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

A New Method for Increasing the Robustness of Similarity-based IoT Malware Detection

Scientific Students' Association Report

Author:

József Sándor

Advisors:

Dr. Levente Buttyán
Roland Nagy

2023

Contents

Abstract

Detecting malware targeting IoT devices has became an important challenge with the recent emergence of IoT botnets. SIMBIoTA, a recently proposed similarity-based IoT malware detection method, has good detection performance, while being very resource efficient at the same time, but its robustness against adversarial malware samples has been shown to be rather weak. In this paper, we propose PATRIoTA, a similarity-based IoT malware detection method inspired by SIMBIoTA, but being significantly more robust than SIMBIoTA is. We describe the operation of PATRIoTA, and compare its malware detection performance and robustness against adversarial samples to that of SIMBIoTA. We show that PATRIoTA outperforms SIMBIoTA with respect to both measures. Moreover, in this paper, we also propose a new framework for Efficient COmparisons of Malware detector Performances (ECOMP), and we illustrate its usage by implementing both SIMBIoTA and PATRIoTA in it and comparing their performances. It is important to emphasize that the ECOMP framework is not limited to SIMBIoTA and PATRIoTA, but it is designed to be an easily usable framework, which simplifies and standardizes the performance measurements and comparisons of any malware detection methods.

Kivonat

Az IoT-eszközökre irányuló rosszindulatú szoftverek detektálása fontos kihívássá vált az IoT botnetek közelmúltbeli megjelenésével. A SIMBIoTA egy nemrégiben javasolt hasonlóság-alapú IoT malware-detekciós módszer, mely kiváló detekciós teljesítményt nyújt és nagyon erőforrás-hatékony is, ám a robusztussága az ún. adversarial mintákkal szemben meglehetősen gyengének bizonyult. Ebben a tanulmányban egy új, a SIMBIoTA-nál lényegesen robusztusabb módszert javaslunk, amit a SIMBIoTA ihletett, ezért ezt az új módszert PATRIoTA-nak neveztük el. Leírjuk a PATRIoTA működését, és összehasonlítjuk a malware detekciós teljesítményét és az adversarial mintákkal szembeni robusztusságát a SIMBIoTA-éval. Megmutatjuk, hogy a PATRIoTA mindkét mérőszám tekintetében felülmúlja a SIMBIoTA-t. Ezen túlmenően ebben a tanulmányban egy új keretrendszert is javasolunk a rosszindulatú szoftverek detektálási teljesítményének hatékony összehasonlítására (Efficient COmparisons of Malware Detector Performances, ECOMP), továbbá a SIMBIoTA és a PATRIoTA implementálásával és teljesítményük összehasonlításával illusztráljuk a használatát. Fontos hangsúlyozni, hogy az ECOMP keretrendszer nem korlátozódik a SIMBIoTA-ra és a PATRIoTA-ra, hanem egy könnyen használható keretrendszer, amely egyszerűsíti és szabványosítja a rosszindulatú szoftverek detektálására szolgáló módszerek teljesítménymérését és összehasonlítását.

Chapter 1

Introduction

The expansion of the Internet-of-Things (IoT) is unwavering: the number of installed IoT devices exceeds 15 billion and it is constantly growing[1]. At the same time, the security of IoT devices is notoriously weak [10, 2]. This poses a threat from two aspects: on the one hand, compromised IoT devices can potentially be used to build huge attacking infrastructures (e.g., botnets), with which Internet-based services can be effectively attacked (see e.g. the Mirai botnet and the DDoS attacks launched from it in 2016 [4]); on the other hand, in cyber-physical applications (e.g., industrial IoT systems, self-driving cars), the compromise of IoT devices can lead to physical damage of expensive equipment or even fatal accidents (see e.g. the proof-of-concept attack on a Jeep Cherokee carried out in 2015 [15] and its potential consequences). The problem is so significant that regulatory processes aiming at increasing the security of IoT systems have been initiated in both the US[2] and Europe[3].

