Malware
N-BaIoT [40]	2018	IoT	Network	Processed network flows	Botnets
DS2OS [45]	2018	IoT	Network	Application layer traces	Network attacks
CIC-IDS 2018 [54]	2018	General purpose computers	Network	Raw and processed network capture	Network attacks
CIC-IDS 2017 [55]	2018	General purpose computers	Network	Raw and processed network capture	Network attacks
CIC-AAGM [32]	2017	Android	Network	Raw and processed network capture	Adware & general malware
NGIDS-DS [21]	2017	Critical Cyber Infrastructure	Network / System Logs	Raw network capture / OS logs	Network attacks
ADFA-WD [10]	2014	General purpose computers	System Calls	DLL traces	60 different attack sets
ADFA-LD [11]	2013	General purpose computers	System Calls	System Logs	60 different attack sets
UNM Dataset [9]	1999	General purpose computers	System Calls	System call identifiers	Trojans, buffer overflows & symbolic link attacks

advancements made during the last 22 years and its substantial age, the UNM dataset is now considered outdated.

Table 2.1 gives an overview of public datasets that try to model device behavior while being under attack. It can be observed that the great majority of datasets are focused on network related metrics. It is a common occurrence for datasets to model botnets such as Mirai or BASHLITE since they are among the most prominent and dangerous IoT malware. Also, as stated in [51], other metrics such as system calls or resource usage are

under-used. The network based datasets fail to capture the internal behavior of a device and only focus on communication. However, information about the internal behavior would be immensely useful to understand how malware works and what it causes inside of the infected device.

2.3 Malware Detection Solutions

The internal behavior of devices is a topic that has attracted more and more interest from the research community in recent years. Device fingerprinting has the goal to describe and model the internal behavior of a device. In [51] the authors summarize the recent research in device fingerprinting. They came to the conclusion that the majority of approaches use network related activity to model the behavior of a device in terms of anomaly/attack detection. However, other promising dimensions such as system calls, the usage of resources, and HPCs were also researched and show potential. Additionally, most solutions only consider relatively sophisticated attacks. While highly sophisticated attacks do happen, they are rare and are in many cases extremely situationally specific and customized for the target [10]. Thus, they are not considered in many solutions and those solutions may not provide protection from highly sophisticated attacks.

The section starts by looking at works about detection solutions using network related activity. The authors of [57] proposed an approach where a combination of network activities and system calls was used in order to detect malware. Using this combination of behavior dimensions, the authors were able to achieve a detection rate of 99.54% utilizing ML algorithms. In [1], a solution using blockchain and ML in the context of IoT devices was developed. A ML algorithm is used to monitor and analyze the behavior of all devices in the network. The N-BaIoT dataset [40] was used in this work. Another solution considering botnets was presented in [19]. The idea was to detect an attack based on a trust score calculated for all traffic flows in an IoT network. The experiment was conducted in a software-defined networking (SDN) environment. Also in the context of SDN, the authors of [8] presented a system that monitors network traffic. Due to its nature, SDN is arguably more susceptible to DDoS or Man in the Middle attacks [31, 37]. The multi-stage system managed to detect and diminish DDoS and port scan attacks.

Hamza et al. proposed a system in [22] where the goal is to identify abnormal behavior in IoT devices using the manufacturer usage description (MUD). The MUD essentially defines the normal behavior of a device and is provided by the devices' manufacturer. ML algorithms then analyze the collected data compared to the specifications of the MUD and try to detect any attacks. The considered attacks consist of spoofing, flooding and DDoS attacks. With a true positive rate of 89.7% at its highest, the authors mention that this might not be enough for a large scale deployment on real IoT networks. This work produced the ACM SOSR 2019 dataset [61].

In [65], the authors tasked themselves with the creation of an anomaly detection system in the context of enterprise IoT devices. A key point in their contribution is the fact that the learning data for the ML model may be polluted since the system is located in a real-world environment. The result of their research was RADAR, a ML-based detection

system capable of detecting attacks on IoT networks using network traces. The system proved to be robust with an F-Score > 0.9 with up to 15% polluted data [65].

System calls are requests or commands which are sent from a process or a program to the operating system. Such calls are in most cases regarding access to the memory or file system [7]. The authors of [24] used an interesting and rather novel approach in using blockchain technology. The system takes a snapshot of the target software and compares it with a trusted and verified snapshot which is acquired from the blockchain. While acquiring the trusted snapshot, the monitoring system also generates a white list of files which the software is allowed to access. The second control mechanism is the monitoring of file system calls made by the target software and comparing them to its whitelist.

Another system was created by Mishra et. al called VMGuard [42]. In the context of cloud computing, the authors used Virtual Machine Introspection (VMI) to monitor system call traces made by programs. Using the bag-of-n-grams technique, features of normal and attack traces are extracted. Subsequently, a ML classifier, which has been trained on an existing data set [9], is then used to detect any malicious behavior. During their experiments, the authors achieved a detection rate of 94%, up to 100%.

Javaheri et al. proposed a specific framework to detect and neutralize three different types of spyware on Windows [28]. Their approach was to use statistical analysis techniques on kernel-level system calls to classify three types of spyware, keyloggers, screen recorders and blockers. Their results show that their method had a high accuracy of detecting malware with a 93% detection rate. On the same platform, Canzanese et al. developed another system that uses system call analysis to detect malware [7]. After collecting all the system calls, a bag-of-n-grams model was used to extract certain features. The authors showed that they were able to detect unknown malware, similar to a zero-day attack.

In [52] the authors provided an interesting approach where they created MADAM (Multi-Level Anomaly Detector for Android Malware). A system which analyzes the behavior of Android devices on multiple levels. On the kernel level, system calls were used to see whether the application in question causes any unexpected or irregular increase of activity [52]. Similar to [24], file system calls were used in particular. Using this approach, the authors managed to detect 96% of malicious applications from a dataset containing botnets, ransomware, rootkits, and trojans among others. In a very similar manner, [39] also uses a multi-level approach, using static and dynamic analysis on the device, user and application behavior. The result is BRIDEMAID (Behavior-based Rapid Identifier Detector and Eliminator of Malware for AndroID) [39]. In general, this paper uses a very similar approach as [52] and is also written by the same authors. BRIDEMAID seems like a more lightweight solution.

Another technique of detecting malware analyzes the usage of resources. Bridges et al. proposed a system using CPU power profiles to detect malware, namely rootkits [6]. The authors defined a series of tasks to be executed on the machine while continuously monitoring the machines power consumption. Using ML and statistical analysis, the authors were able to differentiate between infected and clean systems.

Barbhuiya et al. proposed the Real-time Anomaly Detection System (RADS) for Cloud Data Centres [4]. RADS monitors and learns the behavior of a virtual machine in a cloud

network by analyzing the CPU usage pattern and network traffic usage. In the context of cloud based computing, there are additional challenges the authors faced while creating the system. One such challenge would be that in cloud computing systems there may be actual workload spikes (e.g. during extremely busy hours) which result in higher CPU usage and could potentially lead to false positives. Cloud data centers are especially vulnerable to DDoS and cryptomining attacks [4] which is why RADS is focusing on these attacks. Also in the context of cloud systems, another interesting approach was taken by the authors of [48]. They proposed a system to detect anomalies by looking at the resource behavior. The motivation behind the idea is that many cloud systems are autoscaling. That means that any additionally required resources are provided by the system. The problem is that these additional resources can be requested by a workload created by malware. The behavior model consisted of CPU usage metrics which was used to train an AutoRegressive (AR) model. The model was successfully tested by trying to detect Denial-of-Service (DoS) and stress attacks.

HPCs are special counters built into computer processors with the task of tracking and counting hardware-related events [53]. Such events can be clock cycles, cache hits/misses, branch behavior, memory access patterns etc. [56]. This can slightly vary depending on the processor model. Even though HPCs give a deep insight into system performance, only a limited number of them can be tracked simultaneously [5]. The authors of [53] proposed a system to detect malware using HPCs that consists of two parts and is called 2SMaRT. In a first step, 2SMaRT tries to predict if an application is either benignware or part of one of the four defined malware classes, virus, rootkit, backdoor, or trojan. Then, depending on the malware class, a different ML model is deployed to ensure a high detection rate. By using that two-stage model, the authors were able to achieve an F-score of 92% on average.

Similar to [24], the authors of [18] also proposed a detection system using blockchain technology, named CIoTA. Each device of the network trains their detection model locally. Those locally trained models are then merged and validated by the other devices. Opposed to [24], the CIoTA systems uses HPCs instead of system calls. The control flow is monitored by creating an Extended Markov Model (EMM) using HPCs. In a testing environment of 48 Raspberry Pis, CIoTA was able to detect malicious activities successfully.

Interestingly, [66] raised the question whether malicious behavior actually has an impact on HPC values and therefore, questioning HPC-based malware detection in general. The authors argue that previous work concerning themselves with HPC-based malware detection presume unrealistic circumstances and assumptions [66]. Mainly the correlation of high-level software behavior, benignware versus malware, and low-level events such as HPCs is questioned. In their experiments, the authors were able to achieve an F1-score of 80.78% at best. Additionally, the authors conducted another experiment where they would induce ransomware into beningware and then see whether ML algorithms could detect said malware using HPCs. No ML model was able to detect the ransomware successfully. Also working on a Windows system, Singh et al. proposed another solution using HPCs in [56]. In their research, the authors identified 16 promising HPCs to detect kernel rootkits specifically using ML algorithms. On a Windows based testing platform, the system achieved a high detection rate with 99%.

