MŰEGYETEM 1782

Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Network Systems and Services

Robustness Against Evasion of Similarity-based IoT Malware Detection Methods

Scientific Students' Association Report

Author:

József Sándor

Advisor:

dr. Levente Buttyán
Roland Nagy

2022

Contents

Abstract

Detecting malicious files before they are executed is a fundamental defense mechanism in computer-based systems, including in embedded systems and the Internet-of-Things (IoT). Indeed, with the dramatic increase of the number of deployed embedded IoT devices in the world, the number of known attacks against them has also increased in the recent past, and such attacks include infecting them with malware. Therefore, malware detection on embedded IoT devices became an active research area, resulting in several IoT malware detection methods. In this paper, we introduce two recently proposed solutions, SIMBIoTA and SIMBIoTA-ML. They both use binary similarity measures for detecting even previously unseen malware, and they have good detection performance, while being very resource efficient at the same time. In addition, SIMBIoTA-ML improves the malware detection capability of SIMBIoTA by taking advantage of machine learning.

The main problem addressed in this work is that current IoT malware detection methods, such as SIMBIoTA and SIMBIoTA-ML, are vulnerable to adversarial evasion techniques. This means that, knowing how the malware detection method works, attackers can create specifically crafted malware samples that mislead the detector and evade detection. Unfortunately, existing solutions are not necessarily robust against these type of attacks. In this paper, we demonstrate that creating malware samples that evade detection is relatively easy by proposing two simple adversary example creation strategies and showing that the robustness of SIMBIoTA and SIMBIoTA-ML against them is rather weak. Both strategies append bytes to the end of an existing malware sample such that the malware remains functional, but the new sample becomes dissimilar to the original sample, hence, evading detection by SIMBIoTA and SIMBIoTA-ML. The first strategy appends a chunk of the original sample, whereas the second strategy appends an entire benign file. We measure the robustness of SIMBIoTA and SIMBIoTA-ML against these strategies by measuring their detection accuracy on the crafted adversarial samples. It turns out that SIMBIoTA-ML is somewhat robust against the first strategy, but both SIMBIoTA and SIMBIoTA-ML completely fail against the second one.

To overcome this problem, we propose to use adversarial training as another contribution of this paper. Adversarial training has been used in the image recognition domain to increase the robustness of machine learning-based models against adversarial examples. We adopt this approach in the domain of malware detection and demonstrate its effectiveness. Adversarial training in our case means that that the training set of the malware detector algorithm is extended with samples that are crafted by using the adversarial evasion strategies that we proposed. We measure the detection accuracy of SIMBIoTA-ML trained on such an extended training set and show that it remains high both for the original malware samples and for the adversarial samples. The price that we have to pay for this remarkable robustness is the increased training time and the increased size of the detection model, however, we argue that both are bearable in practice.

Kivonat

A rosszindulatú fájlok futtatás előtti felismerése alapvető védelmi mechanizmus a számítógép-alapú rendszerekben, beleértve a beágyazott rendszereket és a tárgyak internetét (Internet-of-Things, IoT) is. A beágyazott IoT-eszközök számának drámai növekedésével az ellenük irányuló ismert támadások száma is megnőtt, melyek között hangsúlyosak a malware típusú támadások. Ezért a beágyazott IoT-eszközökön a malware programok detektálása aktív kutatási területté vált, és ennek eredményeképpen számos IoT malware detekciós módszer született. Ebben a tanulmányban két nemrégiben javasolt megoldást mutatunk be, a SIMBIoTA-t és a SIMBIoTA-ML-t. Mindkettő bináris hasonlósági mértékeket használ a korábban nem látott rosszindulatú programok felismerésére, továbbá azon kívül, hogy sikeresen felismerik a rosszindulatú fájlokat, hatékonyan bánnak az erőforrásokkal is. Ezen túlmenően a SIMBIoTA-ML a gépi tanulás előnyeit kihasználva javítja a SIMBIoTA detekciós képességét.

A fő probléma, amellyel ez a dolgozat foglalkozik, az, hogy a jelenlegi IoT malware detekciós módszerek, mint például a SIMBIoTA és a SIMBIoTA-ML, sebezhetőek az ún. adversarial technikákkal szemben. Ez azt jelenti, hogy a támadó a malware-t detektáló módszer működésének ismeretében olyan speciálisan kialakított malware mintákat hozhat létre, amelyek elkerülik a detekciót. Sajnos a meglévő megoldások nem feltétlenül robusztusak az ilyen típusú támadásokkal szemben. Ebben a tanulmányban két egyszerű adversarial minta létrehozási stratégiával megmutatjuk, hogy detektálást elkerülő rosszindulatú mintákat viszonylag könnyű létrehozni, és látni fogjuk, hogy a SIMBIoTA és a SIMBIoTA-ML robusztussága ezekkel szemben meglehetősen gyenge. Mindkét stratégia bájtokat csatol egy meglévő malware végéhez úgy, hogy a malware működőképes marad, de az új minta nem hasonlít az eredeti mintához, és így kikerüli a SIMBIoTA és a SIMBIoTA-ML általi észlelést. Az első stratégia az eredeti minta egy darabját, míg a második stratégia egy teljes jóindulatú fájlt csatol. A SIMBIoTA és a SIMBIoTA-ML robusztusságát ezekkel a stratégiákkal szemben úgy határozzuk meg, hogy megmérjük milyen pontossággal detektálják az adversarial mintákat. Kiderül, hogy a SIMBIoTA-ML viszonylag robusztus az első stratégiával szemben, de mind a SIMBIoTA, mind a SIMBIoTA-ML teljesen kudarcot vall a második stratégiával szemben.

Ennek a problémának a leküzdésére az adversarial tanítás módszerének alkalmazását javasoljuk. Az adversarial tanítást a képfelismerés területén már alkalmazták a gépi tanuláson alapuló modellek robusztusságának növelésére az adversarial mintákkal szemben. Ezt a megközelítést adaptáljuk a malware detekció területén, és bemutatjuk annak hatékonyságát. A mi esetünkben az adversarial tanítás azt jelenti, hogy a malware-t detektáló modell tanító halmazát olyan mintákkal bővítjük, amelyeket az általunk javasolt adversarial stratégiák segítségével alakítottunk ki. Megmérjük az ilyen kibővített tanító halmazon tanított SIMBIoTA-ML detekciós pontosságát, és megmutatjuk, hogy ez mind az eredeti malware minták, mind az adversarial minták esetében magas marad. Az ár, amelyet ezért a megnövekedett robusztusságért fizetnünk kell, a megnövekedett tanítási idő és a detektáló modell megnövekedett mérete, de kijelenthető, hogy mindkettő elviselhető mértékű a gyakorlatban.

Chapter 1

Introduction

An embedded device is a specialized device meant for specific purposes and it is usually embedded as part of a larger system. Nowadays, these devices are widespread and could be surprisingly diverse. The collection of these embedded devices that are connected to the Internet, together with their often cloud-based backend infrastructure, is called the Internet-of-Things (IoT).

Just like other types of computers, embedded IoT devices have security weaknesses. IoT devices can be found everywhere (e.g. healthcare, transportation, agriculture), therefore, their vulnerabilities represent a huge attack surface for attackers. IoT devices are desirable targets for attackers, because they handle sensitive information or they control critical processes. Indeed, with the dramatic increase of the number of deployed embedded IoT devices in the world, the number of known attacks against them has also increased in the recent past, and such attacks include infecting them with malware. Furthermore, they are highly connected over the Internet, so the given malware could spread easily from one to other. One of the most infamous examples is the Mirai malware [5], which infected hundreds of thousands of IoT devices and launched one of the largest distributed denial of service attacks against Internet-based services in 2016. But the IoT threat landscape includes other malware families as well, such as Gafgyt, Tsunami, and Dnsamp [11].

Malware detection is an essential part of modern defense mechanisms used in computer-based systems. Malware detection approaches can be categorized into signature-based, heuristic, and cloud-based approaches [7]. In the past, antivirus products only used signatures. A signature, in this context, is a short sequence of bytes that uniquely identifies a set of variants of a malware. Malware detection algorithms scan files and search for signatures. If the given signature is found in the file, the file is considered malware. In practice, however, signature-based detection has significant disadvantages. It is expensive, because signatures are usually created manual by experts [1]. Furthermore, signature-based detection can be mislead with various techniques (e.g. packing, encryption, obfuscation, code polymorphism). These techniques keep the effect of malware, while they make their characteristic signatures disappear.

Heuristic malware detection relies on rules, created by experts that capture more complex static patterns in malware than simple signatures do. Consequently, compared to signature-based approaches, heuristic techniques can detect a larger set of variants of the same malware. Yet, even this approach is unable to cope with obfuscation techniques. Furthermore, the threat landscape is constantly evolving with both new types of malware and variations of existing malware[1]. Both cases require new signatures and heuristic rules to be generated constantly and this poses a major challenge for antivirus companies.

Therefore, there is significant effort to automate the detection process using machine learning [13, 36, 35]. In order to extract features for machine learning, static and dynamic program analysis techniques are used [29]. Features include instruction-level data, data related to control-flow, invoked API functions and system calls, and messages sent over the network. These features are used to train machine learning models for malware detection. Furthermore, machine learning requires lots of training data, i.e., benign and malicious samples in this case.

Regarding the system architecture, nowadays, antivirus products install a client-side component on the users' machines, which typically performs signature-based and heuristic detection. If this client component cannot determine whether a sample is malicious or not, then it sends the sample to a server, which performs a more in-depth analysis, including e.g., dynamic behavior analysis in a sandbox environment. We refer to this architectural setup as cloud-based malware detection. Thanks to using dynamic behavior analysis, it is very effective, it can even cope with advanced evasion techniques (e.g. obfuscation, code polymorphism). Cloud-based approach can be applied to resource constrained IoT devices as well [32].

While many attacks can be successfully prevented with these approaches, unfortunately, no matter how good a malware detection system is, attackers constantly work on methods to evade their detection (this is a cat-and-mouse game). In particular, they want to construct malware samples that have the same function as older samples but not recognized by detectors. In case of machine learning, such kind of inputs are called adversarial examples. The concept of adversarial examples can also be adopted in the domain of malware detection (being machine learning-based or otherwise): an adversarial example in this context would be a specially crafted malware sample that evades detection by a specific detection method. The degree of vulnerability against adversarial examples defines the robustness of the system. History shows that traditional signature and heuristic malware detection is not robust against adversarial samples, which explains the large number of polymorphic malware. And unfortunately, it has been shown in the literature [6, 31, 27] that machine learning based malware detectors can also be misled easily.

In this paper, we compare two recent IoT malware detection solutions, SIMBIoTA and SIMBIoTA-ML, in terms of robustness. SIMBIoTA (SIMilarity Based IoT Antivirus) [34] is an effective and efficient IoT antivirus solution. SIMBIoTA is similar to traditional signature-based solutions, but it uses TLSH hash values of known malware instead of raw binary signatures. TLSH [25] is a similarity hash algorithm, which means it outputs similar hash values for similar inputs. SIMBIoTA-ML [26] improves the malware detection capability of SIMBIoTA with machine learning. In case of SIMBIoTA-ML, for training the machine learning model, feature vectors are extracted from the TLSH hash value of files.

For the comparison of SIMBIoTA and SIMBIoTA-ML in terms of robustness against adversarial examples, we design two adversarial example creation strategies. The purpose of each strategy is to create adversarial examples that evade similarity-based malware detection. Both strategies modify existing malware samples by appending extra bytes to them such that those bytes are never executed but they make the modified samples dissimilar

[1] `https://www.sophos.com/en-us/medialibrary/pdfs/technical-papers/sophoslabs-2019-threat-report.pdf`, (accessed: November 22, 2022)
`https://www.ntsc.org/assets/pdfs/cyber-security-report-2020.pdf`, (accessed: November 22, 2022)

to the original ones. The first strategy adds chunks of the original sample to the malware and ensures that a certain target difference is achieved by doing so. The second strategy embeds a malware into a known benign file and ensures that the resulting sample becomes similar to the benign file (and hence dissimilar to the original malware sample). We show by measurements that SIMBIoTA-ML is robust against the first strategy, but it can be misled by the second one, while SIMBIoTA has poor robustness against both strategies.

To overcome this problem, we propose to use adversarial training as another contribution of this paper. Adversarial training has been used in the image recognition domain to increase the robustness of machine learning-based models against adversarial examples. We adopt this approach in the domain of malware detection and demonstrate its effectiveness. Adversarial training in our case means that that the training set of the malware detector algorithm is extended with samples that are crafted by using the adversarial evasion strategies that we proposed. We measure the detection accuracy of SIMBIoTA-ML trained on such an extended training set and show that it remains high both for the original malware samples and for the adversarial samples. The price that we have to pay for this remarkable robustness is the increased training time and the increased size of the detection model, however, we argue that both are bearable in practice.