An increasingly widespread method for compromising IoT devices at large scale is infecting them with malware (i.e., malicious programs). This is made possible by the fact that IoT devices are essentially embedded computers and malware can be installed on them just like in the case of traditional computers. As a result, several malware families targeting IoT devices have appeared in the past few years (e.g., Mirai, Gafgyt, Tsunami, Hajime). At the same time, traditional anti-virus solutions require too many resources (e.g., storage capacity and CPU cycles), and therefore, they cannot be applied directly on the typically resource-constrained IoT devices. Hence, there is a great demand for efficient and effective IoT malware detection methods.

Gateways placed at the edge between the Internet and the IoT devices deployed in the field are particularly well-positioned for protecting IoT devices against malware [12]. The main reasons are that malware typically spread over the Internet and that resource-constrained IoT devices may not have the means to protect themselves. Edge gateways, on the other hand, have more resources to support malware detection and, thanks to their placement, they can block malware to reach IoT devices. At the same time, malware detection on gateways must be extremely fast for not imposing delays in communications. This essentially excludes the outsourcing of the malware detecion task to some cloud-based backend, and requires performing all operations locally on the edge gateways themselves.

[1] https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/ (accessed on October 17, 2023)

[2] https://www.security.org/blog/california-passes-first-cybersecurity-law-iot/ (accessed on October 17, 2023)

[3] https://digital-strategy.ec.europa.eu/en/library/cyber-resilience-act (accessed on October 17, 2023)

This means that, among other things, edge intelligence should also include effective and efficient IoT malware detection.

A recently proposed similarity-based IoT malware detection method, called SIMBIoTA [28], fits this context perfectly: SIMBIoTA is lightweight and fast, it performs IoT malware detection entirely locally, yet it has a remarkable malware detection capability. However, its robustness against adversarial malware samples has been shown to be rather weak [25]. In the context of malware detection, adversarial samples are meant to be malware samples crafted specifically to evade detection by a given malware detection method. In the case of SIMBIoTA, such adversarial samples can be created easily by appending some extra bytes at the end of existing malware binaries, as shown in [25].

In this paper, we propose PATRIoTA (PArticle TRained IoT Antivirus), a similarity-based IoT malware detection method inspired by SIMBIoTA, but being significantly more robust than SIMBIoTA is. The main idea of PATRIoTA is to split malware samples known to the antivirus provider into multiple fixed-size parts, referred to as *particles*, and to perform the same operations on those particles as the operations performed by SIMBIoTA on entire samples. This means that the antivirus provider builds a similarity graph of known particles, computes its dominating set, and distributes similarity preserving hash values (in our case, TLSH [17] values) corresponding to the particles in the dominating set to the clients, which are the edge gateways in this work. The clients also split any file to be scanned (e.g., a binary extracted from network traffic) into particles, compute the similarity preserving hash values of them, and if a threshold number of those computed hashes are similar to the hashes in the dominating set, then the file is detected as malware.

PATRIoTA is robust against adversarial sample creating strategies that add extra bytes to an existing malware binary, because the sample created in this way will always contain in it the original binary, and, hence, all of its particles, which can be recognized by PATRIoTA despite the presence of the added extra bytes. In addition, PATRIoTA may be robust against even more sophisticated adversarial strategies that keep sizable chunks of the original malware binary intact within the created adversarial sample, as those chunks may result in particles that are similar to the particles of the original sample. Moreover, our measurement results indicate that, besides increased robustness, PATRIoTA also has better malware detection capabilities than SIMBIoTA has.

Moreover, in this paper, we also propose a new framework for Efficient COmparisons of Malware detector Performances (ECOMP), and we illustrate its usage by implementing both SIMBIoTA and PATRIoTA in it and comparing their performances. It is important to emphasize that the ECOMP framework is not limited to SIMBIoTA and PATRIoTA, but it is designed to be an easily usable framework, which simplifies and standardizes the performance measurements and comparisons of any malware detection methods.