Table 2.2: The reviewed attack detection solutions (works are grouped by behavior source, using double horizontal lines to seperate them, and sorted by year)

Work	Year	Device Type	Behavior Source	Approach	Tested Malware	Performance
[57]	2020	Windows PC	Network / System calls	ML	-	99.54% Detection rate
[1]	2020	IoT Devices	Network	ML / Blockchain	Botnets [40]	99.2% True positive rate
[19]	2019	IoT Devices / SDN	Network	ML	DDoS / Botnets	-
[22]	2019	IoT Devices	Network	ML	Network attacks [61]	89.7% True positive rate
[65]	2019	IoT Devices	Network	ML	-	F-Score > 0.9
[8]	2018	SDN	Network	ML	DDoS and portscan attacks	Attacks detected and diminished
[24]	2020	IoT Devices	Software Snapshot / System Calls	Blockchain	Intrusions	Attacks detected
[42]	2020	Cloud Systems	System Calls	ML	-	94 - 100% Detection rate
[28]	2018	Windows PC	System Calls	ML	Spyware / Ransomware	93% Detection rate
[52]	2018	Android	System Calls	ML	125 Malware families	96% Detection rate
[39]	2017	Android	System Calls	ML	123 Malware families	99.7% Detection rate
[7]	2015	Windows PC	System Calls	ML / Statistical Analysis	Trojans, viruses and sub-categories	92% True positive rate
[6]	2018	Windows PC	Resources Usage	ML / Statistical Analysis	Rootkits	93 - 100% Detection rate
[4]	2018	Cloud Data Centres	Resource Usage / Network	ML / Time series	DDoS / Crypotminers	90-95% F1-score
[48]	2018	Cloud Systems	Resources Usage	Statistical Analysis	DoS / Stress Attacks	All attacks detected
[53]	2019	Linux PC	HPC	ML	Backdoor, Rootkit, Trojan, Virus	92% F-score on average
[18]	2018	IoT Devices	HPC	Blockchain	Exploits	Attacks detected
[66]	2018	Windows PC	HPC	ML	35 different malware families	80.78% F1-score at best
[56]	2017	Windows PC	HPC	ML	Rootkits	99% Detection rate

As can be observed, there is a similar distribution of behavior sources as it is the case in the previous section regarding datasets also shown in Table 2.2. Network is the most used source but other metrics such as system calls or HPCs are also promising. Additionally, the majority of the reviewed papers used a ML-based approach. It is also worthy to note that many solutions achieved quite a high detection rate for their respective attack type. However, technological advancements and advancements in malware development will further increase the need for newer solutions.

Chapter 3

Creation of the FabIoT dataset

3.1 Scenario: ElectroSense

To capture and make the radio spectrum data available efficiently, the ElectroSense platform consists of three main components: the sensors, the backend, and the web services (see Figure 3.2). The ElectroSense sensors include a single circuit board computer, a Raspberry Pi for example, a radio frontend, and an antenna as can be seen in Figure 3.1. The sensor used in this work is a Raspberry Pi 4 having an ARMv8 processor with 4 cores and 4GB of RAM. The antenna collects radio signals which are processed by the RTL-SDR Silver V3 radio frontend and sent to the server by the Raspberry Pi. The radio frontend monitors the frequency range between 20MHz and 1.8GHz. The Raspberry Pi needs to be connected to the internet using either an Ethernet cable or a WiFi dongle. In this thesis, an Ethernet cable was used. Since the project is an open initiative [44], the software running on the Raspberry Pi is freely available on GitHub [14].

The ElectroSense backend handles all the data sent by all sensors. Furthermore, it offers an API which makes it possible to receive raw or aggregated data on every single sensor via HTTP request. However, certain information like the exact location of a sensor is not available for privacy reasons. The API uses basic authentication and an ElectroSense account with a username and a password is required to access the API.

Lastly, the web services offer direct access to the sensors on the ElectroSense website. Each sensor can be accessed and its live data can be observed through these services. It is also visible where the sensors are located on a map, whether it is inside or outside and if any other users are currently accessing the sensor. Privacy settings, such as making the exact location of the sensor accessible to other users, can be changed during the setup process of the sensor. Only one user can access one sensor at a time. While accessing a sensor, the user then has the choice between different types of data to view.

Figure 3.1: ElectroSense sensor parts

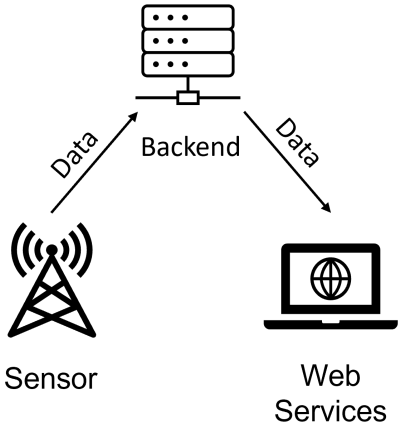

Figure 3.2: ElectroSense architecture during the creation of FabIoT

3.2 Backdoors affecting ElectroSense

This section illustrates the three chosen backdoor implementations. For each implementation, the focus lies on its features, deployment and execution. The section closes with a comparison regarding the most important aspects. In a cyber attack setting, the ElectroSense Raspberry Pi is considered the client whereas the system which sends the attacking commands is referenced as the server or attacker side.

3.2.1 httpBackdoor

Features

”httpBackdoor” is arguably the purest form of backdoor of the selected backdoors. It creates a web server on the Raspberry Pi to which the attacker can send HTTP requests, hence the name ”httpBackdoor” [59]. The general architecture of the setup in this work is depicted in Figure 3.3. This very simple backdoor has two basic functionalities. Firstly, the attacker can extract basic system information such as OS version and name or saved SSH keys by sending a GET request. Secondly, by sending a POST request, the attacker can execute command line commands on the Raspberry Pi as indicated in Figure 3.3.

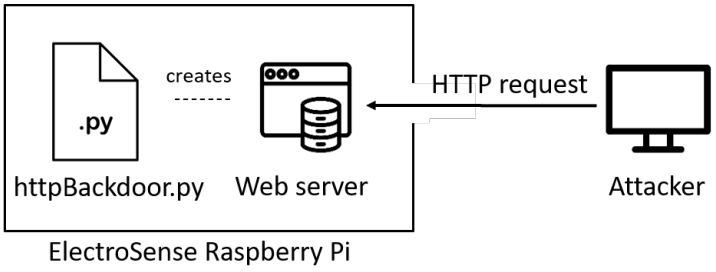

Figure 3.3: ”httpBackdoor” architecture

Deployment

The deployment of ”httpBackdoor” is rather simple. On the Raspberry Pi, also called the client, a Python file needs to be executed. It requires Python to be installed on the Raspberry Pi. However, there is an alternative option which does not require Python being installed. It is possible to create a binary using pyinstaller [63] on another machine. Instead of running the Python file on the target machine, which in this setting is the Raspberry Pi, the created binary needs to be executed. Additionally, the binary can be encrypted and thus, making it more difficult to decompile. On the server side, there is no installation required which is very convenient.

Execution

The attacker needs to include the IP address and port that the Raspberry Pi listens on in the code. To execute the attack the attacker sends HTTP requests to the specified IP address and port while the Python file or the binary is running on the Raspberry Pi acting as an ElectroSense sensor.

The following commands are examples of how to make use of both functionalities offered by the malware using the software curl [12]. Generally, every tool that is able to send HTTP requests can be used, other examples include Insomnia [25] or Postman [26].

- Basic system information extraction

  ```
  curl --location --head IP:PORT -H "Content-Type: text"
  ```

  The first example is used to extract the basic system information. In the malware code, this functionality is hard coded and can potentially lead to errors if certain information the backdoor is looking for is not present on the Raspberry Pi. Since this is the case for the test device, the functionality was not working in its original state and had to be altered in the code for the purpose of the thesis.

- Command execution

  ```
  curl --location -X POST IP:PORT -H "pwd:password" --data-raw "ls" -i
  ```

  The second example is used to send the ”ls” command to the Raspberry Pi in order to receive information about the current directory. Generally, the format ”data-raw” is used. The attentive reader also notices that another header with the content ”pwd:password” is also sent as part of the request. ”httpBackdoor” offers the possibility for the attacker to set a password, in this case ”password”, to ensure that only commands sent by the attacker are executed. Lastly, the additional option ”-i” is included so that the response headers which contain the information the attacker is after are displayed.

3.2.2 backdoor

Features

"backdoor" by "jakoritarleite" consists of two parts, the client and the server side [30]. The server side sends commands to a specified IP address and port on which the malware on the Raspberry Pi listens, as illustrated in Figure 3.4. In other words, the malware on the ElectroSense sensor listens for commmands from the attacker. The most notable feature is that the backdoor can enable the attacker to open a shell on the Raspberry Pi and therefore give the attacker complete access. Additionally, the backdoor offers the functionality to pull the contents of a file on the Raspberry Pi onto the attackers' machine.

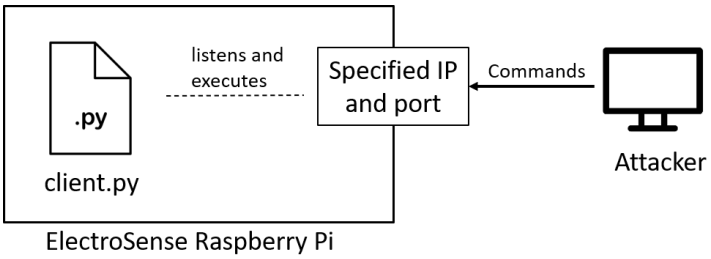

Figure 3.4: "backdoor" architecture

Deployment

The deployment is very similar to "httpBackdoor" since the client-side malware is written in Python. Generating a binary and then deploying the malware is not mentioned specifically in the instructions but should also work. Depending on the target platform, generating a binary can be easier and prove as more effective.

Execution

The specified IP address and port need to be included in both, the server and client Python file. The connection is then established automatically, if everything has been set up correctly. If the connection has been established successfully, the attacker has the opportunity to execute the following commands on the server side.

- shell

  Allows the attacker to open a shell on the Raspberry Pi.

- recv archive

  Pulls a file from the Raspberry Pi to the attacker machine.

- help

  Outputs information about the usage of the backdoor.

- exit

  Terminates the connection and ends the process on the Raspberry Pi and the attacker machine.

3.2.3 thetick

Features

”thetick” is the most complex of the selected backdoors. Similar to ”backdoor”, ”thetick” resembles a Command & Control (C&C) server structure [43] with the difference that ”thetick” can have multiple clients connected at the same time and switch between them seamlessly. In other words, several ElectroSense sensors could potentially be infected and act as a clients simultaneously. The attacker can send commands to each client individually, as shown in Fig. 3.5. In the context of ”thetick”, the terms, client and bot, are to be understood synonymously.