This paper is organized as follows. In Chapter 2 we take a closer look at the design and performance of SIMBIoTA and SIMBIoTA-ML. Chapter 3 presents in detail the strategies for creating adversarial examples. In Chapter 4, we describe our methodology used to measure the performance of the proposed adversarial example creation strategies, and we also present the results of our measurements. In Chapter 5, we present our proposed countermeasure, i.e., adversarial training, and the measurement results showing that it makes SIMBIoTA-ML robust against evasion by adversarial samples. In Chapter 6 we show the current state of the art in this specific scientific field. Finally, Chapter 7 concludes the paper.

Chapter 2

Similarity-based IoT malware detection

Although, IoT malware detection is a challenging problem, there have been solutions proposed in the literature [32, 19]. In this work, we are interested in the similarity-based IoT malware detection solutions called SIMBIoTA and SIMBIoTA-ML, which have been proposed recently, and which have remarkable malware detection capabilities, while being resource efficient at the same time. Before we delve into the architecture of these solutions, we get to know the meaning of similarity hashes that form the basis of the mentioned systems.

2.1 Binary similarity hash function

Cryptographic hashes such as MD5 and SHA-1 are used for many data mining and security applications. The collision resistance property of cryptographic hash functions make them suitable for unique identification of files in practice. However, if a single byte of a file is changed, then its cryptographic hash will be a completely different hash value. The situation is different for similarity hashes: for similar inputs, binary similarity hash functions output similar hash values. SIMBIoTA and SIMBIoTA-ML use TLSH hash function as the basis of their system [25], which is also a similarity based hash function.

TLSH has a lightweight calculation time in the range of milliseconds on contemporary personal computers, which makes it suitable in the context of malware detection even on resource constrained IoT devices. A TLSH value is relatively short, it can be represented in 36 bytes (35 byte hash value +1 byte for versioning), which is also an advantage. More specifically, computing a TLSH hash value involves the following steps [25]:

1. Process the raw byte string using a sliding window of size 5 to populate an array of bucket counts.
2. Calculate quartile points q1, q2, and q3 based on the buckets' values.
3. Construct the hash value's header based on the quartile points, the length of the byte string, and a checksum on its content.
4. Construct the hash value's body.

The first three bytes of the resulting TLSH hash value is a header with following parts:

- the first byte is a checksum value;
- the second byte stores the so-called L value, which is calculated from (the logarithm of) the length of the original byte sequence;
- the two nibbles of the third byte are called the Q1 and Q2 ratios, and they are computed from the quartile points q1 and q3, and the quartile points q2 and q3, respectively.

The rest of the bytes are the binary representations of the 128 buckets that TLSH uses during the construction of the hash value quantized to two bits.

As an illustration, let us consider the following prefix of a TLSH hash value, represented in hexadecimal format:

```
T1 B3 C3 09 A5 BC 43 9B 4F CA C3 DB F6 ...
```

The first two character of the TLSH hash (`T1`) is the version number. The version number is followed by the header. The first byte of the header is a checksum, which has the value of hexadecimal `B3` in our example. This is followed by the L value, which is hexadecimal `C3` in this case. Next come the Q1 and Q2 ratios, which are hexadecimal `0` and `9`, respectively, in the example. The remaining bytes are the binary representations of the buckets turned into hexadecimal numbers. As each bucket value is represented by two bits, the next hexadecimal number `A`, in the example, encodes the 2-bit values 10 and 10 of the first two buckets. The same way, the next hexadecimal number `5` encodes the 2-bit values 01 and 01 of the next two buckets, etc.

We can measure the similarity of two inputs by comparing their TLSH hash values with a comparison algorithm [25]. The output of the comparison algorithm is an integer number. The minimum value of this number is 0, which means that the two files of the compared hashes are almost identical. A higher score should represent that there are more differences between the inputs. This algorithm calculates the similarity value of the two given TLSH hash with various weighting. E.g. differences in hash value's header are taken into account with a larger weight than the differences in the hash value's body. For more details on the calculation of TLSH values and TLSH differences, we refer the reader to [25].

It is important to note that the similarity hash functions and comparison algorithms operate only on raw byte sequences, therefore they are suitable for capturing static byte level similarity of binary files, but nothing more.

2.2 SIMBIoTA

SIMBIoTA is an IoT malware detection solution that was proposed in [34]. It saves storage, memory, computation, and bandwidth, which resources are constrained in IoT field. In a certain sense, SIMBIoTA is a hybrid solution, it combines preferably the properties of signature-based and cloud-based solutions.

We can see the high-level architecture of SIMBIoTA in Figure 2.1. It consists of 3 major components: IoT device (i.e. client), backend (i.e. server), and intelligence network. The intelligence network (e.g. honeypot farms, commercial malware feeds, and public malware repositories) provides malware samples for the malware database of the backend. SIMBIoTA is similar to the signature-based approach, but it uses TLSH hash values instead of signatures. SIMBIoTA is also similar to the cloud-based approach, because the backend does the resource-intensive tasks. The backend processes the malware samples and

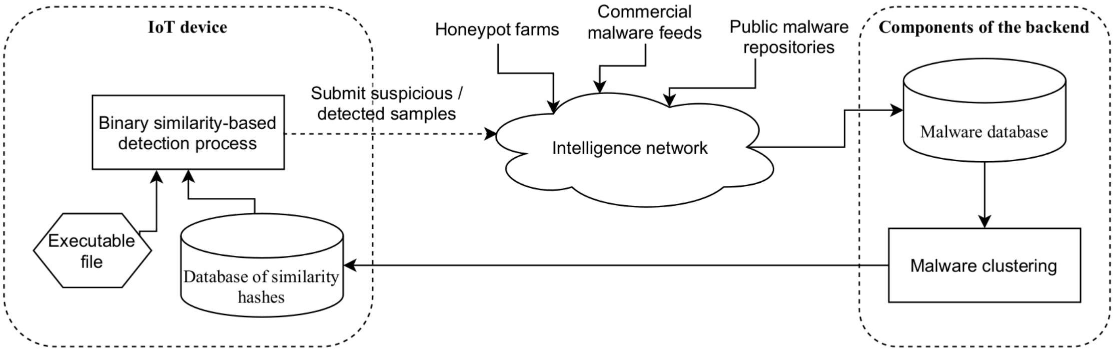
IoT device
Binary similarity-based detection process
Executable file
Database of similarity hashes
Submit suspicious / detected samples
Honeypot farms
Commercial malware feeds
Public malware repositories
Intelligence network
Components of the backend
Malware database
Malware clustering

Figure 2.1: Architecture of SIMBIoTA.

it creates a representative subset of TLSH hashes that is pushed to the IoT device. The IoT device uses this small database of TLSH hashes, and it runs a lightweight algorithm to detect malware, based on binary similarity.

Similarity hashes are a good alternative to signatures in IoT malware detection. Firstly, the similarity hash used by SIMBIoTA (i.e. the TLSH hash) is represented in a very short sequence of bytes (36 bytes). Secondly, based on binary similarity property, one hash can fully represent a group of malware. Hence, all malware samples on the backend can be covered with a relatively small database on the IoT device. In addition, computing similarity hashes does not require manual work of experts, but it can be completely automated. Furthermore, another advantage of similarity hashes is their short calculation time. Processing a single file (i.e. hash generation time + hash comparison time) takes only a few milliseconds even on CPU constrained devices.

2.2.1 Malware analysis at the backend

The backend database of SIMBIoTA is built from the malware samples collected from the intelligence network. Typically, thousands of samples are collected each day. The IoT device could not handle this number of TLSH hashes, so the backend can transmit only a few TLSH hash values. These TLSH hash values form a representative subset that represents the whole backend malware database.

We can imagine the malware database as a graph, where nodes are the malware samples, and two nodes are connected if the TLSH similarity score of their samples is below a selected threshold[1]. More precisely, the mentioned representative subset of samples must form a dominating set[2] for the imagined graph. SIMBIoTA uses a simple greedy algorithm for constructing the dominating set: if a new sample received by the backend is not similar to any of the samples in the current dominating set, it adds the new sample to the dominating set, otherwise it moves on to the next new sample.

Based on the above SIMBIoTA can create the representative subset of the malware database and it can extend if it is necessary. The backend sends the TLSH hashes of

[1] In case of SIMBIoTA, 40 is used as threshold value, which was selected by extensive empirical analysis described in [10].

[2] A dominating set for a graph $G = (V, E)$ is a subset D of V such that every vertex not in D is adjacent to at least one member of D. In graph theory minimal dominating set problem is a classical NP-complete decision problem. Therefore it is believed that there may be no efficient algorithm that finds a smallest dominating set for all graphs.

this dominating set to the client. If the dominating set on the server side is expanding, then the server informs the client with the updates.

2.2.2 Detection process on the IoT device

On IoT device the detection process is called before executing a file. If the TLSH distance of the given file's TLSH hash and one of the hashes in the dominating set is below the selected threshold, then it is considered as a malicious file. According to the client's needs and possibilities, it can further customize his policy of use. SIMBIoTA's client has a great advantage with respect to other cloud-based malware detection system's clients, because with its local database it can operate even if the backend is unavailable.

2.3 SIMBIoTA-ML

SIMBIoTA-ML was proposed in [26]. The purpose of SIMBIoTA-ML is to improve the malware detection capability of SIMBIoTA with machine learning. In order to do so, SIMBIoTA-ML replaces the dominating set construction by machine learning. The modified architecture is shown in Figure 2.2. On embedded IoT device, the database of TLSH hash values is replaced with a machine learning model. Hence, in place of the lightweight detecting algorithm, the client only have to give the examined file to the stored machine learning model for detection. The machine learning model is trained on the backend using both malicious and benign samples. Therefore, SIMBIoTA's intelligence network is extended with sources that also provide benign samples for the backend. Benign samples could be received from IoT software vendors or from public software repositories.

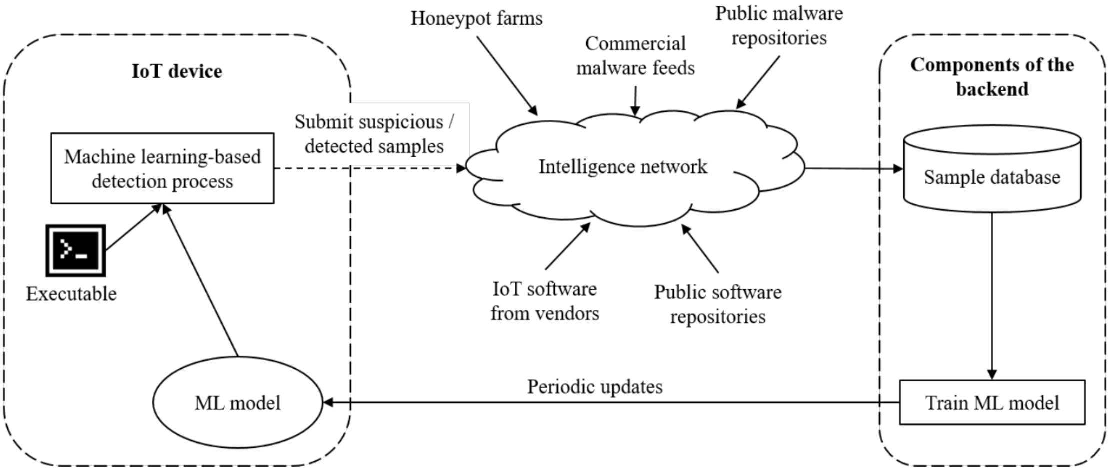
IoT device
Machine learning-based detection process
Executable
ML model
Submit suspicious / detected samples
Honeypot farms
Commercial malware feeds
Public malware repositories
Intelligence network
IoT software from vendors
Public software repositories
Periodic updates
Components of the backend
Sample database
Train ML model

Figure 2.2: Architecture of SIMBIoTA-ML.

2.3.1 Design choices for machine learning

Machine learning models for malware detection must be trained using features that represent important qualities of executable files. In general, features can be derived using static or dynamic program analysis. However, dynamic program analysis (i.e. monitoring a program's execution) is not very economical, and that would be important in case of IoT devices. Therefore, the extraction of feature vectors has to be lightweight and has to be done statically.