The structure of this paper is the following: In Chapter 2, we present the operation of SIMBIoTA and introduce two simple adversarial sample creation strategies that mislead it. In Chapter 3, we introduce PATRIoTA and present its design considerations in details. In Chapter 4, we compare PATRIoTA to SIMBIoTA in terms of malware detection capability and robustness against the adversarial sample creation strategies introduced in Chapter 2. We discuss the robustness of PATRIoTA against another adversarial sample creation strategy, as well as some alternative ways of increasing robustness against adversarial samples in Chapter 5. In Chapter 6 we introduce our newly proposed framework. Finally, we present some related work in Chapter 7 and conclude the paper in Chapter 8.

Chapter 2

Background

Let us start with a more detailed introduction of SIMBIoTA and the lack of its robustness against some simple adversarial sample creation strategies.

2.1 SIMBIoTA

SIMBIoTA [28] (SIMilarity-based IoT Antivirus) is a lightweight malware detection method tailored to resource constrained IoT devices. It detects malware by checking the similarity of scanned files to known malware samples, but it does this efficiently. In particular, SIMBIoTA exploits the fact that malware samples that belong to the same malware family are typically similar to each other, while samples from different families, as well as benign programs are dissimilar. This means that the similarity graph of the malware samples known to the antivirus provider is clustered and it is disconnected from the similarity graph of benign programs. This phenomenon is illustrated in Figure 2.1. Here, the similarity graph of a set of binary files is defined as a graph whose vertices represent the binaries and two vertices are connected with an edge if the corresponding files are similar according to some similarity measure. Typically, each cluster in the similarity graph can be represented by a few representative samples such that all members of the cluster are similar to at least one of the representative samples. It is then sufficient for malware detection to know only about the representative samples of the clusters: any scanned file that is similar to any of these representative samples are likely to be malware, while files that are not similar to any of the representative samples are likely to be benign.

More specifically, SIMBIoTA assumes that the antivirus provider continuously collects malware samples from its malware intelligence network (e.g., honeypots and various malware feeds), stores them in a database, and also creates their similarity graph. The (dis)similarity measure applied by SIMBIoTA is called TLSH difference [17] and two vertices of the similarity graph are connected if the TLSH difference between the corresponding malware samples is below the threshold 40. The selection of this particular threshold value is explained in [6]. The antivirus company then calculates a dominating set of the current similarity graph. A dominating set is a subset of the graph's vertices such that each vertex of the graph is either included in the dominating set or it is adjacent to a dominating vertex. The vertices in the dominating set are the representatives of all malware in the database of the antivirus provider. Finally, the antivirus provider delivers the TLSH hash values of the samples belonging to the current dominating set to all clients.

Figure 2.1: Illustration of a similarity graph of a set of binaries. Vertices represent the binaries and two vertices are connected with an edge if the corresponding files are similar according to some similarity measure. In this figure, two files are considered to be similar, if their TLSH difference score is below the threshold 40, where TLSH is a locality sensitive hash function and TLSH difference is a TLSH-based dissimilarity metric. Malware binaries are represented by red nodes and benign binaries are represented by green nodes. As it can be seen, there is no similarity between malware and benign files and the malware similarity graph is strongly clustered.

When scanning any file, the client examines how close the TLSH hash value of the scanned file is to the TLSH hash values of the samples belonging to the dominating set. It detects the scanned file as malware if it is similar to any of the samples in the dominating set, and as benign otherwise.

This procedure guarantees that the client detects all malware samples seen by the antivirus company (and included in the similarity graph), and can also detect previously unseen malware that is similar to the samples belonging to the dominating set. In addition, efficiency stems from the facts that the size of the dominating set is just a small fraction of the size of the entire similarity graph, thanks to the high clusteredness of the latter, and the TLSH value and TLSH difference calculations are very fast. Indeed, according to the evaluation in [28], SIMBIoTA required only 6-8 KB of storage capacity and it could decide about any file if it was malicious or benign in 0.12-0.14 ms. In addition, its malware detection capability proved to be surprisingly accurate: it achieved approximately 95% true positive detection rate even on previously unseen malware samples, while its false positive rate remained at 0% throughout the experiments.