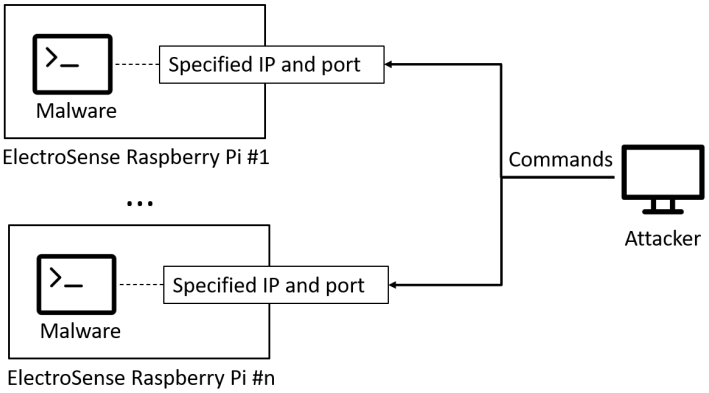

Figure 3.5: ”thetick” architecture

Deployment

The advanced applicability comes at a price, however. The building and deploying process is more complicated and requires more dependencies compared to the other two backdoors. On the client side, a Linux executable needs to be built. Depending on the hardware, the process could possibly take some time. To avoid that, the executable can be built beforehand on similar hardware and then shipped to the client. On the test Raspberry Pi, the build was negligible, however.

Execution

As soon as the executable has been built, it can be executed with the designated IP address and port passed as arguments. On the server side, a Python file is executed and the connection should be established automatically. If any other clients are started with same designated IP address and port passed as arguments, they also connect automatically which causes a notification on the server interface. With 18 different commands that can be issued in the C&C server, ”thetick” provides extensive possibilities for an attacker to use which go beyond the traditional definition of a backdoor. The commands have been grouped into either administrative commands or executable commands.

Administrative commands are used to organize different settings on the server side.

- bots

  Returns a list of all connected clients at this point of time with the status either ”live”, ”busy” or ”gone”. The status ”live” means that the client is available and ready while ”busy” means that the client is currently executing a task. Finally, ”gone” means that the client has disconnected. There can be any number of reason for that. Additionally, the IP address, number and UUID for each client are displayed. The UUID is a unique ID created by the server after the client has connected to it. The number signifies in what order the clients connected and the IP address is where the client connected from.

- clear

  Clears the screen. The usage is similar to the ”cls” command on a Windows shell or ”clear” on a Linux shell.

- current

  Shows the currently selected client. Displays the same information as in the ”bots” command, just for the currently selected client.

- exit

  Exits the program and terminates all connections to clients.

- use <IP address> <number> <UUID>

  With this command, the attacker can select a new client using one of the three options to identify the bot. Only one of the three arguments is required since every one of them uniquely identifies a client. If the command is executed without any arguments, the currently selected client is deselected.

Executable commands are used to do interact with the client.

- chmod <file> <access mode>

  Alters the access mode flag of a file on the Raspberry Pi.

- dig <domain name>

  Resolvs a domain name at the client.

- download <url>

  With this command, the attacker can download a file onto the Raspberry Pi via HTTP. It can potentially be used to download any further malicious software.

- exec

  Executes a single non-interactive command (e.g. ”ls” or ”pwd”) and returns the output of the command. It is possible to chain commands together using a semicolon, e.g. ”cd;ls”.

- fork

  Creates a new instance of a client connection with a new UUID.

- help * <command>

  Shows additional information in general or per command.

- kill

  Terminates the connection to the currently selected client and sets its status to ”gone”.

- pivot <listen on port> <connect to IP address> <connect to port>

  Used to create a one-shot TCP tunnel. According to the authors, this is rather useful to launch exploits since the tunnel is only available to localhost and is closed once a client has connected [43].

-
  - proxy [ls]
    Lists all active proxies at the moment.
  - proxy [add] <port> [bind address] [username] [password]
    Adds a new proxy.
  - proxy [rm] <port>
    Removes an active proxy.

- pull <remote file> <local file>

  Pulls the content of a remote file (located on the Raspberry Pi) to a local file.

- push <local file> <remote file>

  Does the exact opposite of ”pull”. The contents of a local file can be pushed into a remote file located on the Raspberry Pi.

- rm <remote file>

  Used to delete a file on the client.

- shell

  This command is used to open an interactive shell of the Raspberry Pi and give the attacker full control.

3.2.4 Comparison

Each implementation has its advantages and disadvantages as illustrated in table 3.1. ”httpBackdoor” is very lightweight and its code can easily be altered and customized. That means that the limited functionality can be extended in any way one deems fit or necessary. Basic knowledge of Python is required, however. The term ”limited” is relative to the other backdoors in this comparison. While not being able to open a shell, ”httpBackdoor” still enables the attacker to issue commands and therefore, have control of the Raspberry Pi. The architecture of ”backdoor” is also rather simple with the additional requirement of needing Python on the server side in order to establish a connection between client and server. The added necessity rewards the attacker with the

Table 3.1: Backdoor Comparison

	httpBackdoor	**backdoor**	**thetick**
Functionality	Limited	Simple but powerful	Various
Deployment	No installation on server side	Written in pure Python	Executable on client, Python on server
Architecture	Lightweight	Few dependencies	Complex

ability to pull files from the Raspberry Pi, however. It uses very few dependencies which makes the installation rather straightforward. Again, basic Python knowledge is required as well. Lastly, "thetick" offers a wide variety of available commands and functionalities which go beyond a regular backdoor. Not only does it provide access to a system, but it also offers options on what to do with that kind of access. It exhibits aspects of a botnet which makes it difficult to categorize it into a single malware family. Nevertheless, these similarities are still limited. While it is possible to connect multiple clients to a C&C server and switch between them, it is not possible to issue a simultaneous command to all or multiple clients. There is also one thing that all of the implementations have in common. While they are classified as backdoors, all of them can also act as spyware in their own way. Since all of them can access the command line in some form, they can provide information about the Raspberry Pi to the attacker which further shows the problem with categorizing malware strictly into one category.

3.3 Malware Attack Behavior

To fully capture and compare all three backdoors, this work defines six different attack behaviors. Naturally, because of their differences in complexity, not all backdoors are able to exhibit all behaviors, only behavior1-3 can be exhibited by all three implementations. The behaviors themselves also differ in complexity and therefore, cause different processes on the Raspberry Pi. Each implementation achieves the result of each behavior a bit differently. For example, "httpBackdoor" needs to send single commands to achieve behavior3 whereas "backdoor" and "thetick" open up a shell on the Raspberry Pi. During the monitoring process, the attacks are repeated every few seconds throughout the specified monitoring time period using either batch or UI Path [64] automation scripts. The behaviors are defined as follows:

The goal of **behavior1** is to retrieve some basic system information. To do that simple non-interactive commands susch as "pwd" are executed through the respective backdoor. It returns information about where the backdoor execution file is located on the Raspberry Pi. In **behavior2**, a more complex chain of commands is executed. First a new directory is created and subsequently entered. Next, a repository is cloned into the new directory. In a real-world setting, that could be used to bring further malware onto the device. **behavior3** focuses on deleting files on the target system. To make sure that there are enough files to be deleted, there were roughly 2000 files created before starting the monitoring process. After completing the measurements, the remaining files were deleted before capturing the next behavior. **behavior4** aims to pull a file from the Raspberry Pi to the attacker. In a real-world scenario, that could potentially be used to steal confidential or critical

Table 3.2: Attack Behaviors

Name	Description
behavior1	Execute a simple non-interactive command ("pwd") and returning its ouput to the attacker
behavior2	Execute a more complex chain of commands (creating a new directory and cloning a repository)
behavior3	Deleting a file.
behavior4	Pulling a file from the Raspberry Pi.
behavior5	Resolving a domain name at the Raspberry Pi.
behavior6	Changing an access flag for a file on the Raspberry Pi.

information. In **behavior5**, the goal is to resolve a domain name at the Raspberry Pi. It is especially useful to resolve local domains at at the target network. Finally, in **behavior6**, the malware should change access flags for files on the Raspberry Pi. Similar to behavior3, a large number of files was prepared so that the malware would not run out of files to change. Table 3.2 gives a general overview of the different behaviors.

3.4 Behavioral Monitoring Process

3.4.1 Events

As pointed out in the related work chapter, system calls, HPCs, and resource usage were among the most promising parameters to model the internal behavior of a resource constrained device. This work focuses on HPCs and the usage of resources. Unfortunately, the literature fails to mention concrete HPCs for Raspberry Pis. In [53], the authors mention concrete HPCs used to detect backdoors on Linux systems but said HPCs are not supported on the Raspberry Pi test device. Therefore, this work monitors a large amount of different available performance events. The monitoring script collects the amount of occurrences of 72 different performance events during a specified time period. Table 3.3 gives on overview of the different events and what information they contain.

The authors of [66] argue that detection of malware using HPCs is not possible. However, in their arguments they assumed that an algorithm is not capable of distinguishing between benevolent and malevolent behavior by using HPCs. For example, encrypting a file can be interpreted as a hostile action by ransomware or as a security measure taken by the user. The mentioned problem is the difficulty to link high level malware behavior to low level information such as HPCs [66]. They argue that it is difficult to capture the whole behavior of a device which has such wide variety of possible and potentially very complex behaviors. A similar observation was made by the authors of [19] where they compared the network behavior of a MacBook Pro and a Philips Hue light bulb. Both are part of an IoT network but their possible behavior is vastly different. To justify the chosen approach and collecting HPCs, we argue that in our setup the test device operates inside known parameters and every deviation from the normal behavior is considered an anomaly and possibly caused by malware.

Table 3.3: Monitoring Events

Events	Information
cpu_usage_pct, ram_usage_pct, network_in, network_out	Usage of Resources
block_bio_backmerge, block_bio_remap, block_dirty_buffer, block_getrq, block_touch_buffer, block_unplug	I/O Block Devices
clk_set_rate	Clock Framework
cpu-migrations	CPU Migrations
cs	Context Switches
fib_table_lookup	Packet Forwarding
mm_filemap_add_to_page_cache	File System
gpio_value	GPIO Signals
ipi_raise	Inter-Processor Interrupts
irq_enable, 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
mm_lru_insertion	Kernel Interfaces of Page Tables
qdisc_dequeue	Queuing Disciplines
get_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
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

3.4.2 Monitoring

Architecture

Although the monitoring script is straightforward, there was still a need to organize it in different code blocks. These blocks are described as follows:

- Setup

  In this step, any given argument is assigned to a variable. If no argument was given for a variable, the variable is assigned the default value. Additionally, the respective file is created if does not exist yet. If the file does not exist however, any new monitored data would simply be appended to the existing file. The performance events that are to be captured are also defined in a string in this section.