TLSH hash values can be considered static features because their calculation involves only the processing of the raw bytes in the program file. The TLSH hash value is transformed into 131 features by splitting the hash value into smaller parts. Specifically, from the header the L value, the Q1 ratio, and the Q2 ratio is taken. The bytes representing buckets are split into bit pairs, which gives 128 2-bit features for the 128 buckets. A random forest classifier is trained on these extracted features. Choosing a random forest classifier is advantageous because it automatically filters non-predictive features [9].

2.4 Performance

Before we show the performance of SIMBIoTA and SIMBIoTA-ML on adversarial examples, we take a look at their results on the unmodified samples. As for the results, we mention only the false and true positive rates of the two systems, because in the context of this paper only these are relevant. In addition to the measurements presented below, one can read about the experiment in more detail in [26].

For the experiments of SIMBIoTA and SIMBIoTA-ML the same data set is used. We discuss the origin and composition of the samples in Section 4.1. For now, it is enough to know that there is a relatively large sample data set. This data set contains malicious and benign files and these files are executables written for either the ARM or the MIPS architecture.

The same evaluation is taken separately on ARM and MIPS samples, however in the following we refer to them together. SIMBIoTA and SIMBIoTA-ML use the same experiment design. The experiment uses samples created between January 1st, 2018 and September 15th, 2019. This time interval is divided into weeks. Both SIMBIoTA and SIMBIoTA-ML receive updates for their detection methods at the beginning of each week.

Malicious samples are organized into weekly batches based on the date they were first seen. 10% of each weakly batch is available to the backend, this training set represents the knowledge provided by the intelligence network. The other 90% is never shown to the backend, this is the test set. The malware detection rate on the test set gives the system's true positive detection rate.

SIMBIoTA-ML requires a balanced data set for training and testing the machine learning model, so the same number of malicious and benign samples per week is needed. However, there is no information about when the benign samples were first seen. Therefore, benign samples are randomly assigned to be part of either the training or test set. Each week, the same number of benign samples are selected from the benign training set as the number of malicious samples in the malware training set. Selected benign samples are available to the backend for training purposes. The test set of benign samples is selected in the same way. Samples of the benign test set are never shown to backend and their evaluation gives the system's false positive detection rate.

The method of assigning benign samples to the training and test sets introduces randomness into the experiment. To eliminate the effects of this randomness on the measurement results, the experiment is repeated 12 times and traditional box plots are used to present the results. The data points of box plots show the results of the 12 runs of the experiment for each week.

2.4.1 True positive detection rate

There are two approaches for measuring the true positive detection rate of SIMBIoTA and SIMBIoTA-ML. The first approach evaluates the test sets of all previous weekly batches, while the second approach takes only the current weakly batch.

Results of the first approach is shown on Figure 2.3. The left-hand side of the figure shows the performance of SIMBIoTA and the right-hand side shows the performance of SIMBIoTA-ML. Both solutions show a learning curve for both the MIPS and the ARM architectures, i.e., their true positive detection rate improves as time passes and more samples are made available to the backend. However, SIMBIoTA-ML consistently outperforms SIMBIoTA by having a true positive detection rate above 95% throughout the measurement.

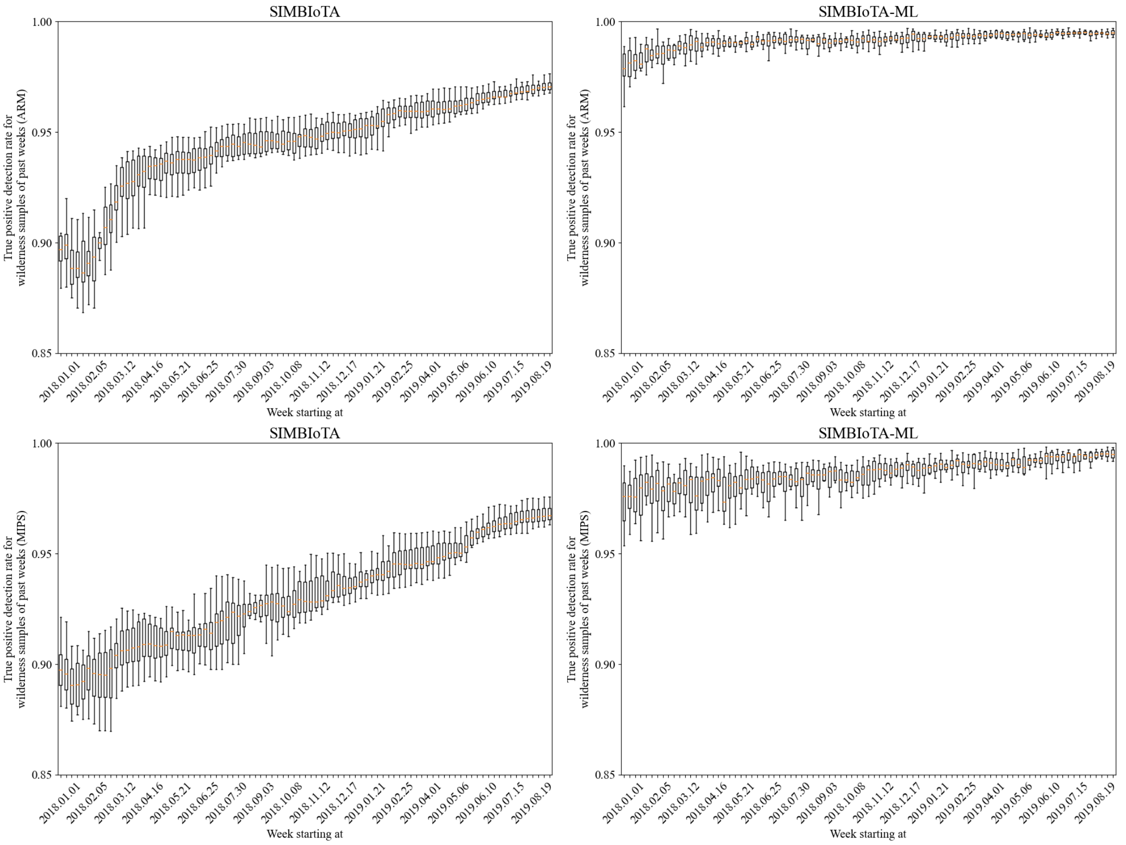

Figure 2.3: Box plot of the true positive detection rate for malware test sets of all previous weeks.

Results of the second approach is shown in Figure 2.4. The left-hand side of the figure shows the performance of SIMBIoTA and the right-hand side shows the performance of SIMBIoTA-ML. SIMBIoTA's performance varies in time and its performance reaches 90-95% only for the second half of the measurement. SIMBIoTA-ML also shows variations in its true positive detection rate but the variation is smaller than that of SIMBIoTA, and performance stays above and around 95% for the majority of the experiment. Therefore, it can be stated that SIMBIoTA-ML outperforms SIMBIoTA in this case as well.

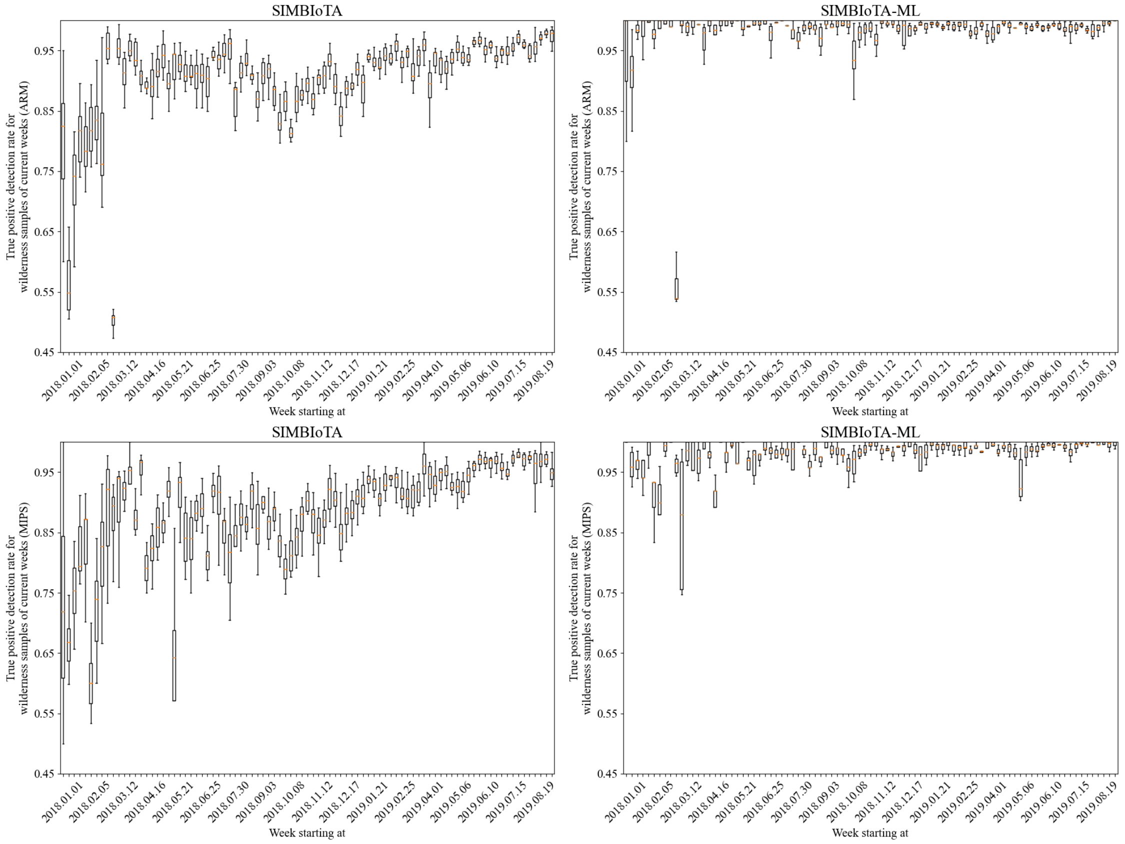

Figure 2.4: Box plot of the true positive detection rate for malware test set of the current week.

2.4.2 False positive detection rate

In order to measure the false positive detection rate of the systems, the following experiment is executed. SIMBIoTA does not use benign samples for learning, thus, all benign samples are submitted to SIMBIoTA for measuring false positive detection rate. In the case of SIMBIoTA-ML, however, the benign test set of actual weakly batch is given to detection process.

As reported in [34] SIMBIoTA did not detect any benign samples as malicious, hence achieved a false positive rate of 0. However, SIMBIoTA-ML has a false positive detection rate 1% on average, as Figure 2.5 shows. This phenomenon is common in machine learning field. Interestingly, MIPS samples show higher false positive detection rate than ARM samples, but it decreases in both cases as time goes on. Overall, it can be stated that while SIMBIoTA-ML's false positive detection rate is higher than SIMBIoTA's, it is still acceptable for malware detection.

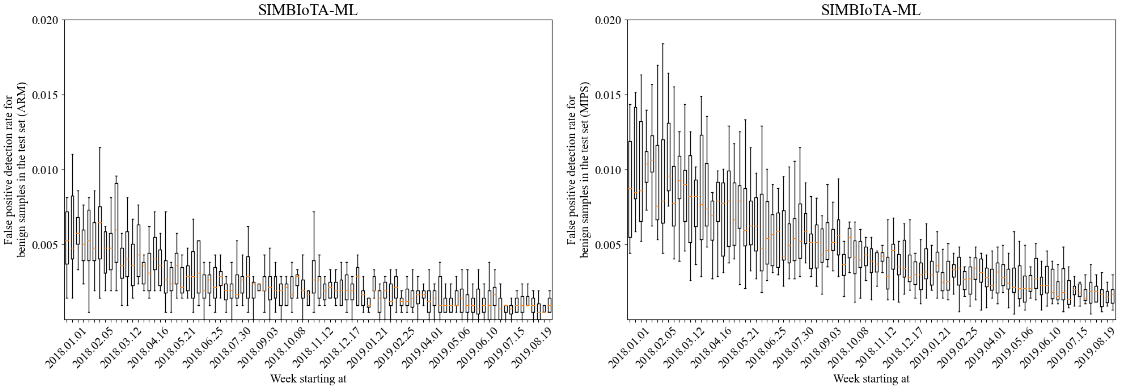

Figure 2.5: Box plot of the false positive detection rate for benign samples in the test set for SIMBIoTA-ML.

Chapter 3

Strategies for creating adversarial examples

We can say based on what we have seen so far that SIMBIoTA and SIMBIoTA-ML appropriately recognize malicious files. However, both of our systems detection mechanism is based on binary similarity. What if the attacker knows this? By using this information, could the attacker increase the chances of evading detection of his malware? Can the attacker achieve that his malware is classified as a benign file by the detecting system? If so, what strategies might he have to do this? In this chapter we are looking for answers to these questions.