2.2 Adversarial strategies

The malware detection operation in SIMBIoTA is simple, hence, it may be easy to mislead. In order to achieve that, one has to manipulate the TLSH hash value of a malware sample by modifying the sample without harming its malicious functionality. As SIMBIoTA uses the TLSH difference 40 as the similarity threshold, the intuition is that if the TLSH difference between the original sample and the modified sample becomes larger than 40, then SIMBIoTA will likely misclassify it. A modified malware sample with the same functionality as the original one is called an adversarial sample if it is likely to be misclassified as benign by the malware detection mechanism.

One approach for creating adversarial samples is to append extra bytes at the end of a malware binary such that those bytes are never executed, but they affect the calculation of the TLSH hash value. Two specific adversarial sample creating strategies, following this approach, have been proposed in [25]. They are called Chunker and Disguiser. Chunker appends a carefully chosen chunk of the original sample to itself with the goal of increasing the TLSH difference between the modified and the original samples above 40 (or beyond). Disguiser appends an appropriately chosen benign file to the malware binary and its goal is to decrease the TLSH difference between the modified malware and the benign file below 40 (i.e., to make the modified malware similar to the added benign file). These strategies are simple enough to be easily implemented by a real-world attacker. In addition, as shown in [25] (and later replicated in Chapter 4 of this paper), SIMBIoTA can be completely misled by these simple adversarial sample creating strategies such that its detection rate on the adversarial samples created by them is close to 0%.

Chapter 3

PATRIoTA

In Chapter 2, we mentioned that SIMBIoTA is not robust against the adversarial samples created with the Chunker and Disguiser strategies. Both attack strategies append some bytes at the end of an existing malware binary in such a way that those bytes are never executed, while the TLSH value of the modified sample becomes dissimilar to that of the original malware, and therefore, SIMBIoTA has a good chance of misclassifying it. In the case of these, and similar, append attacks, the original malware binary can be found in the adversarial sample. Hence, in order to detect such an adversarial sample as malware, we need a method that identifies the original malware inside the adversarial sample. PATRIoTA, the method we propose and describe in this section, will do exactly this: it identifies parts of known malware samples inside any file being checked with it. PATRIoTA can be viewed as a general method of defense against adversarial samples created with append attack strategies.

3.1 Overview

The design of PATRIoTA was inspired by SIMBIoTA (and their similarity is also reflected in their names). Basically, PATRIoTA is a modified version of SIMBIoTA where the difference is that PATRIoTA works with fixed size parts of malware samples instead of entire malware binaries. Not surprisingly, the architecture of PATRIoTA is also almost the same as that of SIMBIoTA, as it is illustrated in Figure 3.1. Malware samples are continuously collected from the intelligence network of the antivirus provider, and the PATRIoTA backend splits them into fixed-size parts, which we call *particles* in the sequel. Similar to SIMBIoTA, a similarity graph is built by the backend, but in this case, this graph is built from the malware particles. In addition, PATRIoTA uses a different similarity threshold to build the similarity graph. In Section 3.4, we explain how to determine the optimal values for the particle size and the similarity threshold used by PATRIoTA. Again similarly to SIMBIoTA, the backend computes a dominating set of the current similarity graph and makes the list of TLSH hash values of the dominating vertices available to clients.

The detection method on the client side is somewhat different in the case of PATRIoTA: the client splits the file to be checked into particles (of the same size used by the backend); calculates the TLSH hashes of the particles; and compares these TLSH hashes with those of the current dominating set. A file is considered malicious if it contains a threshold number of particles that are similar to known malicious particles. The selection of this threshold is discussed in Section 3.3.