- Output Formatting

  These lines are responsible for the headers of the data file. The *perf* command needs to be executed because *perf* does not necessarily output the events in the order that were specified earlier in the code. Therefore, the order is extracted from the first execution.

- Monitoring Loop

  The monitoring loop consists of two parts, data collection and output. It runs until the desired amount of samples has been reached. Each loop, the data collection process is executed.

- Data Collection

  The main aspect of the monitoring process is being conducted in this section. All the specified data is collected using different commands. The output of the *perf* command is being extracted from a temporary file.

- Output

  Lastly, the collected data needs to be added to the final output file as a new sample. If the number of samples has reached its goal, the script terminates at this point.

Execution

For the monitoring script, a bash shell script is used because of its ease of use. The script is specifically designed to be executed on a Raspberry Pi but is compatible with any machine running a Linux OS. There are two prerequisites that need to be installed, however. To capture the performance events the correct kernel-compatible version of the *perf* tool [46] needs to be installed as well as the *ifstat* tool [50] for collecting network measurements. The output will be a csv file. The script is executed by issuing the following command:

```
$ bash script.sh [-a] [-f] [-t]
```

In order to make the script versatile and perhaps even useful beyond the thesis, the script has been designed with certain parameters being customizable. That way, the user can easily decide for how long the script should run and how many samples should be taken. There are several arguments that can be passed to customize the output.

- **[-a]**

  Defines the amount of samples that are to be taken. The default amount is 10 samples.

- **[-f]**

  Defines the name of the data file. The default value is ”data”.

- **[-t]**

  Defines the time between measurements. The default value is 10. Note that this argument is in seconds.

The list of monitored performance events is also customizable. However, it needs to be altered directly in the code of the script. In order to add another event that should be monitored, the event in question just needs to be added to the list, duplicate events are possible but could potentially induce performance issues. The same process can be applied for removing a certain event. If the list is empty, an error occurs and the script terminates.

Another argument that has been implemented is the [-h] argument. It is common for most command line tools to have such a feature to help the user and give an overview of the capabilities of the tool.

```
bash script.sh [-h]
```

This command displays the aforementioned information about usage and the available arguments. If the [-h] argument is issued, all other given arguments are ignored and the script terminates after outputting the information.

Chapter 4

Evaluation

4.1 Results

In this section, the results of the data collection process described in the previous chapter are discussed. For the sake of simplicity, not all results of all 72 events are presented. This work only demonstrates the relevance of the selected events when comparing normal and under attack behavior. Table 4.1 indicates what behaviors have been considered for which backdoor. For every attack behavior, one file was generated. Each file contains a total of 300 samples with ten second intervals. All behaviors were recorded sequentially. After finishing the data collection process for a backdoor implementation, the Raspberry Pi would be flashed with a new, clean operating system and thus, any traces of the previous backdoor would be erased. Each subsection contains one of the malware implementations and analyzes each of the possible behaviors of the respective malware. Additionally, every subsection includes a table containing statistical information like the standard deviation and mean. In said table the events are grouped by attack behavior and for each event there is statistical information for the clean, labeled as "clean", and infected behavior displayed. A complete version of the dataset is freely available on GitHub [58].

4.1.1 httpBackdoor

For "httpBackdoor", behavior1-3 were considered. In this section, two events are presented per behavior. Figure 4.1 and Figure 4.2 depict behavior1, whereas Figure 4.3 and Figure

Table 4.1: Attack Behaviors per Backdoor

Backdoor Implementation	**Considered Behaviors**
httpBackdoor	behavior1, behavior2, behavior3
backdoor	behavior1, behavior2, behavior3, behavior4
thetick	behavior1, behavior2, behavior3, behavior4, behavior5, behavior6

4.4 illustrate events under attack behavior2. Finally, 4.5 and Figure 4.6 show events while under attack behavior3.

In Figure 4.1 it can be observed, that the RAM usage is significantly higher while the devices is experiencing attack behavior1 than during the normal behavior. The numerical data of this event is also presented in Table 4.2. A possible explanation for that could be the additionally required resources by the running "httpBackdoor" Python file. On the other hand, there are also events that do not offer such a clear picture. In the case of "httpBackdoor" exhibiting behavior1, an examples is presented in Figure 4.2.

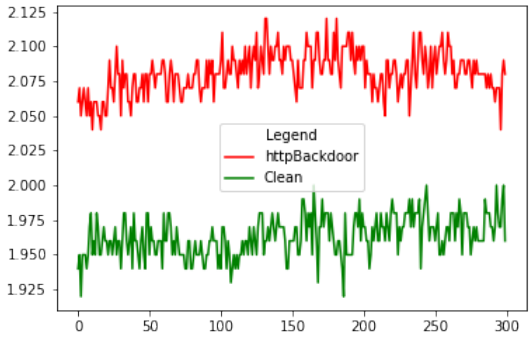

Figure 4.1: RAM usage in % for "http-Backdoor" during behavior1

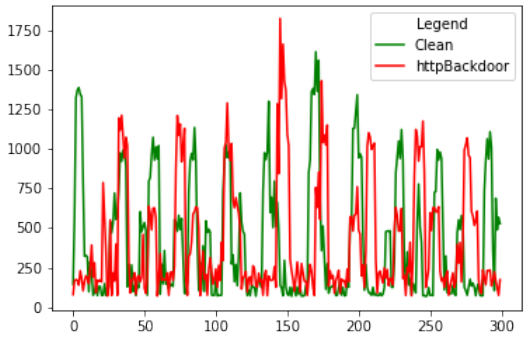

Figure 4.2: mm_page_free event for "httpBackdoor" during behavior1

Opposed to Figure 4.2, in both figures, Figure 4.3 and Figure 4.4, the events under attack behavior2 are vastly different from the clean behavior. What is most striking is the much greater variation in both events. That is supported by the greater standard deviation visible in Table 4.2 in the behavior2 column. It is logical for behavior2 to cause a lot of activity in the device since it is among the more complex of attack behaviors.

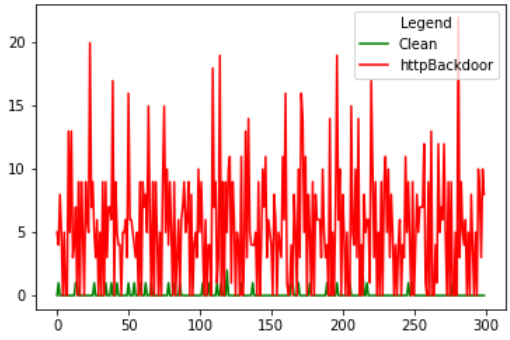

Figure 4.3: block_unplug event for "httpBackdoor" during behavior2

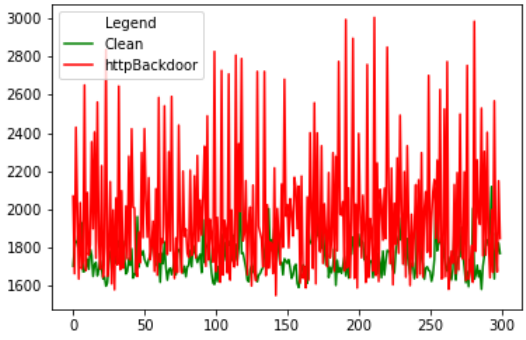

Figure 4.4: softirq_entry event for "httpBackdoor" during behavior2

Another clear contrast is presented in Figure 4.5. The figure clearly shows the event being different while under attack behavior3. For this event, the standard deviation of the infected behavior is almost eight times as high. Also, the green line is barely visible since there is little activity in the clean behavior as 75% of the values are equal to 0.

Figure 4.6 does not provide any clear indication of any unambiguous differences between the clean behavior and under attack behavior3.

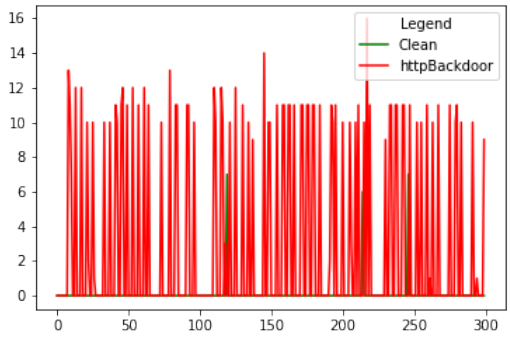

Figure 4.5: block_bio_backmerge event for "httpBackdoor" during behavior3

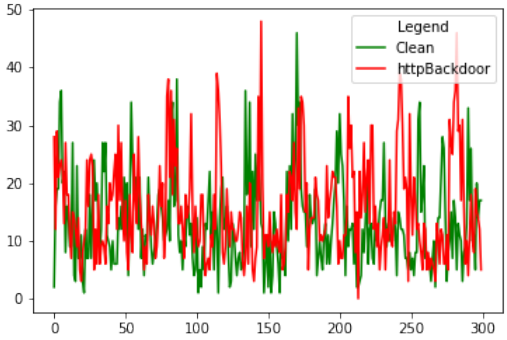

Figure 4.6: cpu-migrations event for "httpBackdoor" during behavior3

Table 4.2: Statistical information of "httpBackdoor"

	behavior1		behavior2		behavior3	
	RAM usage %		block_unplug		block_bio_backmerge	
	clean	http	clean	http	clean	http
mean	1.96	2.08	0.12	5.41	0.07	2.89
std	0.01	0.02	0.35	4.46	0.67	4.79
min	1.92	2.04	0.00	0.00	0.00	0.00
25%	1.95	2.07	0.00	1.00	0.00	0.00
50%	1.96	2.08	0.00	5.00	0.00	0.00
75%	1.97	2.09	0.00	8.25	0.00	9.25
max	2.00	2.12	2.00	22.00	7.00	16.00

4.1.2 backdoor

For "backdoor", behavior1-4 were considered. For each behavior, two events are presented in this section. For behavior1, the qdisc_dequeue event in Figure 4.7 and the network_in metric in Figure 4.8 are described. Figure 4.10 and Figure 4.9 follow and decribe behavior2. Next, behavior3 is illustrated in Figure 4.11 and Figure 4.12. Finally, Figure 4.13 and Figure 4.14 represent behavior.