3.1 Overview of the adversarial examples problem

Before we delve into answering the previous questions, we take a look at the adversarial examples problem in general. Besides the malware detection context, the adversarial example problem appear in several other machine learning field. A prominent example is image recognition, which creates adversarial example with so-called image perturbation. Here, we add some adversarial perturbation noise to an image, such that no difference is visible for the human eye, but the given machine learning model confidently does a wrong prediction. This popular example on Figure 3.1 shows how a Deep Neural Network (in this case, GoogLeNet) can be misled this way [14].

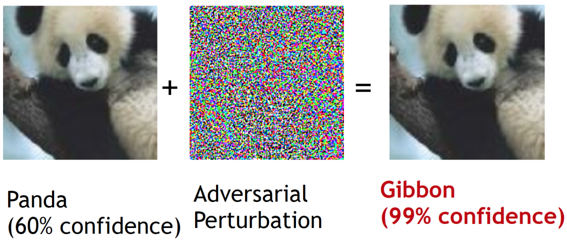

Figure 3.1: Adversarial example against image recognition machine learning model that achieves misclassification of panda as gibbon.

Even though the concept of adversarial examples were first defined the in machine learning field, they can be interpreted in the context of non-ML-based classifiers too. Hence, we can measure the robustness of SIMBIoTA and SIMBIoTA-ML against the same adversarial examples. In general, adversarial examples are those inputs that were specifically designed to cause the given model to make a mistake [8]. In our case this mistake would be the misclassification of malicious samples as benign. The attacker's interest is to create successful adversarial examples that evade malware detection.

In the machine learning field there are two major approaches for creating adversarial examples. We can craft adversarial examples in the feature space, which means we construct feature vectors instead of real inputs that mislead the model. Furthermore, we can create real inputs as adversarial examples by creating completely new samples or by modifying existing ones. In case of computer programs we would like to create adversarial examples that not only mislead the classifier but that remain executable too. Since there is no guarantee that an input reconstructed from a feature vector would be a meaningful computer program, constructing in feature space is excluded, so we have to create real inputs. Malware files are also computer programs, however creating a brand-new malware can be expensive, therefore, it looks more reasonable to modify existing ones.

3.2 Criteria

To answer the questions asked at the beginning of the chapter, we have to think with the head of the attacker. Firstly, we assume that we have already a fancy malware and we do not want to spoil its functionality. The only problem is that SIMBIoTA-ML, but even SIMBIoTA, indicates correctly that it is malicious. However, we know that the similarity hash generation and comparison algorithms (which are also used by our detection systems) do not take into account the format of the inputs, they only consider raw sequences of bytes. Furthermore, we can even find out that this similarity hash function is the TLSH.

Our idea is that we can mislead the detection system if we could manipulate the TLSH hash value of our malware. Knowing the nature of TLSH, to do so, we have to modify the raw binary. Targeted modification of the binary by manipulating the source code without spoiling the original functionality is not so trivial task[1]. It would be much easier to add a few extra bytes to the end of the binary that actually will never be executed, but will change the TLSH hash value. However, we have to do this expansion of the binary carefully, because too much growth can be noticeable for the defenses system.

We can increase the success rate of our attack in the IoT field, especially against malware detection, with the quantity of the adversarial examples. So the attacker needs economical solutions for creating adversarial examples. Simply adding bytes requires no particular sophistication from the attacker, and it is also economical in terms of resources.

3.3 Overview of strategies

For creating adversarial examples, we developed two strategies. The first one is called Chunker, the second is called Disguiser. These represent two different approaches. In case of Chunker we add to the malware chunks of itself and the goal is to increase the TLSH difference between the malware and the crafted adversarial example. In case of Disguiser we want to hide our malware in a benign file, so the goal is to decrease the TLSH difference between the benign file and the adversarial example. Moreover, these

[1] Techniques like obfuscation preserve the functionality of programs, but changes completely their binary. We do not deal with such techniques, because for SIMBIoTA and SIMBIoTA-ML obfuscated malware would seem totally new malware and their detection performance in this case has already been measured in [26].

strategies are relatively simple, an attacker can easily implement them in a real-world situation.

3.4 Strategy 1: Chunker

As mentioned in Section 2.2, SIMBIoTA uses 40 as TLSH similarity threshold. So, our intuition is that if the TLSH distance between the original malware and our crafted adversarial example is at least 40, SIMBIoTA misclassifies it. Chunker's main idea is to simply add bytes to the end of the malware binary. The question is, how many and what kind of bytes need to be added to reach our goal. If all attached bytes are constant or random, the byte entropy[2] would change and a static analyzer would easily detect it. It seems a reasonable solution to add some chunks from the original malware to itself. With this solution, if we choose properly the chunks, the byte entropy of the modified file will be almost the same as the original.

The exact algorithm of Chunker is the following.

- Select an arbitrary malware binary file.
- Split the raw byte sequence of the given file to 20 equal parts. With this, we get 20 chunks, each is 5% of the original file size.
- Select the chunk with the entropy closest to the entropy of the original file.
- Add the selected chunk to the end of the binary file as many time as it is necessary to reach our goal (i.e., to get TLSH difference large enough between the crafted sample and the original one).

To find out how many chunks are needed to be added, we performed an empirical study. We take randomly 2000 samples from the whole malware data set (for description of whole data set see Subsection 4.1.1). To each malware we add different number of chunks and we calculate the TLSH difference between the original and the modified malware. The result is shown in Figure 3.2. One can see that in most cases, 4 chunks (i.e. 20% of the original malware) are enough to be added for reaching TLSH difference 40.

We can observe many boxplot outliers in Figure 3.2. The reason is that, especially in case of the small malware files (<1kB), there are some special samples, where only a few added bytes can cause a large TLSH difference (>80).

3.5 Strategy 2: Disguiser

The false positive rate of a good malware detection system is as low as possible. This means that only in very few cases a benign file will be classified as a malware. Our idea is that we could mislead the detection system, if we hide a malware inside of a benign file. So, the Disguiser strategy concatenates a benign file to the end of a malware binary.

[2] Measure of disorder or uncertainty in the byte distribution of the given binary file.
$H = -\sum_{i=0}^{255} \frac{n_i}{N} \cdot \log_2(\frac{n_i}{N})$, where N is the length of file in bytes and n_i is the number of occurrences of byte value i.
0 is the minimum value of byte entropy, this occurs when all bytes in the binary have the same value.
8 is the maximum value of byte entropy, this occurs when the byte values are distributed uniformly at random.

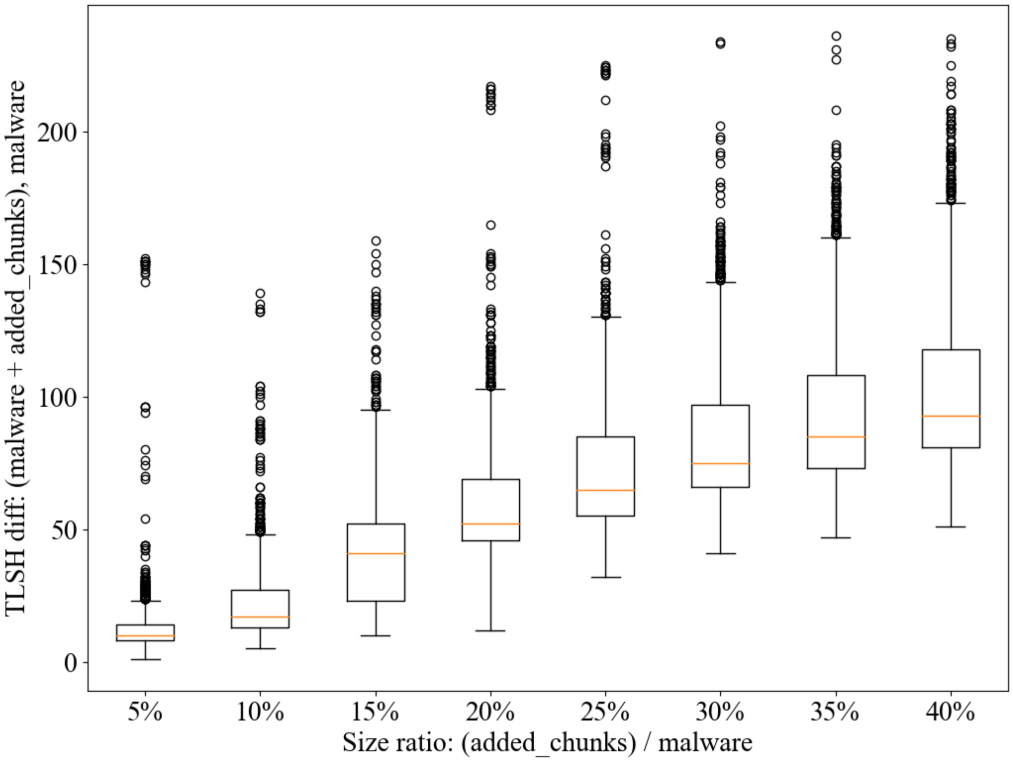

Figure 3.2: TLSH difference between the original sample and the adversarial example created from it by strategy Chunker as a function of the amount of bytes added.

Hence when this file is executed, the malware will run, although the TLSH hash of the file may be determined by the added benign content.

Here, our intuition is that a small malware in a large benign file can be hidden easier, because the TLSH difference between the benign content and the adversarial example will be small. So our goal is to maintain this TLSH difference under the explained threshold of 40.

To find out what is the sufficient size ratio between the benign and the adversarial example to remain under the threshold of 40, we take a little empirical study. We select randomly 100 malware and 300 benign files. We concatenate each benign to each malware, and we calculate the TLSH difference between the hosting benign file and the crafted adversarial example. The result of this measurement is shown on Figure 3.3. On the left side of the Figure 3.3 are represented all adversarial examples. On the right side of the Figure 3.3 (which is a zoomed-in version of the left figure) are represented only those adversarial examples that have a TLSH distance less than 50 from the hosting benign file. We can observe (on the left side of the figure) that in general the higher the size ratio is, the higher the TLSH difference is. We can also observe (on the right side of the figure) that there are quite a few points in the area where the TLSH difference is below 40 and the size ratio is above 1.2. Practically, this means that only the pairs with size ratio below 1.2 have the chance to have TLSH difference below 40.

Using the information described above, the exact algorithm of Disguiser is the following.

1. Select an arbitrary malware binary M.
2. Search a benign file B, so that the size ratio of (M+B) and B is below 1.2.

3. Concatenate B to the end of M.
4. Calculate the TLSH difference between (M+B) and B.
5. If the TLSH difference is below the required threshold we got an adversarial example.
6. Continue with Step 2, as long as there are still unscanned benign files.

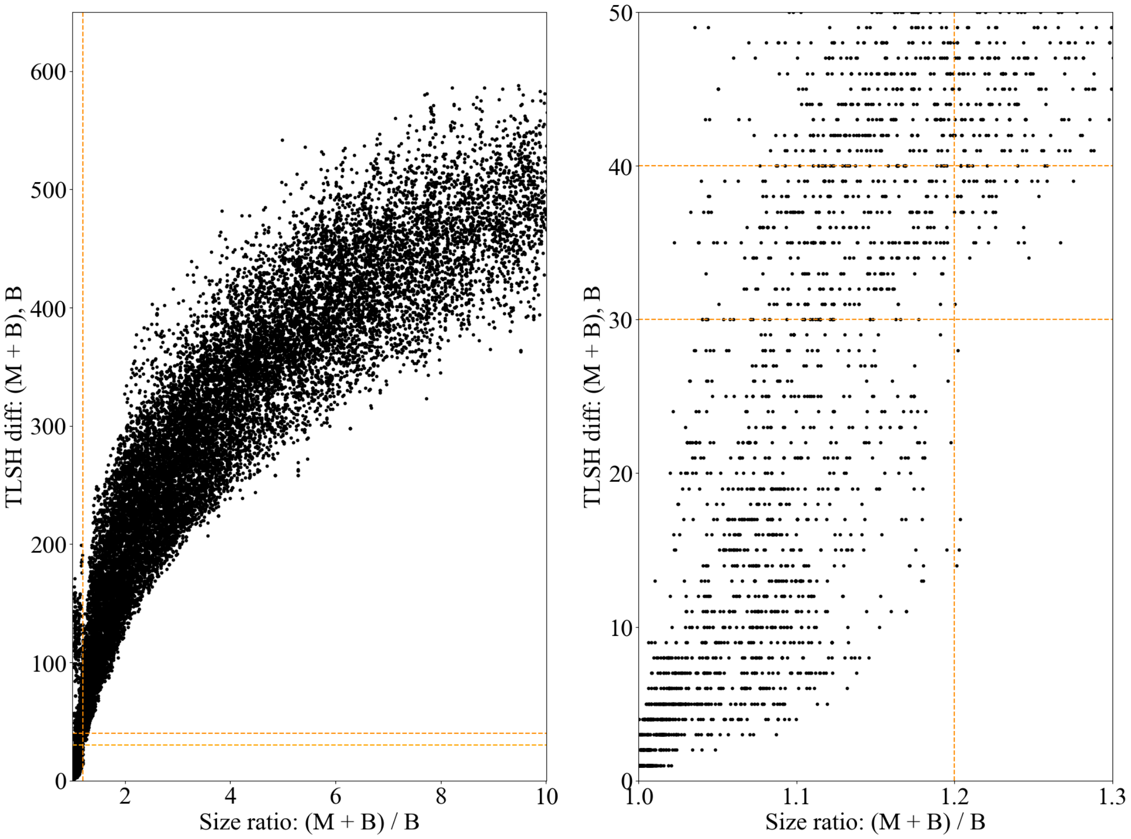

Figure 3.3: Illustration of the effect of the size ratio between the adversarial example and the hosting benign file on the TLSH difference between them when strategy Disguiser is used. The x-axis shows the ratio of the sizes and the y-axis shows the TLSH difference.

Chapter 4

Measurement

In this chapter we present in detail the setup and results of the measurement of the robustness against the crafted adversarial examples in case of SIMBIoTA and SIMBIoTA-ML.

4.1 Setup

Before we look at the results of SIMBIoTA and SIMBIoTA-ML against the set of crafted adversarial examples, we present the background of our experiment. We say a few words about the used malware and benign dataset, we give a more detailed description about the specific parameters, and we show the selection of base material for adversary examples. Finally, we dive into the measurement methodology.

4.1.1 Dataset

We perform all experiments using the same data set as used for the evaluation of SIMBIoTA and SIMBIoTA-ML in [26]. This dataset is called CrySyS-Ukatemi benchmark dataset of IoT malware 2021 (or CUBE-MALIoT-2021 for short). The dataset consists of 29,209 malicious ARM samples and 18,715 malicious MIPS samples, extended with 4,727 benign ARM samples and 9,392 benign MIPS samples. For malicious samples, metadata is also available, which details, among others, the date the sample was first seen in the wild (i.e., submitted to VirusTotal). CUBE-MALIoT-2021 is publicly available[1] for use by the IoT malware research community.

4.1.2 Specific parameters of strategies

In order to get a more accurate picture of the SIMBIoTAs' robustness against adversarial examples we worked out a quite sophisticated parameter set for testing.

We divided the malware sample dataset into three equal parts by size (Small-Medium-Large). The exact intervals are given in Table 4.1.

Disguiser and Chunker have further special parameters. Chunker creates adversarial examples with TLSH difference 40 and 60 from the original malware. Based on the information of Figure 3.2, practically to reach the threshold difference 40, 4 chunks are needed to

[1] `https://github.com/CrySyS/cube-maliot-2021` (accessed: November 22, 2022)

Arch.	S	M	L
ARM	1 - 59,900	59,901 - 120,875	120,876 - 1,942,729
MIPS	1 - 71,508	71,509 - 104,712	104,713 - 2,423,149

Table 4.1: Maximum and minimum limits of small (S), medium (M), and large (L) size intervals of malware dataset (in bytes), in the ARM and MIPS cases.

be added, and to reach the threshold difference 60, 6 chunks are needed to be added. As explained in Section 3.5, Disguiser always uses 0.2 as the maximum of the M/B size ratio (this 20% size increase is still acceptable). Moreover, Disguiser creates adversarial examples with TLSH difference of maximum 40 and 30 from the hosting benign file.

4.1.3 Selection of base material for adversarial examples

The size of the sample data set is relatively large. So, we randomly select a subset as base material for adversarial examples.

For Chunker we select 4000 samples from each size category. When creating adversarial examples from these samples by using the Chunker strategy with 4 or 6 added chunks, the criteria described in Subsection 4.1.2 are not met by all of these selected samples: some of the resulting samples were not far enough in TLSH difference from the original sample. We ignored these samples, and kept only those that satisfy our constraints. The exact numbers of samples obtained in this way are shown in Table 4.2. The data in Table 4.2 indicates that it is more difficult to create adversarial examples from larger malware samples than it is from smaller ones. Furthermore, it is more difficult to reach TLSH threshold 60 than it is to reach TLSH threshold 40. These observations are consistent with intuition.

ARM			
Target TLSH diff.	S	M	L
40	3823	3619	3626
60	3647	3380	3392

MIPS			
Target TLSH diff.	S	M	L
40	3708	3265	2935
60	3288	3065	2593

Table 4.2: Number of adversarial examples created with strategy Chunker from the set of small (S), medium (M), and large (L) samples, with target TLSH difference values 40 and 60, in the ARM and MIPS cases.

In case of Disguiser we randomly select 100 malware from each size category and we pair them with 300 randomly selected benign files. Table 4.3 shows how many pairs meet the criteria defined in Subsection 4.1.2. The data of Table 4.3 shows that it is more difficult to hide a larger malware into a benign file than a smaller one[2].

[2] We could use a more efficient algorithm for pairing malware files with benign files. However, in this study we rather concentrate on the robustness of the detection system against the adversarial examples. For now we showed that is also possible to create sufficient amount of adversarial examples with this simple method.

ARM			
Target TLSH diff.	S	M	L
40	3429	1565	512
30	3868	1298	390

MIPS			
Target TLSH diff.	S	M	L
40	5046	3720	736
30	5055	2370	829

Table 4.3: Number of adversarial examples created with strategy Disguiser from the set of small (S), medium (M), and large (L) samples, with target TLSH difference values 30 and 40, in the ARM and MIPS cases.

4.1.4 Measurement methodology

In order to measure the robustness of SIMBIoTA and SIMBIoTA-ML against the adversarial examples, we need to simulate their behavior. Therefore, we train both SIMBIoTA and SIMBIoTA-ML on 10% of the dataset introduced in Subsection 4.1.1. When we have the trained SIMBIoTA and SIMBIoTA-ML, we can give them an adversarial example, and we can observe whether it is detected as a malware or not. Systematically, we give our adversarial examples (grouped by their parameters) to the detection systems, and we measure their detection accuracy. Similar to the experiment in [26], we repeated all measurements 12 times to eliminate the effects of randomly splitting the dataset into a 10% size training and 90% size testing part. In the next section we will see the results.

4.2 Results

We arrived to the presentation of measurement results. As described in previous sections, we prepared our adversarial examples. Now we see how SIMBIoTA and SIMBIoTA-ML react to these malicious files. Similar to the performance evaluation, we repeat the whole experiment 12 times, and we show the box plot of the accuracy results obtained in the 12 runs in Figures 4.1 and 4.2 for strategy Chunker and strategy Disguiser, respectively. Some of the results are according to expectations, some of them are somewhat surprising.

4.2.1 Results of Chunker

Here we measure the robustness of SIMBIoTA and SIMBIoTA-ML against adversarial examples created by strategy Chunker with the parameter set described in Subsection 4.1.2. In case of Chunker our intuition could be that SIMBIoTA-ML is more robust than SIMBIoTA. This is true, as Figure 4.1 shows, SIMBIoTA-ML has higher detection rate in all cases than that of SIMBIoTA. Moreover, this higher accuracy is actually close to 1 for medium and large size samples, while we can observe a much lower accuracy (but still higher than SIMBIoTA's) for small samples

It is clear that SIMBIoTA can be mislead with this strategy, because the classifier system of SIMBIoTA operates directly with TLSH differences, and Chunker takes advantage of it. It seems that such a big TLSH difference (40-60) does not really matter for SIMBIoTA-ML. Furthermore, in case of SIMBIoTA it is also true that larger TLSH difference causes lower accuracy, as expected.

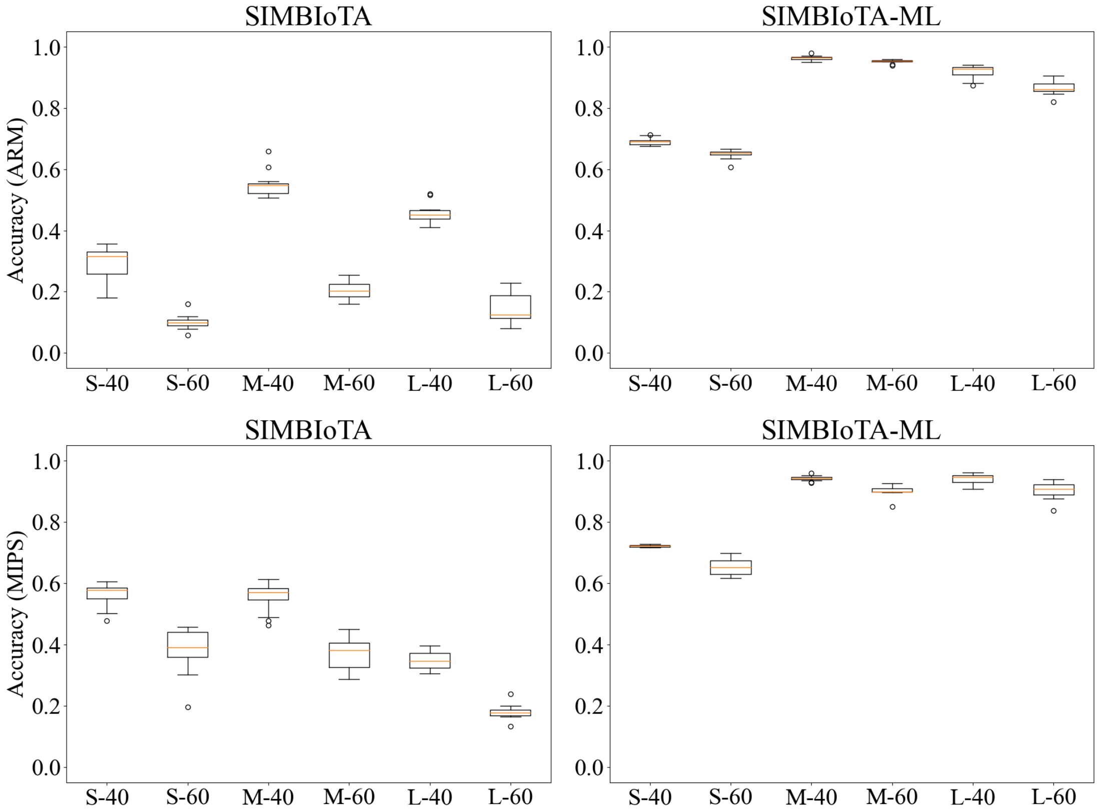

Figure 4.1: Comparison of true positive detection rate of SIMBIoTA and SIMBIoTA-ML against strategy **Chunker**, in the ARM and MIPS cases.

4.2.2 Results of Disguiser

Here we measure the robustness of SIMBIoTA and SIMBIoTA-ML against adversarial examples created by strategy Disguiser with the parameter set described in Subsection 4.1.2. The robustness of the two systems against adversarial examples of Disguiser are surprisingly poor. Basically, these adversarial examples are constructed by concatenating a malware and benign file in such a way that the size of the benign part is much larger than the size of malware part. Thus, the TLSH values of these adversarial examples are more similar to the TLSH values of benign files than to the TLSH values of malware samples. Therefore, SIMBIoTA and SIMBIoTA-ML misclassify these adversarial examples as a benign file.

As Figure 4.2 shows, the accuracy of SIMBIoTA-ML is not exactly 0. This may be because in case of SIMBIoTA-ML there is a small false positive detection rate (see Subsection 2.4.2), where benign files are detected as malware.

4.2.3 Discussion

In this section we have seen the results of the two strategies. We can state that SIMBIoTA-ML is more robust against adversarial examples of Chunker than SIMBIoTA is. Unfortunately, the robustness of SIMBIoTA-ML is not preserved at all against strategy Disguiser.