There are small but notable differences between the clean and under attack behavior1 as can be observed by investigating Figure 4.7. The suspicion is confirmed in Table 4.3 since both, mean and standard deviation, are higher during the attack. However, in Figure 4.8, these differences are not as clear if outliers are ignored. It seems that in the clean and attack behavior1, the events are fairly consistent and do not indicate any special activity. Since behavior1 is rather simple and does not cause a lot of network traffic, it is logical that the network_in metric is not very different compared to the clean behavior.

However, by looking at Figure 4.9, depicting the incoming network traffic during attack behavior2, it is evident that the attack behaviors cause different device behavior. The

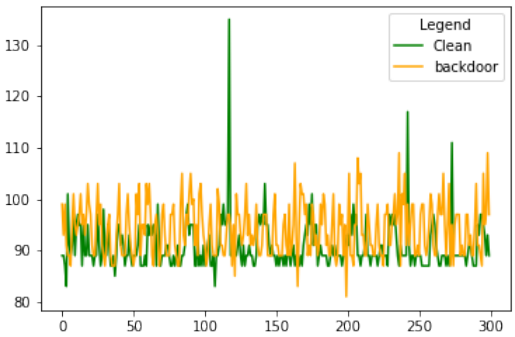

Figure 4.7: qdisc_dequeue event for "backdoor" during behavior1

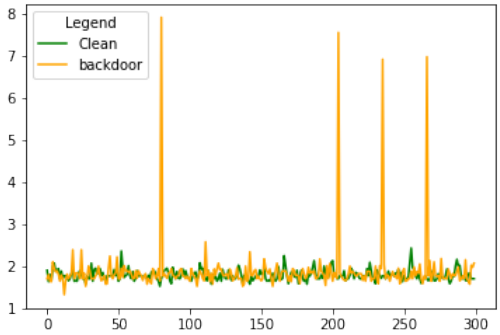

Figure 4.8: network_in in kBit/s for "backdoor" during behavior1

traffic is substantially higher compared to Figure 4.8. That phenomenon can be explained due to the repository cloning that is being caused by attack behavior2. While the clean behavior is very stable and consistent, the infected behavior seems to have higher variation, as confirmed by Table 4.3. Figure 4.10 illustrates the mm_page_alloc event, a memory allocation metric. It is clear that during attack behavior2 there is more activity in the memory allocation system.

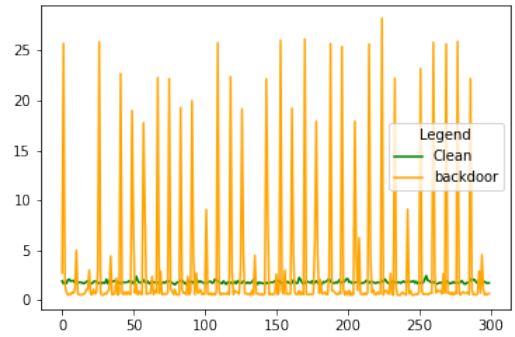

Figure 4.9: network_in in kBit/s for "backdoor" during behavior2

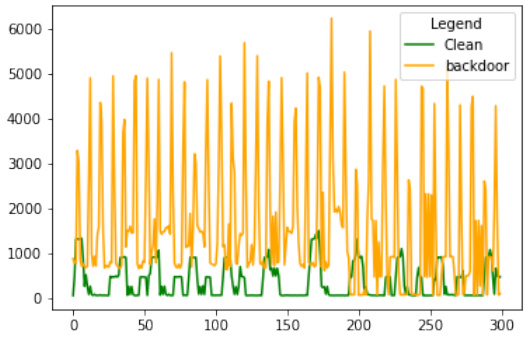

Figure 4.10: mm_page_alloc event for "backdoor" during behavior2

Two further events that may indicate the presence backdoor under attack behavior3 are illustrated in Figure 4.11 and Figure 4.12. Both events show vast differences compared to the normal behavior. While the writeback_dirty_inode event has a similar pattern in both the clear and attack behavior3, the level of the attack data is generally higher. From a statistical view, there are significant differences in both, mean and standard deviation for the mm_page_pcpu_drain event, as it is observable in Table 4.3.

In Figure 4.13, the difference between the clean and attack behavior4 is quite large. The kfree metric describes memory that was being used and has been freed. It seems that while the device is under attack a lot less memory is being freed, possibly due to the fact that this memory is occupied by the backdoor implementation itself or by the processes the attack behavior has caused. The differences are even clearer in numerical form as

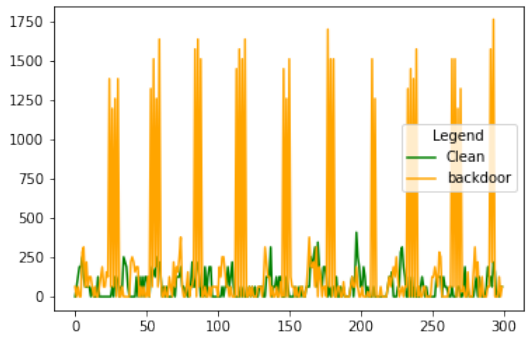

Figure 4.11: mm_page_pcpu_drain event for ”backdoor” during behavior3

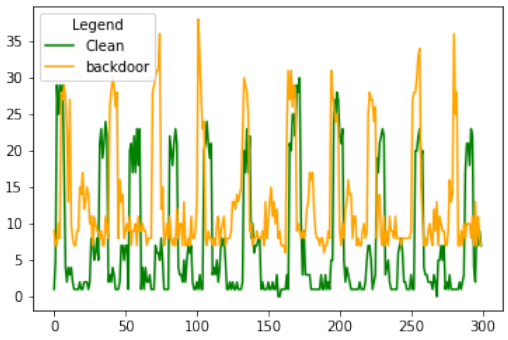

Figure 4.12: writeback_dirty_inode event for ”backdoor” during behavior3

can be observed in Table 4.3 in the behavior4 column. further On the other hand, the sys_enter event offers a less clear picture which is further illustrated in Figure 4.14.

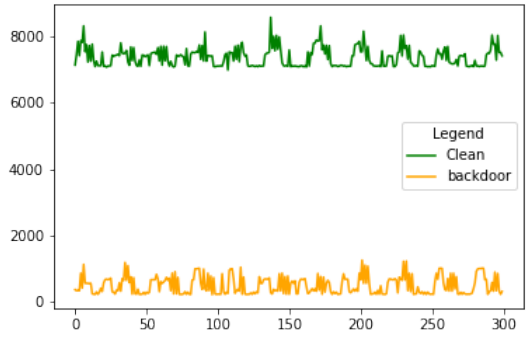

Figure 4.13: kfree event for ”backdoor” during behavior4

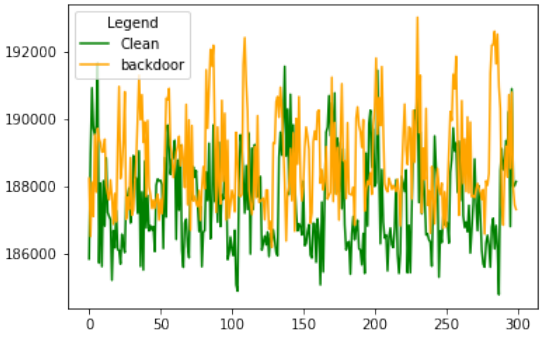

Figure 4.14: sys_enter event for ”backdoor” during behavior4

Table 4.3: Statistical information of ”backdoor”

	behavior1 qdisc_dequeue		behavior2 network_in		behavior3 mm_page_pcpu_drain		behavior4 kfree	
	clean	backdoor	clean	backdoor	clean	backdoor	clean	backdoor
mean	90.80	94.14	1.78	3.16	66.23	218.89	7369.33	491.72
std	4.69	5.16	0.13	6.56	78.94	446.69	280.77	266.24
min	83.00	81.00	1.52	0.43	0.00	0.00	6986.00	213.00
25%	89.00	89.00	1.70	0.54	0.00	0.00	7109.00	247.25
50%	89.00	94.00	1.76	0.66	63.00	63.00	7313.00	370.00
75%	93.00	97.00	1.85	1.02	126.00	133.75	7524.25	677.25
max	135.00	109.00	2.43	28.23	409.00	1764.00	8573.00	1254.00

4.1.3 thetick

Since ”thetick” can exhibit all six different attack behaviors, only one event is presented per behavior. Figure 4.15 presents the mm_page_alloc_zone_locked during behavior1. For behavior2, Figure 4.16 describes the incoming network traffic. In Figure 4.17 and Figure 4.19, the same event describes behavior3 and behavior5 and their differences. For behavior4, the fib_table_lookup event is illustrated in fib_table_lookup 4.18. Finally, in Figure 4.20 behavior6 is represented.

In Figure 4.15, there is another memory allocation related event illustrated under attack behavior1. There are clear variations visible in the figure as well as the data itself in Table 4.4. Figure 4.16 displays the same event, also during attack behavior2, as in Figure 4.9 with a similar result. It can be observed that although the attacks were conducted using different backdoors, the caused device behavior is very similar. That shows the integrity of the defined attack behaviors across different backdoor implementations.

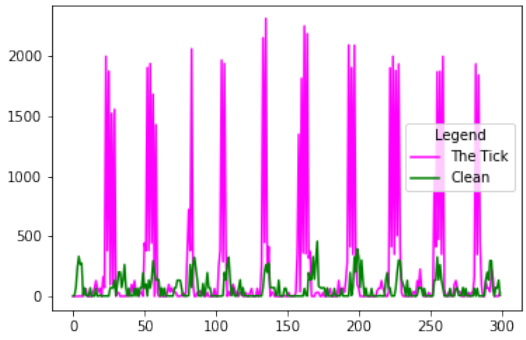

Figure 4.15: mm_page_alloc_zone_locked event for ”thetick” during behavior1

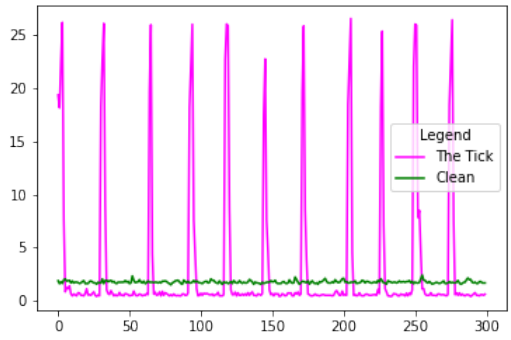

Figure 4.16: network_in in kBit/s for ”thetick” during behavior2

Another interesting observation is to be made regarding Figure 4.17 and 4.19. Although it is the same performance event, in the latter figure there is no discernible difference between the two behaviors. The explanation for the differences lies in the different attack behaviors. While behavior3 deletes files on the Raspberry Pi, behavior5 simply resolves domains at the Raspberry Pi and therefore, does not interact with the file system. These fundamental differences are clearly visible. It is important to see that certain events can be more or less efficient to detect certain attack behaviors than others. In Figure 4.18, similar to Figure 4.3, there is a striking contrast in terms of standard deviation between the clean and attack behavior4. Additionally, Table 4.4 shows significant numerical differences in the aforementioned statistics for the fib_table_lookup event under attack behavior4.