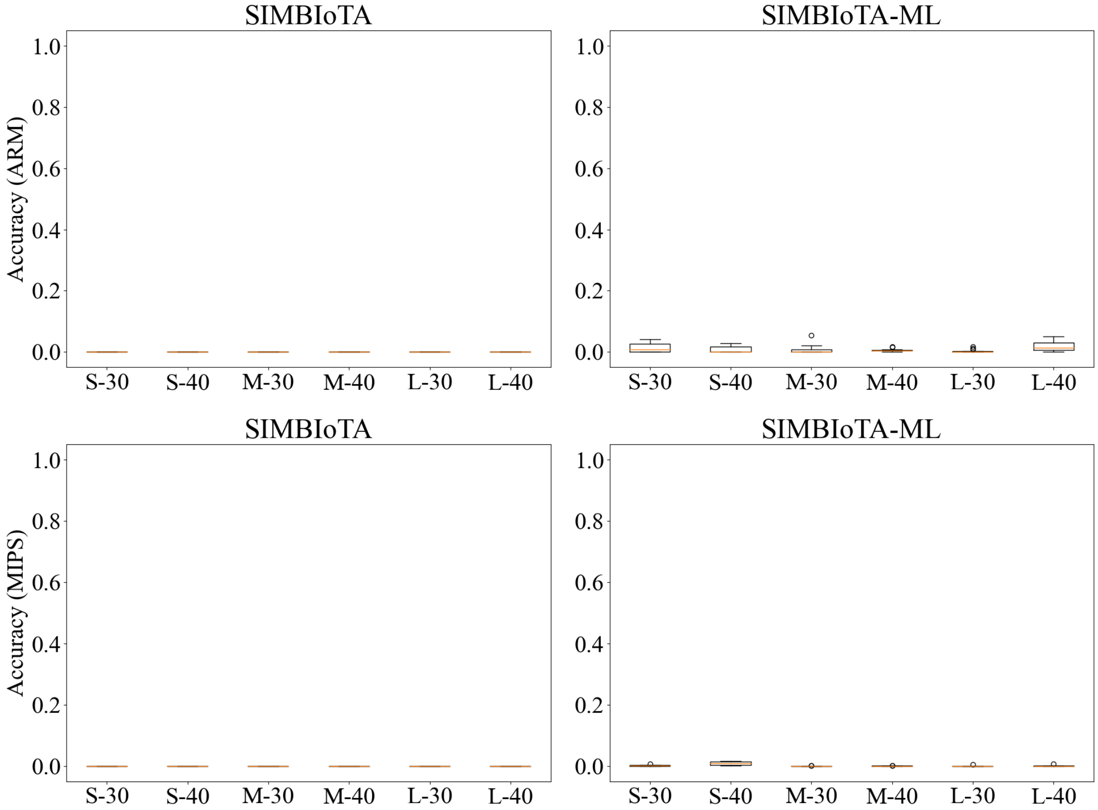

Figure 4.2: Comparison of true positive detection rate of SIMBIoTA and SIMBIoTA-ML against strategy **Disguiser**, in the ARM and MIPS cases.

Moreover, in the perspective of these measurements, there does not seem to be any significant difference in the results in the ARM and the MIPS cases.

As in engineering work in general, these presented solutions also apply some trade-offs, e.g. Chunker can create relatively more adversarial examples than Disguiser, but the adversarial examples of Disguiser are more successful at evading the detection of SIMBIoTA and SIMBIoTA-ML. Furthermore, the Chunker strategy is a trade-off in itself. Chunker appends some bytes to the end of an existing malware, and we want to produce the added content relatively fast, on the other hand, we want the added content to look like some meaningful program code. We consider two approaches: Firstly, adding some constant or random bytes would be simple and fast, but easy to spot with simple static analysis; Secondly, if we add some bytes that have exactly the same byte distribution as the original malware, then it would be hard to spot with simple static analysis, but the generation of these bytes is too slow. The Chunker strategy adds some chunks from the original malware to itself, so it is a trade-off between the two previous alternatives, because in case of Chunker, the appended content looks like the binary of some meaningful program code, furthermore, we can add chunks faster than generating bytes with a specific byte distribution.

Chapter 5

Adversarial training

Unfortunately, no matter how good a malware detection system is, attackers constantly work on methods to evade their detection, this is a cat-and-mouse game. Attackers have many advantages over antivirus companies, one of these is that attackers need only one successful adversarial example construction strategy to reach their goal, but antivirus companies should prepare for all possible adversarial strategies. In this chapter we present a possible solution that antivirus companies could use to increase the robustness of existing malware detection systems against adversarial examples.

In previous chapters, we saw two possible methods that attackers could use to create adversarial examples that evade detection of SIMBIoTA and SIMBIoTA-ML. We showed by measurements that SIMBIoTA-ML is robust against the Chunker strategy, but it can be misled by the Disguiser strategy, while SIMBIoTA has poor robustness against both strategies. To overcome this problem, we propose to antivirus companies that they use SIMBIoTA-ML with adversarial training.

Adversarial training has been used in the image recognition domain to increase the robustness of machine learning-based models against adversarial examples. We adopt this approach in the domain of malware detection and demonstrate its effectiveness. Adversarial training in our case means that that the training set of the malware detector algorithm is extended with samples that are crafted by using the adversarial evasion strategies that we proposed.

We apply adversarial training only on SIMBIoTA-ML, because based on the measurements so far, SIMBIoTA-ML was more robust against the created adversarial examples than SIMBIoTA, so it seems a reasonable decision to improve only the better system.

5.1 Setup

To use adversarial training on SIMBIoTA-ML we have to extend the original training sample set with adversarial examples. Originally, SIMBIoTA-ML is trained on 10% of the malware dataset introduced in Subsection 4.1.1. The samples in the training set represent malware samples known to the antivirus company. Therefore, we construct adversarial examples for adversarial training from malware samples only from the training set, because the antivirus company has knowledge only about these files. After training SIMBIoTA-ML on this extended set of training samples, we test its performance on the original test set and adversarial examples generated from the test set. In the following, SIMBIoTA-ML is referred to as the upgraded SIMBIoTA-ML after adversarial training and the original

SIMBIoTA-ML before adversarial training. Furthermore, we apply adversarial training separately in case of the Chunker and Disguiser strategies.

For adversarial training we have to determine how many adversarial examples should be included in the training set. In case of Chunker this is somewhat simpler than in case of Disguiser, because the Chunker strategy creates one adversarial example from one malware[1]. Therefore, in case of Chunker we use for training all adversarial examples created from malware files from the training set. While in case of Disguiser, the standard deviation of the number of adversarial examples created from a single malware is much larger, because the Disguiser strategy pairs one malware with all possible benign files and selects the pairs that meet the constraints described in Subsection 3.5.

To overcome this problem we created the LooseDisguiser strategy that is similar to Disguiser. The LooseDisguiser strategy, like the Disguiser strategy, pairs benign files with malicious files and creates an adversarial example from a pair if the ratio of the size of the malicious file to the size of the benign file is below 0.2. Unlike the Disguiser strategy, the LooseDisguiser strategy does not consider the TLSH distance between the hosting benign file and the constructed adversarial example. The LooseDisguiser strategy has a so-called multiply factor parameter (instead of TLSH threshold) that define the maximum number of adversarial examples created from a malware. With LooseDisguiser we can create constant[2] number of adversarial examples per malware.

Depending on how many adversarial examples are added to the training set, the accuracy of SIMBIoTA-ML changes on the test adversarial example set and the original test set. In case of Chunker we use for training all adversarial examples created from malware files from the training set. In case of LooseDisguiser, we measure the accuracy of the upgraded SIMBIoTA-ML on the test adversarial example set and on the original test set with different multiply factors. In Figure 5.1 we see the results of this measurement. In this figure the ideal point is (1,1) which means 100% accuracy on adversarial example test set and 100% on the original test set. In case of ARM samples, the point corresponding to multiply factor 4 is the closest to this ideal point, however, in case of MIPS samples this multiply factor is 2. To keep it simple, we choose multiply factor 4 for LooseDisguiser in case of both architectures[3].

The exact numbers of samples obtained in this way are shown in Table 5.1. In case of Disguiser the train adversarial sample set is empty, because we use the adversarial examples of this strategy only for testing the original and the upgraded SIMBIoTA-ML. Similar to the experiment in [26], we repeated the adversarial training 12 times to eliminate the effects of randomly splitting the dataset into a 10% size training and 90% size testing part. Some cells of Table 5.1 contain intervals, rather than specific values, because the number of elements in the train and test sets may differ slightly in the 12 measurements.

[1] In a small number of cases it occurs that the sample created with Chunker strategy does not reach the required TLSH distance from the original malware. In such a case, from this original malware the Chunker strategy cannot create an adversarial example.

[2] Or close to constant, because e.g. at multiply factor 10, we may not find 10 benign files that are five times the size of a very large malware.

[3] A different multiply factor can be chosen depending on which is more important: higher accuracy on the test adversarial example set or higher accuracy on the original test set. In addition, different multiply factors can be selected even for the ARM and MIPS cases.

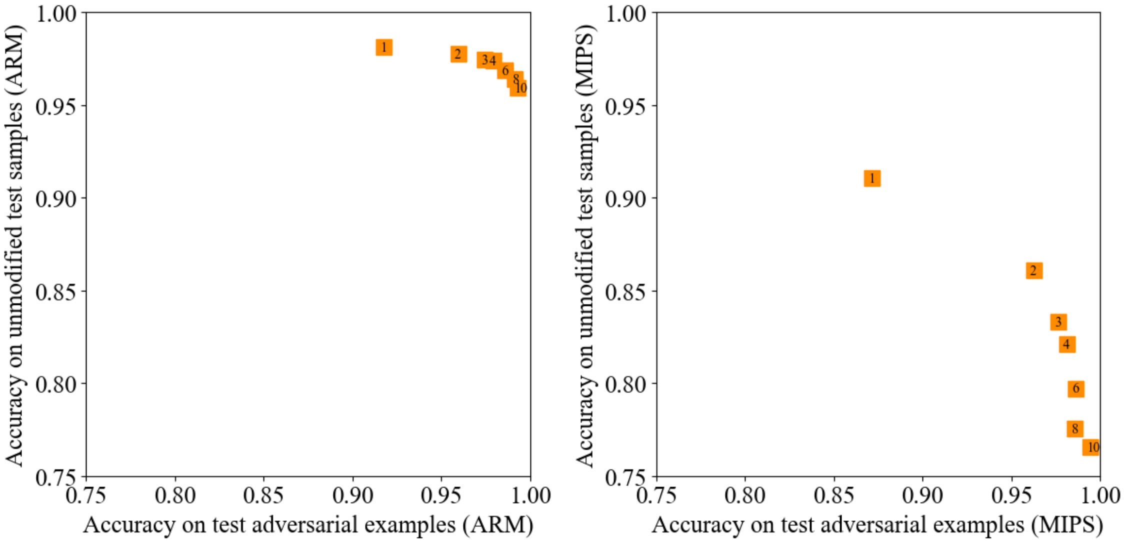

Figure 5.1: The effect of the multiply factor of the LooseDisguiser strategy on the accuracy of SIMBIoTA-ML with adversarial training in the ARM (on the left) and MIPS (on the right) cases. The multiply factors are shown in the small rectangles, where each rectangle corresponds to a given combination of accuracy values.

ARM			
Adversarial sample set	Chunker	LooseDisguiser	Disguiser
Training	2,685 - 2,715	11,644 - 11,680	–
Test	24,297 - 24,327	26,223 - 26,289	1,856 - 3,371
MIPS			
Adversarial sample set	Chunker	LooseDisguiser	Disguiser
Training	1,524 - 1,562	7,460 - 7,476	–
Test	13,869 - 13,907	16,816 - 16,820	3,483 - 4,382

Table 5.1: Number of elements in the train and test adversarial sample set constructed with the Chunker, LooseDisguiser, and Disguiser strategies, in the ARM and MIPS cases.

5.2 Results

In this section we present the results of adversarial training on SIMBIoTA-ML. We measure the detection accuracy of SIMBIoTA-ML trained on the extended training set and show that it remains high both for the original malware samples and for the adversarial samples.

First, we present the results of adversarial training with samples created with the Chunker strategy. On the left side of Figure 5.2 we see that both the original and the upgraded SIMBIoTA-ML have ca. 99% accuracy on the original test set. On the right side we notice that the test adversarial examples of Chunker somewhat mislead the original SIMBIoTA-ML, its accuracy decreases to ca. 93%, while accuracy of the upgraded SIMBIoTA-ML remains high at ca. 99%.