The results of the workqueue_activate_work event are presented Figure 4.20. It seems that this event might also be a suitable candidate to detect attacks. In the Linux architecture, work queues are a system to manage and distribute the workload to different kernel threads. The workqueue_activate_work event is triggered every time a task is assigned to a thread and subsequently activated. This event seems to have been triggered more during the attacks than during the normal behavior as can be observed in Table 4.4. A

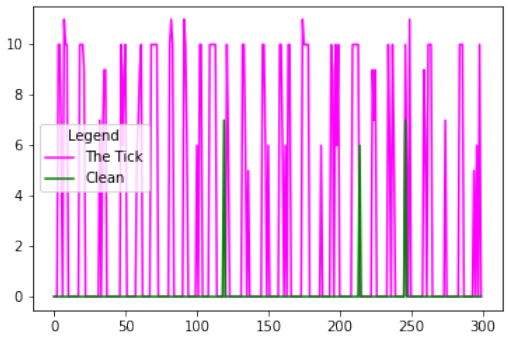

Figure 4.17: block_dirty_buffer event for "thetick" during behavior3

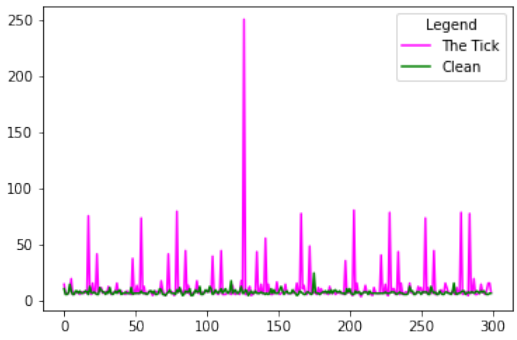

Figure 4.18: fib_table_lookup event for "thetick" during behavior4

possible reason for the difference is the additional actions that are caused by the backdoor during attack behavior6.

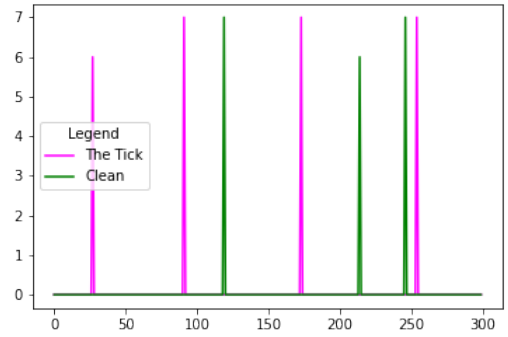

Figure 4.19: block_dirty_buffer event for "thetick" during behavior5

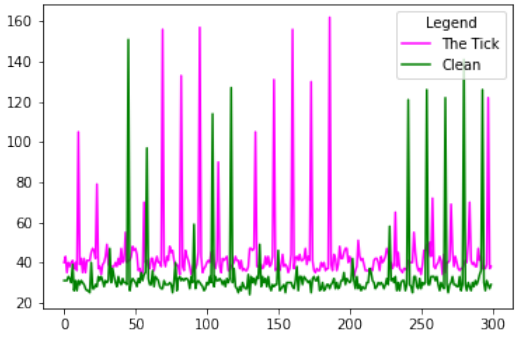

Figure 4.20: workqueue_activate_work event for "thetick" during behavior6

Table 4.4: Statistical information of "thetick"

	behavior1		behavior2		behavior4		behavior6	
	mm_page_alloc_zone_locked		network_in		fib_table_lookup		workqueue_activate_work	
	clean	tick	clean	tick	clean	tick	clean	tick
mean	69.67	253.81	1.78	3.15	7.87	12.79	33.13	44.08
std	92.24	560.89	0.13	6.71	2.07	19.62	16.85	18.56
min	3.00	0.00	1.52	0.43	5.00	4.00	24.00	33.00
25%	4.00	2.00	1.70	0.54	7.00	7.00	28.00	37.00
50%	10.00	33.00	1.76	0.60	7.00	8.00	30.00	40.00
75%	80.25	130.25	1.85	0.80	9.00	10.00	32.00	43.00
max	459.00	2312.00	2.43	26.49	25.00	251.00	151.00	162.00

4.2 Discussion

This work focuses on modeling the internal behavior of resource-constrained devices while being under several attack behaviors from backdoors. In the literature, there are currently very few datasets of this kind and none of them regarding IoT devices and backdoors. Most malware detection approaches and datasets follow the approach of modeling device behavior using network related metrics. Promising dimensions such as system calls, resource usage and HPCs do exist and are being considered more and more. The results in the previous subsection do support that claim and are promising in general. In many cases, there were notable and significant differences between the clean and the infected behavior, as can be observed in Figure 4.1 or Figure 4.13. Often, the standard deviation seems to be an indicator. The greater variation is visible in many figures presented in the previous section. It has been established that certain events are more suitable to detect certain attack behaviors than others. As seen in Figure 4.9, Figure 4.16 and Figure 4.16, the network_in metric seems a valuable indicator to detect behavior2, regardless which backdoor was used to conduct the attack. It is intuitive since behavior2 includes the cloning of repositories and therefore, downloading potentially a large amount of data. By comparing Figure 4.17 and Figure 4.19, it also becomes evident that the used event is more suited to detect behavior3 rather than behavior5.

FabIoT was produced using three real-world backdoor implementations found on GitHub. While they do serve their purpose, one could argue that they are not highly sophisticated malware which is true. One limitation of the dataset is that highly sophisticated implementations are not considered. The reason for that is that these implementations are in many cases highly specific to the target platform and extremely complex. Another limitation is that only backdoors were considered. Other types of malware were not used in this dataset, meaning that its results may not applicable for the detection of other malware since other malware may exhibit vastly different behavior. However, these backdoors do share some traits with other malware families as has been discussed in the previous chapter. Since most real-world malware attacks consist of multiple malware families, an interesting extension of this dataset and perhaps even a future approach would be the combination of multiple malware implementations. Also, since many HPCs were not supported by the test devices' kernel, the inclusion of these additional events to the dataset could prove a valuable extension.

Chapter 5

Summary and Conclusions

The increasing demand for IoT devices and services leads to an ever growing number of IoT concepts, platforms and networks, such as crowdsensing. The test device used in the present work is part of a real-world IoT crowdsensing platform, ElectroSense [44], which has the goal to monitor the radio frequency spectrum on a global scale. IoT platforms and networks, such as ElectroSense, are often vulnerable to cyber attacks by spyware and backdoors [3]. A promising approach to mitigate cyber attacks and also zero-day attacks is the application of device behavior fingerprinting and ML models. Recent works have identified the usage of resources and HPCs as promising metrics to model the internal behavior of a device [51]. The present thesis proposes FabIoT, a novel dataset modeling the internal behavior of a Raspberry Pi acting as an ElectroSense sensor while being attacked by backdoors with different behaviors. Building upon previous research, the present work shows that detecting backdoors by monitoring the internal behavior of a device is feasible [6, 24, 53]. A script to monitor the internal device behavior while the device is under attack from three real-world backdoors is developed as part of the present work. The backdoors exhibit up to six different attack behaviors to capture a wide variety of independent attacks. The monitoring script outputs a file containing 300 samples of 72 performance events for each possible combination of backdoor and attack behavior. The interval between samples is ten seconds. Then, is was presented that there are significant differences in the collected data and therefore, illustrated the value FabIoT provides. Preliminary results indicate that depending on the behavior different events are more suited to detect attacks than others.

Aligned with the current trend, future work in this field contains most likely the creation and development of further datasets. Building upon FabIoT future datasets could include device fingerprinting while under attack from other types of malware dangerous to IoT networks and platforms, such as botnets and ransomware. Another interesting approach would be to monitor several devices in a real-world IoT platform and also conduct attacks on a larger scale compared to the present thesis where the focus was on a single device. Moreover, researchers could consider combinations of several malware families as this is often the case in the wild. Another promising avenue to explore is the applicability of HPCs for resource-constrained devices such as the Raspberry Pi used in the present thesis.

Bibliography

[1] J. Ali, A. S. Khalid, E. Yafi, S. Musa, and W. Ahmed. Towards a secure behavior modeling for iot networks using blockchain. *arXiv preprint arXiv:2001.01841*, 2020.

[2] A. Alsaedi, N. Moustafa, Z. Tari, A. Mahmood, and A. Anwar. Ton_iot telemetry dataset: a new generation dataset of iot and iiot for data-driven intrusion detection systems. *IEEE Access*, 8:165130–165150, 2020. DOI: 10.1109/ACCESS.2020.3022862.

[3] R. Ande, B. Adebisi, M. Hammoudeh, and J. Saleem. Internet of things: evolution and technologies from a security perspective. *Sustainable Cities and Society*, 54:101728, 2020. DOI: https://doi.org/10.1016/j.scs.2019.101728.

[4] S. Barbhuiya, Z. Papazachos, P. Kilpatrick, and D. S. Nikolopoulos. Rads: real-time anomaly detection system for cloud data centres. *arXiv preprint arXiv:1811.04481*, 2018.

[5] K. Basu, P. Krishnamurthy, F. Khorrami, and R. Karri. A theoretical study of hardware performance counters-based malware detection. *IEEE Transactions on Information Forensics and Security*, 15:512–525, 2020. DOI: 10.1109/TIFS.2019.2924549.

[6] R. Bridges, J. Hernández Jiménez, J. Nichols, K. Goseva-Popstojanova, and S. Prowell. Towards malware detection via cpu power consumption: data collection design and analytics. In *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)*, pages 1680–1684, 2018. DOI: 10.1109/TrustCom/BigDataSE.2018.00250.

[7] R. Canzanese, S. Mancoridis, and M. Kam. System call-based detection of malicious processes. In *2015 IEEE International Conference on Software Quality, Reliability and Security*, pages 119–124, 2015. DOI: 10.1109/QRS.2015.26.

[8] L. F. Carvalho, T. Abrão, L. d. S. Mendes, and M. L. Proença. An ecosystem for anomaly detection and mitigation in software-defined networking. *Expert Systems with Applications*, 104:121–133, 2018. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2018.03.027.

[9] U. o. N. M. Computer Science Department. Computer immune systems. 2021. URL: https://www.cs.unm.edu/~immsec/systemcalls.htm (visited on 06/15/2021).

[10] G. Creech. *Developing a high-accuracy cross platform Host-Based Intrusion Detection System capable of reliably detecting zero-day attacks*. PhD thesis, University of New South Wales, 2014.

[11] G. Creech and J. Hu. A semantic approach to host-based intrusion detection systems using contiguousand discontiguous system call patterns. *IEEE Transactions on Computers*, 63(4):807–819, 2013. DOI: 10.1109/TC.2013.13.

[12] Curl.se. 2021. URL: https://curl.se/ (visited on 06/15/2021).

[13] R. Damasevicius, A. Venckauskas, S. Grigaliunas, J. Toldinas, N. Morkevicius, T. Aleliunas, and P. Smuikys. Litnet-2020: an annotated real-world network flow dataset for network intrusion detection. *Electronics*, 9(5), 2020. DOI: 10.3390/electronics9050800.

[14] Electrosense. Es-sensor. 2020. URL: https://github.com/electrosense/es-sensor (visited on 07/15/2021).

[15] E. Gandotra, D. Bansal, and S. Sofat. Malware analysis and classification: a survey. *Journal of Information Security*, 2014, 2014.

[16] E. Gandotra, D. Bansal, and S. Sofat. Zero-day malware detection. In *2016 Sixth International Symposium on Embedded Computing and System Design (ISED)*, pages 171–175, 2016. DOI: 10.1109/ISED.2016.7977076.

[17] S. Garcia, A. Parmisano, and M. J. Erquiaga. Iot-23: a labeled dataset with malicious and benign iot network traffic. 2020. URL: https://www.stratosphereips.org/datasets-iot23 (visited on 07/09/2021).

[18] T. Golomb, Y. Mirsky, and Y. Elovici. Ciota: collaborative iot anomaly detection via blockchain, 2018. arXiv: 1803.03807 [cs.CY].

[19] K. Haefner and I. Ray. Complexiot: behavior-based trust for iot networks. In *2019 First IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, pages 56–65, 2019. DOI: 10.1109/TPS-ISA48467.2019.00016.

[20] I. Hafeez, M. Antikainen, A. Y. Ding, and S. Tarkoma. Iot-keeper: detecting malicious iot network activity using online traffic analysis at the edge. *IEEE Transactions on Network and Service Management*, 17(1):45–59, 2020. DOI: 10.1109/TNSM.2020.2966951.

[21] W. Haider, J. Hu, J. Slay, B. Turnbull, and Y. Xie. Generating realistic intrusion detection system dataset based on fuzzy qualitative modeling. *Journal of Network and Computer Applications*, 87:185–192, 2017. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2017.03.018.

[22] A. Hamza, H. H. Gharakheili, T. A. Benson, and V. Sivaraman. Detecting volumetric attacks on lot devices via sdn-based monitoring of mud activity. In *Proceedings of the 2019 ACM Symposium on SDN Research*, SOSR '19, pages 36–48, San Jose, CA, USA. Association for Computing Machinery, 2019. ISBN: 9781450367103. DOI: 10.1145/3314148.3314352.

[23] S. Hashemi and M. Zarei. Internet of things backdoors: resource management issues, security challenges, and detection methods. *Transactions on Emerging Telecommunications Technologies*, 32(2):e4142, 2021. DOI: https://doi.org/10.1002/ett.4142.

[24] S. He, W. Ren, T. Zhu, and K.-K. R. Choo. Bosmos: a blockchain-based status monitoring system for defending against unauthorized software updating in industrial internet of things. *IEEE Internet of Things Journal*, 7(2):948–959, 2020. DOI: 10.1109/JIOT.2019.2947339.

[25] K. Inc. Insomnia.com. 2021. URL: https://insomnia.rest/ (visited on 06/15/2021).

[26] P. Inc. Postman.com. 2021. URL: https://www.postman.com/ (visited on 06/15/2021).

[27] S. Inc. The heartbleed bug. 2020. URL: https://heartbleed.com (visited on 07/18/2021).

[28] D. Javaheri, M. Hosseinzadeh, and A. M. Rahmani. Detection and elimination of spyware and ransomware by intercepting kernel-level system routines. *IEEE Access*, 6:78321–78332, 2018. DOI: 10.1109/ACCESS.2018.2884964.

[29] H. Kang, D. H. Ahn, G. M. Lee, J. D. Yoo, K. H. Park, and H. K. Kim. Iot network intrusion dataset. 2019. URL: https://dx.doi.org/10.21227/q70p-q449 (visited on 07/09/2021).

[30] J. Koritar. Backdoor. 2020. URL: https://github.com/jakoritarleite/backdoor (visited on 07/07/2021).

[31] D. Kreutz, F. M. Ramos, and P. Verissimo. Towards secure and dependable software-defined networks. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 55–60, New York, NY, USA. Association for Computing Machinery, 2013. ISBN: 9781450321785. DOI: 10.1145/2491185.2491199.

[32] A. H. Lashkari, A. F. A.Kadir, H. Gonzalez, K. F. Mbah, and A. A. Ghorbani. Android adware and general malware dataset (cic-aagm2017). 2017. URL: https://www.unb.ca/cic/datasets/android-adware.html (visited on 07/09/2021).

[33] A. H. Lashkari, A. F. A.Kadir, H. Gonzalez, K. F. Mbah, and A. A. Ghorbani. Investigation of the android malware (cic-invesandmal2019). 2019. URL: https://www.unb.ca/cic/datasets/invesandmal2019.html (visited on 07/09/2021).

[34] A. H. Lashkari, A. F. A.Kadir, H. Gonzalez, K. F. Mbah, and A. A. Ghorbani. Towards a network-based framework for android malware detection and characterization. In *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, pages 233–23309, 2017. DOI: 10.1109/PST.2017.00035.

[35] A. H. Lashkari, A. F. A. Kadir, L. Taheri, and A. A. Ghorbani. Android malware dataset (cic-andmal2017). 2018. URL: https://www.unb.ca/cic/datasets/andmal2017.html (visited on 07/09/2021).

[36] A. H. Lashkari, A. F. A. Kadir, L. Taheri, and A. A. Ghorbani. Toward developing a systematic approach to generate benchmark android malware datasets and classification. In *2018 International Carnahan Conference on Security Technology (ICCST)*, pages 1–7, 2018. DOI: 10.1109/CCST.2018.8585560.

[37] W. Li, W. Meng, and L. F. Kwok. A survey on openflow-based software defined networks: security challenges and countermeasures. *Journal of Network and Computer Applications*, 68:126–139, 2016. DOI: https://doi.org/10.1016/j.jnca.2016.04.011.

[38] Y. Lu and L. D. Xu. Internet of things (iot) cybersecurity research: a review of current research topics. *IEEE Internet of Things Journal*, 6(2):2103–2115, 2019. DOI: 10.1109/JIOT.2018.2869847.

[39] F. Martinelli, F. Mercaldo, and A. Saracino. Bridemaid: an hybrid tool for accurate detection of android malware. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, pages 899–901, Abu Dhabi, United Arab Emirates. Association for Computing Machinery, 2017. ISBN: 9781450349444. DOI: 10.1145/3052973.3055156.

[40] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, and Y. Elovici. N-baiot—network-based detection of iot botnet attacks using deep autoencoders. *IEEE Pervasive Computing*, 17(3):12–22, 2018. DOI: 10.1109/MPRV.2018.03367731.

[41] O. Or-Meir, N. Nissim, Y. Elovici, and L. Rokach. Dynamic malware analysis in the modern era—a state of the art survey. *ACM Comput. Surv.*, 52(5), Sept. 2019. ISSN: 0360-0300. DOI: 10.1145/3329786.

[42] P. Mishra, V. Varadharajan, E. S. Pilli, and U. Tupakula. Vmguard: a vmi-based security architecture for intrusion detection in cloud environment. *IEEE Transactions on Cloud Computing*, 8(3):957–971, 2020. DOI: 10.1109/TCC.2018.2829202.

[43] nccgroup. The tick. 2020. URL: https://github.com/nccgroup/thetick (visited on 07/07/2021).

[44] T. E. Network. Electrosense - collaborative spectrum monitoring. 2021. URL: https://electrosense.org (visited on 07/01/2021).

[45] M.-O. Pahl and F.-X. Aubet. All eyes on you: distributed multi-dimensional iot microservice anomaly detection. In *2018 14th International Conference on Network and Service Management (CNSM)*, pages 72–80, 2018.

[46] Perf wiki. 2020. URL: https://perf.wiki.kernel.org/index.php/Main%5C_Page (visited on 07/13/2021).

[47] F. Pierazzi, G. Mezzour, Q. Han, M. Colajanni, and V. S. Subrahmanian. A data-driven characterization of modern android spyware. *ACM Trans. Manage. Inf. Syst.*, 11(4), 2020. DOI: 10.1145/3382158.

[48] R. Ravichandiran, H. Bannazadeh, and A. Leon-Garcia. Anomaly detection using resource behaviour analysis for autoscaling systems. In *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pages 192–196, 2018. DOI: 10.1109/NETSOFT.2018.8460025.

[49] R. Reynolds. The four biggest malware threats to uk businesses. *Network Security*, 2020(3):6–8, 2020. ISSN: 1353-4858. DOI: https://doi.org/10.1016/S1353-4858(20)30029-5.

[50] G. Roualland. Ifstat. URL: https://linux.die.net/man/1/ifstat (visited on 07/13/2021).

[51] P. M. Sánchez Sánchez, J. M. Jorquera Valero, A. Huertas Celdrán, G. Bovet, M. Gil Pérez, and G. Martínez Pérez. A survey on device behavior fingerprinting: data sources, techniques, application scenarios, and datasets. *IEEE Communications Surveys Tutorials*, 23(2):1048–1077, 2021. DOI: 10.1109/COMST.2021.3064259.