In Subsection 4.2.2 we showed that the original SIMBIoTA-ML can be completely mislead by the Disguiser strategy. In Figure 5.3 we see that the original SIMBIoTA-ML can be

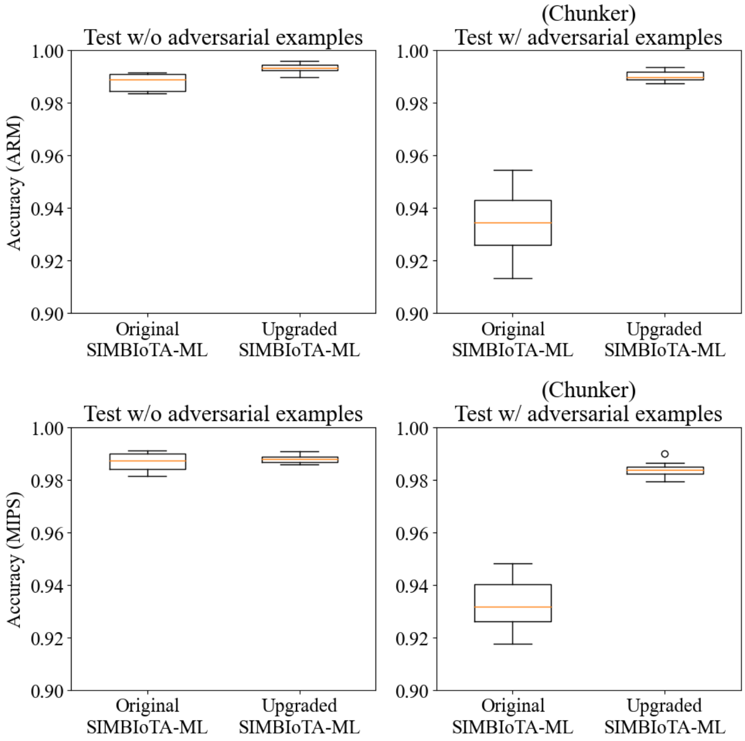

Figure 5.2: Comparison of the accuracy of the original and the upgraded SIMBIoTA-ML on the original test sample set, and on the adversarial test sample set constructed with Chunker strategy, in the ARM and MIPS cases.

completely mislead by the LooseDisguiser strategy too. After adversarial training with the samples of LooseDisguiser, the upgraded SIMBIoTA-ML has a significantly increased accuracy on the adversarial test set constructed with LooseDisguiser (ca. 97%). Moreover, the upgraded SIMBIoTA-ML, which was trained with the adversarial examples of LooseDisguiser, remains surprisingly robust against adversarial examples of Disguiser too. While the accuracy of SIMBIoTA-ML remarkably increases on adversarial examples after adversarial training, the accuracy of the upgraded SIMBIoTA-ML on the original test sample set is only slightly lower than the original SIMBIoTA-ML's accuracy.

5.3 Discussion

In this chapter we showed that, by using adversarial training, antivirus companies can make SIMBIoTA-ML more robust against previously presented adversarial evasion techniques. SIMBIoTA-ML is not only a theoretical solution, but can also be used in industry. We strived for realistic adversarial training methodology that can be used in real-life situations. Therefore, we constructed adversarial examples for adversarial training from malware samples only from the original training set, because the antivirus company has knowledge only about these files. One may notice that in previous chapters we did not consider the test/train set when we selected malware samples for creating adversarial examples. This is because in the previous chapters we looked at the malware detection evasion scenario from the attacker's perspective. The attacker does not know the malware database of the antivirus company, he can construct adversarial examples from all malware samples that he knows about.

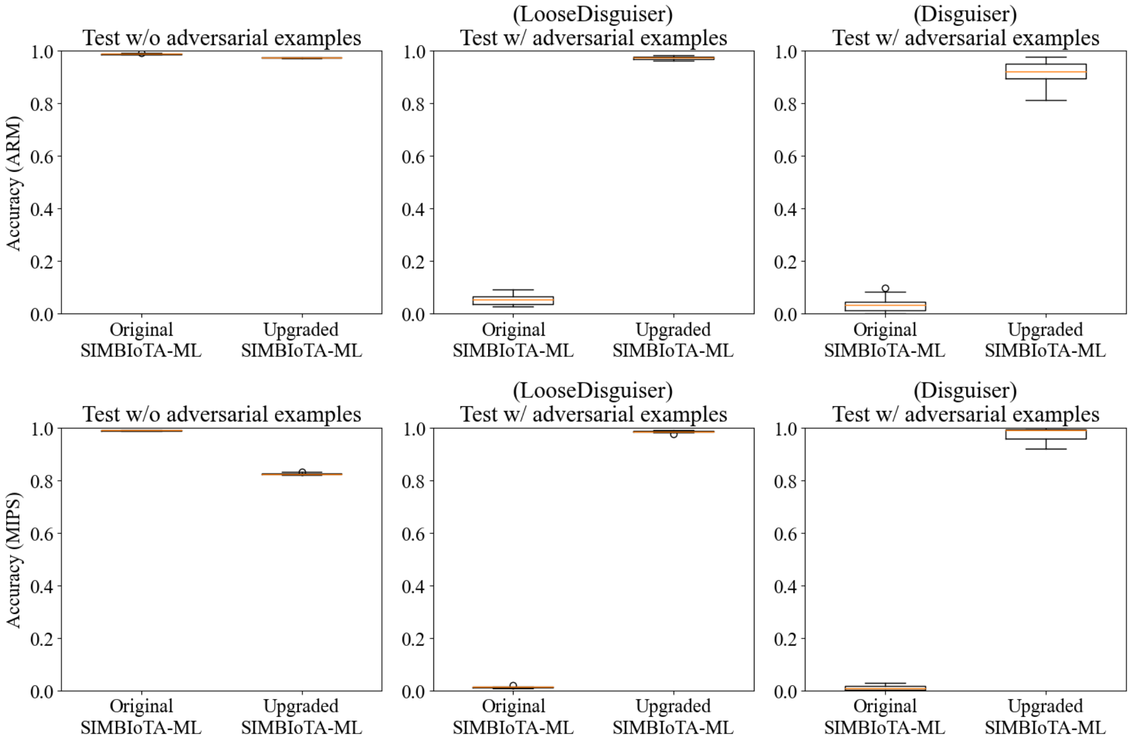

Figure 5.3: Comparison of the accuracy of the original and the upgraded SIMBIoTA-ML on the original test sample set, and on the adversarial test sample set constructed with LooseDisguiser strategy, and on the adversarial sample set constructed with Disguiser strategy, in the ARM and MIPS cases.

The antivirus company does not necessarily know the exact algorithm of the attacker, in fact, most of the time this is the case. A good example of this is the presented scenario, where the antivirus company uses LooseDisguiser for training, but the attacker uses the more powerful Disguiser strategy. Nonetheless, the upgraded SIMBIoTA-ML, which was trained with the adversarial examples of LooseDisguiser, is surprisingly robust against adversarial examples of Disguiser too.

The price that we have to pay for this remarkable robustness is the increased training time and the increased size of the detection model, however, we argue that both are bearable in practice. In Section 5.1 we saw that adversarial training requires an extended training sample set. In machine learning, usually, an increased training set comes with increased training time and increased model size. This time, the increased training time is not critical, because in a real-life situation, similar to the original training of SIMBIoTA-ML, the adversarial training would be performed on the backend (see Subsection 2.3). Only the updated detection model is sent to the resource-constrained IoT device, however, the size of the updated model is larger than the size of the original model because the training set is twice the size of the original sample set for Chunker and 5 times for Disguiser. In Figure 5.4 we see that the price of this significant robustness is ca. 10% model size increase in case of Chunker, and ca. 20% in case of Disguiser, which is acceptable.

In Subsection 2.3.1 we mentioned that SIMBIoTA-ML uses a random forest classifier [9], which also need to be configured. Specifically, the number of decision trees that make up the random forest has to be specified. Similar to the original SIMBIoTA-ML measurements

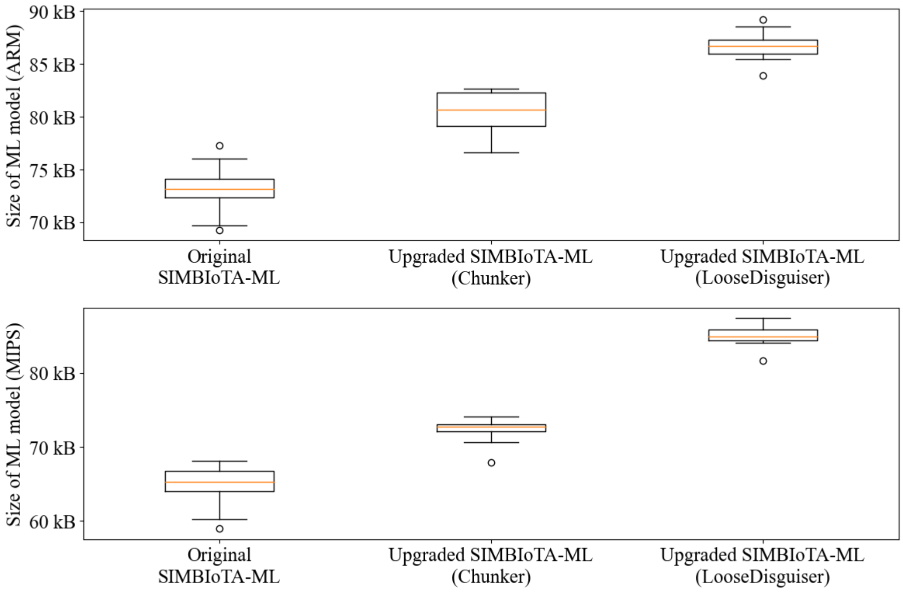

Figure 5.4: Comparison of the size of ML models trained without adversarial examples and trained with adversarial examples, in the ARM and MIPS cases.

in [26], we set the number of decision trees to 10, which give a good trade-off between the detection capability of the machine learning model and the memory required to apply the model on the embedded IoT device. Another important random forest parameter is the maximum depth of the decision trees. In the case of the original SIMBIoTA-ML measurements in [26], the maximum depth of the decision trees of the random forest was not limited. However, the upgraded SIMBIoTA-ML requires more memory on the embedded IoT device due to the extended training data set. Therefore, we set the maximum depth of random forest's decision trees to 6, thus reducing the size of the model to half of the original in [26], while its detection capability practically remained the same.

For adversarial training of SIMBIoTA-ML, similar to [26], our implementation for the random forest classifier uses the scikit-learn[4] Python module. In order to measure the amount of storage necessary to hold the model, similar to [26], we used the pickle[5] module to transform the Python object into a byte string that could be written to disk and later reloaded into memory. We then calculated the length of the byte string to get the number of bytes necessary to represent the object. We understand that there are more efficient ways to serialize a random forest model than using the pickle module. So, in practice, the representation of the model may require even smaller amount of memory on the IoT device than the amount we observed in our measurements (which is already acceptably small anyway).

[4] `https://scikit-learn.org/stable/` (accessed: November 22, 2022)

[5] `https://docs.python.org/3/library/pickle.html` (accessed: November 22, 2022)

Chapter 6

Related work

In this chapter we present a comprehensive overview of the state of the art in the related field. SIMBIoTA-ML uses machine learning for malware detections, hence we show other works that use ML-based solutions. The two major topic of this paper is creating adversarial examples, and analyzing robustness of malware detection systems against these adversarial examples. Therefore, we mention some articles that address these topics from other perspectives too.

6.1 ML-based (IoT) malware detection

As we mention in Chapter 1, traditional (i.e. signature-based and heuristic solutions) malware detection systems could have scalability problems. In addition, traditional systems use only static properties of malware files for detection, hence with special techniques (e.g. obfuscation) they can be deceived.

Unlike traditional solutions, ML-based malware detection can be highly automated [36, 35, 13]. Furthermore, they use static and dynamic program analysis for extracting the required feature vectors [29]. Hence, their detection capabilities are better than that of traditional malware detection approaches. Feature vectors can be extracted from different sources, including the samples' instructions [12, 33], their control-flow [2], invoked API functions and system calls [1, 28], grey scale images of binaries [21], strings [20], and messages sent over network [24, 15].

In addition, solutions that combine machine learning with cloud-based approach scale well and can be applied also in the IoT field [32, 19]. This construction is advantageous for resource constrained IoT devices, because resource heavy calculation and processing can be passed to cloud, and only a lightweight algorithm is needed on client side. They can use different ML models, including convolutional neural networks [30], recurrent neural networks [16], random forest classifiers [33], fuzzy and fast fuzzy pattern trees [12].

6.2 Adversarial examples and robustness analysis

For making machine learning models more accurate and reliable we have to prepare them against adversarial examples. Therefore, this is an actively researched area with a rich and diverse literature.