[52] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli. Madam: effective and efficient behavior-based android malware detection and prevention. *IEEE Transactions on Dependable and Secure Computing*, 15(1):83–97, 2018. DOI: 10.1109/TDSC.2016.2536605.

[53] H. Sayadi, H. M. Makrani, S. M. Pudukotai Dinakarrao, T. Mohsenin, A. Sasan, S. Rafatirad, and H. Homayoun. 2smart: a two-stage machine learning-based approach for run-time specialized hardware-assisted malware detection. In pages 728–733, 2019. DOI: 10.23919/DATE.2019.8715080.

[54] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani. Cse-cic-ids2018. 2018. URL: https://www.unb.ca/cic/datasets/ids-2018.html (visited on 07/09/2021).

[55] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani. Intrusion detection evaluation dataset (cic-ids2017). 2018. URL: https://www.unb.ca/cic/datasets/ids-2017.html (visited on 07/09/2021).

[56] B. Singh, D. Evtyushkin, J. Elwell, R. Riley, and I. Cervesato. On the detection of kernel-level rootkits using hardware performance counters. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, pages 483–493, New York, NY, USA. Association for Computing Machinery, 2017. ISBN: 9781450349444. DOI: 10.1145/3052973.3052999.

[57] J. Singh and J. Singh. Detection of malicious software by analyzing the behavioral artifacts using machine learning algorithms. *Information and Software Technology*, 121:106273, 2020. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2020.106273.

[58] F. Sisi. Fabiot-dataset. 2021. URL: https://github.com/Fabiooo98/FabIoT-Dataset (visited on 07/15/2021).

[59] SkryptKiddie. Httpbackdoor. 2020. URL: https://github.com/SkryptKiddie/httpBackdoor (visited on 07/07/2021).

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

[61] U. Sydney. Data collected for acm sosr 2019. 2019. URL: https://iotanalytics.unsw.edu.au/attack-data (visited on 07/01/2021).

[62] L. Taheri, A. F. A. Kadir, and A. H. Lashkari. Extensible android malware detection and family classification using network-flows and api-calls. In *2019 International Carnahan Conference on Security Technology (ICCST)*, pages 1–8, 2019. DOI: 10.1109/CCST.2019.8888430.

[63] P. D. Team. Pyinstaller. 2021. URL: https://www.pyinstaller.org/ (visited on 07/01/2021).

[64] UIPath. Ui path. 2021. URL: https://www.uipath.com (visited on 07/18/2021).

[65] T. Yu, Y. Sun, S. Nanda, V. Sekar, and S. Seshan. RADAR: A robust behavioral anomaly detection for IoT devices in enterprise networks. Technical report, Technical Report CMU-CyLab-19-003, Carnegie Mellon University, 2019.

[66] B. Zhou, A. Gupta, R. Jahanshahi, M. Egele, and A. Joshi. Hardware performance counters can detect malware: myth or fact? In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ASIACCS '18, pages 457–468, Incheon, Republic of Korea. Association for Computing Machinery, 2018. ISBN: 9781450355766. DOI: 10.1145/3196494.3196515.

Abbreviations

ADFA	Australian Defense Force Academy
AR	AutoRegressive
CIC	Canadian Insitute for Cyber Security
CPU	Central Processing Unit
C&C	Command & Control
DDoS	Distributed Denial of Service
DoS	Denial of Service
EMM	Extended Markov Model
FIB	Forwarding Information Base
GPIO	General Purpose Input/Output
HPC	Hardware Performance Counter
IIoT	Industrial Internet of Things
IoT	Internet of Things
ML	Machine Learning
MMC	Multi Media Card
MUD	Manufacturer Usage Description
RADS	Real-time Anomaly Detection System
SDN	Software-Defined Networking
UNM	University of New Mexico
VM	Virtual Machine
VMI	Virtual Machine Introspection

Glossary

Autoscaling Autoscaling is a concept where a system provides more resources if indicated by the user. The system basically scales the amount of available resources to the needs of the user.

Behavior Fingerprinting Behavior Fingerprinting is the process of modeling the internal behavior of a device by monitoring different metrics of the device [51]. Possible metrics include network related activities, system calls, the usage of resources and HPCs among others.

Blockchain A blockchain is a digital, public and decentralized ledger, first introduced to be used in the context of crypto currencies [24]. The foundation for a blockchain network is a peer-to-peer network [1].

Brute Force In a Brute Force attack, the attacker tries to guess a password or pass code using a computer program. The program works in trial-and-error fashion, trying all possibilities until it found the correct one.

Crowd Sensing A concept where a large amount of individuals contribute small measurements to create a big collection of data.

Dynamic Analysis See **Behavior Fingerprinting**.

Flooding Flooding is a type of DDoS Attack where the goal is to overwhelm the target servers with a large amount of sent requests.

Heartbleed Heartbleed is a vulnerability in the cryptographic software OpenSSL. Under certain circumstances, this bug allowed attackers to steal information that should have been secured [27].

pcap Files These files contain network packet data and can be used to analyze network traffic.

Spoofing In a Spoofing attack a piece of software or a person identifies itself as a benevolent identity but has malicious intents. It can also reroute network traffic to malicious web sites and/or steal information and distribute malware in the process.

Static Analysis Static Analysis attempts to identify possibly malicious software by inspecting the code of said software without running it. In many cases, any found signature is compared to a database of known malware signatures.

Zero-Day Attacks Attacks that are conducted by malware that has never been seen before.

List of Figures

List of Tables

Appendix A

Installation Guidelines

A.1 Backdoors

The same installation instructions can be found on the GitHub page of the respective backdoor.

A.1.1 httpBackdoor

First of all, the repository needs to be clone onto the machine with the following command:

```
$ git clone https://github.com/SkryptKiddie/httpBackdoor.git
```

To execute this backdoor, Python needs to be installed. In the thesis, Python 3.5 was used but there is no minimal version mentioned by the author. Python can be installed using the following command:

```
$ sudo apt install python3
```

After installing python, the IP address of the Raspberry Pi needs to be inserted in the file *httpBackdoor.py* in line 5. Then, you should be able to execute the backdoor using the following command:

```
$ python3 httpBackdoor.py
```

It is possible that certain packages are not yet installed. This can be done with the following command.

```
$ pip3 install "package-name"
```

A.1.2 backdoor

Both on server and the client, the repository needs to be cloned using the following command:

```
$ git clone https://github.com/jakoritarleite/backdoor.git
```

Client

Again, Python needs to be installed for this backdoor. The installation process is the same as it is in the previous section.

There are 5 packages that need to be installed in order to run this backdoor. These are *socket*, *os*, *subprocess*, *time* and *sleep*.

As described in the previous section, the packages can be installed using the following command:

```
$ pip3 install "package-name"
```

The IP address of the Raspberry Pi also needs to be entered in the code in the file *client.py* on line 10.

After these steps, the client can be executed on the Raspberry Pi with the following command:

```
$ python3 client.py
```

Server

The same Python packages, *socket*, *os*, *subprocess*, *time* and *sleep*, need to be installed for the server as well.

Also, the IP address of the Raspberry Pi needs to be entered in the file *server.py* on line 7.

The server is started with the following command:

```
$ python3 server.py
```

A.1.3 thetick

Again, on server and the client, the repository needs to be cloned using the following command:

```
$ git clone https://github.com/nccgroup/thetick.git
```

Client

To compile the client executable the following commands needs to be executed in order to install the necessary dependencies.

```
$ sudo apt-get install libcurl-dev
```

After that, the makefile needs to be run using the following commands:

```
$ cd src
$ make clean
$ make
```

Once this process is finished, in the *bin* folder of the repository, there is an executable. The client is started by executing this command:

```
$ ./ticksvc ADDR PORT
```

ADDR and PORT refer to the IP address and port of the machine running the server is running.

Server

On the server side Python is required. In this thesis, Python 2.7 was used since 3.5 caused errors. To install the necessary dependencies, simply navigate to the repository folder and execute the following command:

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

The *requirements.txt* contains all necessary packages for the server which can then be started by executing the following command:

```
$ python server.py -b ADDR -p PORT
```

Again, ADDR and PORT refer to the IP address and port of the machine running the server.

A.2 Monitoring Script

As mentioned in section 3.4.2, the monitoring script is executed using the following command:

```
$ bash script.sh [-a] [-f] [-t]
```

There are two requirements that need to be installed on the system in order to run the script, *ifstat* and *perf*. The *ifstat* tool can be installed using the following command:

```
$ sudo apt-get install ifstat
```

While in many cases it is already installed on the device, the installation of the *perf* tool can be a bit more tedious. Since the tool is part of the *linux-tools* package, the package should be installed. Also, since *perf* interacts with certain kernel modules, *linux-tools-generic* and *linux-tools-common* should also be installed. The three packages can be installed using the following command:

```
$ sudo apt-get install "package-name"
```

However, it is possible that a kernel specific version of *perf* needs to be installed. That can be achieved using the following command:

```
$ sudo apt-get install linux-tools-‘uname -r‘
```

A.3 Attack Scripts

In order to repeat the attacks continuously, automated attack scripts were created. Since sending attack commands to ”httpBackdoor” can be done using the curl [12] tool, we have decided to write batch scripts to automate the attacking process for this backdoor. For behavior3 an additional preparation script was written in order to create a large amount of files which then could be deleted by the attack.

Since the server side of the other two backdoors consist of command line tools, a Robotic Process Automation (RPA) software was used to interact with the command line tools and continuously send attack commands. The software, UI Path [64], can be downloaded from the website specified in the reference. A free account is required, however. After an account has been created, the UI Path Studio can be downloaded. Using the IDE-like software, the scripts can be openend and executed.

Appendix B

Contents of the zip File

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

BA_Fabio_Sisi.pdf A PDF version of the final report.

BA Fabio Sisi.zip The LaTeX source code of the final report. It includes all used figures.

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

source_code This folder includes all software components that were created during the present thesis. It contains the following:

- **attack_scripts** A folder containing the attack scripts. In order to execute the scripts for the "backdoor" and "thetick", the software UI Path [64] is necessary. The scripts are grouped by backdoor, and the further sorted by attack behavior.
- **FabIoT-Dataset.zip** A zip file containing the repository of the collected dataset.
- **Monitoring-Script.zip** A zip file containing the repository of the created monitoring script.