Originally, ML-based image recognition was the first scientific field where adversarial attacks were applied. As an example, consider Figure 3.1, where some perturbation is added to the original image for the purpose of deception of the given ML classifier. Practically, this concept can be applied to ML-based malware detection field too, because also in case of malware files we usually want some perturbation on the original binary. In addition, as the functionality of image remains the same (i.e. in Figure 3.1 we still recognize the panda), usually we want to preserve the functionality of malware too.

Based on the attacker's knowledge on the targeted ML detection system we can distinguish two type of attacker models [6]. In the white-box model, the attacker has a comprehensive idea of the ML model, he knows the exact training data and concrete model parameters. Moreover, there exists the black-box model, when the attacker has knowledge only about the input and output of the model. Our presented strategies (Chunker & Disguiser) rather follow a grey-box attacker model, because they do not know about concrete model parameters, but they take into account the fact that SIMBIoTA and SIMBIoTA-ML use similarity based hashes. So in our case, the attacker has partial information about the detection systems.

There are many different approaches for adversarial attacks also in the context of malware detection [6]. From these approaches we can highlight *append* and *slack* attacks [31] for their simplicity. Append attacks generate bytes and add them to the end of malware binary. Slack attacks add or modify bytes in slack regions of a binary, which are gaps between neighboring sections of an executable file. Our presented strategies (Chunker & Disguiser) resemble the previously mentioned append attack. There are other solutions for generating and appending bytes to the end of a binary, including gradient-based approach [22, 23].

Another more advanced technique is program obfuscation, which can change the binary representation of a program while preserving its functionality [27]. In order to do so, ML solutions can be used, including reinforcement learning-based approaches [4, 3], Generative Adversarial Networks (GAN)[18] and Recurrent Neural Networks (RNN) [17]. Obfuscating existing malware samples may be a successful strategy, but we do not use it, because from the perspective of SIMBIoTA and SIMBIoTA-ML, obfuscated samples appear to be new malware, as their binary representations can be completely different from those of the original samples from which they were created. In other words, obfuscated samples are considered new malware by SIMBIoTA and SIMBIoTA-ML, and their detection performance on them has already been measured in [26].

Chapter 7

Conclusion

We have reached the end of our report. It is time to summarize what has been discussed so far. Moreover, we share some of our ideas for possible future work.

Firstly, we presented the problem of IoT malware detection with its challenges and highlighting its importance. We introduced two recent similarity-based IoT malware detection solutions, SIMBIoTA and SIMBIoTA-ML. We were interested in how these systems, which perform well initially, behave against adversarial examples. Therefore, we constructed two different strategies for creating adversarial examples from existing malware files: Chunker and Disguiser. Basically, both strategies append a few bytes to the end of the malware, so they are relatively simple methods. Our measurement study shows that in case of Chunker, SIMBIoTA-ML has higher detection rate than SIMBIoTA, while in case of Disguiser, both detection system have poor performance.

To overcome this problem we used the adversarial training concept to increase the robustness of SIMBIoTA-ML against the presented adversarial evasion strategies. For adversarial training, we extended the training sample set of SIMBIoTA-ML with adversarial examples constructed by the adversarial evasion strategies. After adversarial training, the upgraded SIMBIoTA-ML became much more robust against samples of Chunker and Disguiser. Indeed, the upgraded SIMBIoTA-ML detects the adversarial examples with practically the same accuracy as the original samples. The price that we have to pay for this remarkable robustness was the increased training time and the increased size of the detection model, however, we showed that both are bearable in practice.

As described in previous chapters, we have achieved significant results. However, this project is far from being finished. Our future plan is to optimize our existing adversarial strategies. Furthermore, we would like to create other, even more successful methods for creating adversarial examples.

Based on experiences, especially in case of the small malware files (<1kB), there are some special samples, where only a few added bytes can cause a large TLSH difference (>80). An interesting study would be to examine these special malware files, represented by the boxplot outliers in Figure 3.2.

SIMBIoTA-ML uses a random forest classifier to detect malware samples [26]. In Chapter 5 we saw that appropriately chosen random forest parameters drastically decrease the size of the model, which is critical in embedded IoT environment. The presented random forest parameter configuration is good enough, but it cannot be ruled out that a better one exists. We want to answer this question too.

Acknowledgements

The presented work was carried out within the SETIT Project (2018-1.2.1-NKP-2018-00004), which has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the 2018-1.2.1-NKP funding scheme. The research was also supported by the Ministry of Innovation and Technology NRDI Off within the framework of the Artificial Intelligence National Laboratory Program. The author would also like to acknowledge the help received from Dr. Dorottya Futóné Papp in the interpretation of existing implementations of SIMBIoTA and SIMBIoTA-ML, which were previously created in the context of the SETIT Project.

Bibliography

[1] Muhamed Fauzi Bin Abbas and Thambipillai Srikanthan. Low-complexity signature-based malware detection for iot devices. In Lynn Batten, Dong Seong Kim, Xuyun Zhang, and Gang Li, editors, *Applications and Techniques in Information Security*, pages 181–189, Singapore, 2017. Springer Singapore. ISBN 978-981-10-5421-1.

[2] Hisham Alasmary, Aminollah Khormali, Afsah Anwar, Jeman Park, Jinchun Choi, Ahmed Abusnaina, Amro Awad, Daehun Nyang, and Aziz Mohaisen. Analyzing and detecting emerging internet of things malware: A graph-based approach. *IEEE Internet of Things Journal*, 6(5):8977–8988, 2019.

[3] H. Anderson, Anant Kharkar, Bobby Filar, David Evans, and Phil Roth. Learning to evade static pe machine learning malware models via reinforcement learning. *ArXiv*, abs/1801.08917, 2018.

[4] Hyrum S. Anderson, Anant Kharkar, Bobby Filar, and Phil Roth. Evading machine learning malware detection. *BlackHat USA*, 2017.

[5] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the mirai botnet. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1093–1110, Vancouver, BC, August 2017. USENIX Association. ISBN 978-1-931971-40-9.

[6] Kshitiz Aryal, Maanak Gupta, and Mahmoud Abdelsalam. A survey on adversarial attacks for malware analysis. *ArXiv*, abs/2111.08223, 2021.

[7] Ömer Aslan and Refik Samet. A comprehensive review on malware detection approaches. *IEEE Access*, 8:6249–6271, 2020.

[8] Marco Barreno, Blaine Nelson, Anthony D. Joseph, and J. D. Tygar. The security of machine learning. *Machine Learning*, 81(2):121–148, Nov 2010. ISSN 1573-0565.

[9] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001. ISSN 1573-0565.

[10] Levente Buttyán. SIMBIoTA++: Improved similarity-based iot malware detection. 2022.

[11] Emanuele Cozzi, Pierre-Antoine Vervier, Matteo Dell'Amico, Yun Shen, Leyla Bilge, and Davide Balzarotti. The tangled genealogy of iot malware. In *Annual Computer Security Applications Conference*, ACSAC '20, page 1–16, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450388580.

[12] Ensieh Modiri Dovom, Amin Azmoodeh, Ali Dehghantanha, David Ellis Newton, Reza M. Parizi, and Hadis Karimipour. Fuzzy pattern tree for edge malware detection and categorization in iot. *Journal of Systems Architecture*, 97:1–7, 2019. ISSN 1383-7621.

[13] Daniel Gibert, Carles Mateu, and Jordi Planes. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications*, 153:102526, 2020. ISSN 1084-8045.

[14] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2014.

[15] Mohit Goyal, Ipsit Sahoo, and G. Geethakumari. Http botnet detection in iot devices using network traffic analysis. In *2019 International Conference on Recent Advances in Energy-efficient Computing and Communication (ICRAECC)*, pages 1–6, 2019.

[16] Hamed HaddadPajouh, Ali Dehghantanha, Raouf Khayami, and Kim-Kwang Raymond Choo. A deep recurrent neural network based approach for internet of things malware threat hunting. *Future Generation Computer Systems*, 85:88–96, 2018. ISSN 0167-739X.

[17] Weiwei Hu and Ying Tan. Black-box attacks against RNN based malware detection algorithms. *CoRR*, abs/1705.08131, 2017.

[18] Weiwei Hu and Ying Tan. Generating adversarial malware examples for black-box attacks based on GAN. *CoRR*, abs/1702.05983, 2017.

[19] Fatima Hussain, Rasheed Hussain, Syed Ali Hassan, and Ekram Hossain. Machine learning in iot security: Current solutions and future challenges. *IEEE Communications Surveys & Tutorials*, 22(3):1686–1721, 2020.

[20] Chanwoong Hwang, Junho Hwang, Jin Kwak, and Taejin Lee. Platform-independent malware analysis applicable to windows and linux environments. *Electronics*, 9(5), 2020. ISSN 2079-9292.

[21] Evanson Mwangi Karanja, Shedden Masupe, and Mandu Gasennelwe Jeffrey. Analysis of internet of things malware using image texture features and machine learning techniques. *Internet of Things*, 9:100153, 2020. ISSN 2542-6605.

[22] Bojan Kolosnjaji, Ambra Demontis, Battista Biggio, Davide Maiorca, Giorgio Giacinto, Claudia Eckert, and Fabio Roli. Adversarial malware binaries: Evading deep learning for malware detection in executables. In *2018 26th European Signal Processing Conference (EUSIPCO)*, pages 533–537, 2018.

[23] Felix Kreuk, Assi Barak, Shir Aviv-Reuven, Moran Baruch, Benny Pinkas, and Joseph Keshet. Adversarial examples on discrete sequences for beating whole-binary malware detection. *CoRR*, abs/1802.04528, 2018.

[24] Yair Meidan, Michael Bohadana, Yael Mathov, Yisroel Mirsky, Asaf Shabtai, Dominik Breitenbacher, and Yuval Elovici. N-baiot—network-based detection of iot botnet attacks using deep autoencoders. *IEEE Pervasive Computing*, 17:12–22, 07 2018.

[25] Jonathan Oliver, Chun Cheng, and Yanggui Chen. Tlsh – a locality sensitive hash. In *2013 Fourth Cybercrime and Trustworthy Computing Workshop*, pages 7–13, 2013.

[26] Dorottya Papp, Gergely Ács, Roland Nagy, and Levente Buttyán. SIMBIoTA-ML: Light-weight, machine learning-based malware detection for embedded iot devices. In *Proceedings of the 7th International Conference on Internet of Things, Big Data and Security - IoTBDS,*, pages 55–66. INSTICC, SciTePress, 2022. ISBN 978-989-758-564-7.

[27] Daniel Park, Haidar Khan, and Bülent Yener. Generation & evaluation of adversarial examples for malware obfuscation. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pages 1283–1290, 2019.

[28] M. Shobana and S. Poonkuzhali. A novel approach to detect iot malware by system calls using deep learning techniques. In *2020 International Conference on Innovative Trends in Information Technology (ICITIIT)*, pages 1–5, 2020.

[29] Silvia Wahballa Soliman, Mohammed Ali Sobh, and Ayman M. Bahaa-Eldin. Taxonomy of malware analysis in the iot. In *2017 12th International Conference on Computer Engineering and Systems (ICCES)*, pages 519–529, 2017.

[30] Jiawei Su, Danilo Vargas Vasconcellos, Sanjiva Prasad, Daniele Sgandurra, Yaokai Feng, and Kouichi 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.

[31] Octavian Suciu, Scott E. Coull, and Jeffrey Johns. Exploring adversarial examples in malware detection. *2019 IEEE Security and Privacy Workshops (SPW)*, pages 8–14, 2019.

[32] Hao Sun, Xiaofeng Wang, Rajkumar Buyya, and Jinshu Su. Cloudeyes: Cloud-based malware detection with reversible sketch for resource-constrained internet of things iot devices. *Softw. Pract. Exper.*, 47(3):421–441, mar 2017. ISSN 0038-0644.

[33] Hayate Takase, Ryotaro Kobayashi, Masahiko Kato, and Ren Ohmura. A prototype implementation and evaluation of the malware detection mechanism for iot devices using the processor information. *International Journal of Information Security*, 19: 71–81, 2019.

[34] Csongor Tamás, Dorottya Papp, and Levente Buttyán. SIMBIoTA: Similarity-based malware detection on iot devices. In *IoTBDS*, pages 58–69. SCITEPRESS, 2021.

[35] Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. Survey of machine learning techniques for malware analysis. *Computers & Security*, 81:123–147, 2019. ISSN 0167-4048.

[36] Yanfang Ye, Tao Li, Donald Adjeroh, and S. Sitharama Iyengar. A survey on malware detection using data mining techniques. *ACM Comput. Surv.*, 50(3), jun 2017. ISSN 0360-0300.