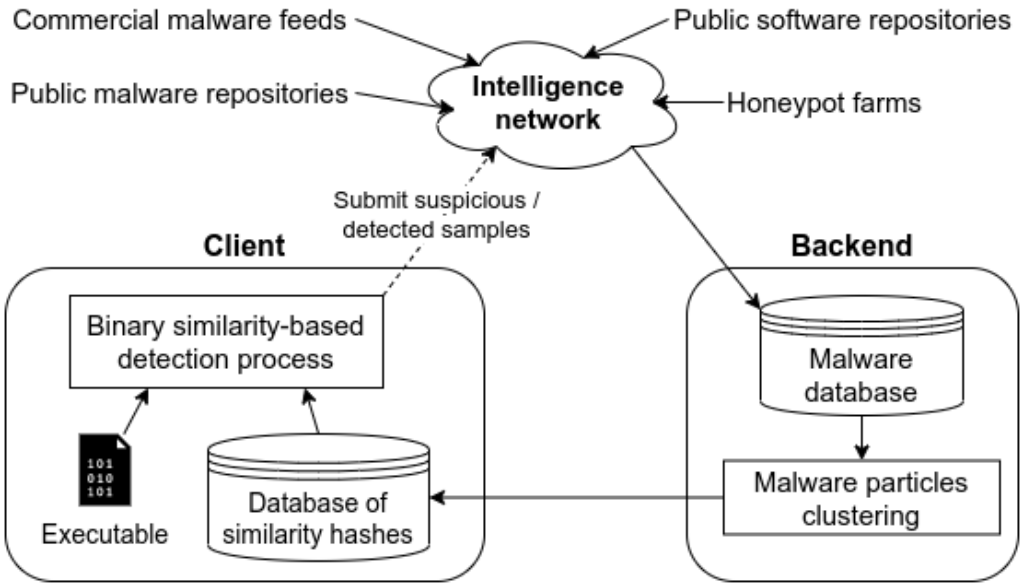

Figure 3.1: High-level overview of PATRIoTA.

3.2 Particle size and similarity threshold

PATRIoTA uses some special parameters, which we have already mentioned in the previous subsection, including the size of the particles and the similarity threshold used during the graph construction from the TLSH hashes. Finding the optimal configuration of these parameters is not a trivial task. We can state that the optimal configuration (if it exists at all) is highly context dependent; for example, we can imagine a situation where the low latency of the detection process is more critical than its memory usage.

Despite all this, we developed an iterative methodology to determine a recommended parameter configuration. We performed measurements to determine the optimal parameters on a smaller data set, not the one presented in Section 4.1. This data set consisted of 2000 malware and 2000 benign samples for both the ARM and the MIPS architectures.

When we were designing PATRIoTA, the first question was the size of the particles. We first considered the values of 1 kB, 2 kB, 4 kB, 8 kB, 12 kB and 16 kB, but later excluded 1 kB and 2 kB, because the number of graph nodes built from particles of those sizes grew unmanageably large.

PATRIoTA builds a graph from the TLSH values of malware particles, where the TLSH hash values are the nodes and there is an edge between two nodes if the TLSH difference of the two hash belonging to the nodes is below a certain similarity threshold value. SIMBIoTA uses 40 as TLSH similarity threshold, because the average clustering coefficient of the built graph is the highest in that case [6]. The same value cannot be used for PATRIoTA, because it does not build the graph from the TLSH hashes of entire malware samples, but from its particles. To determine the optimal value of the TLSH similarity threshold for different particle sizes, we used the same technique as for SIMBIoTA. In Figure 3.2, we measure the average clustering coefficient of the graph built from the particles of the 2000 malware samples using different TLSH similarity thresholds. We select the TLSH similarity threshold that gives the highest average clustering coefficient for each particle sizes (e.g., for particles of size 4 kB, the selected TLSH similarity threshold is 65 in the case of ARM samples).

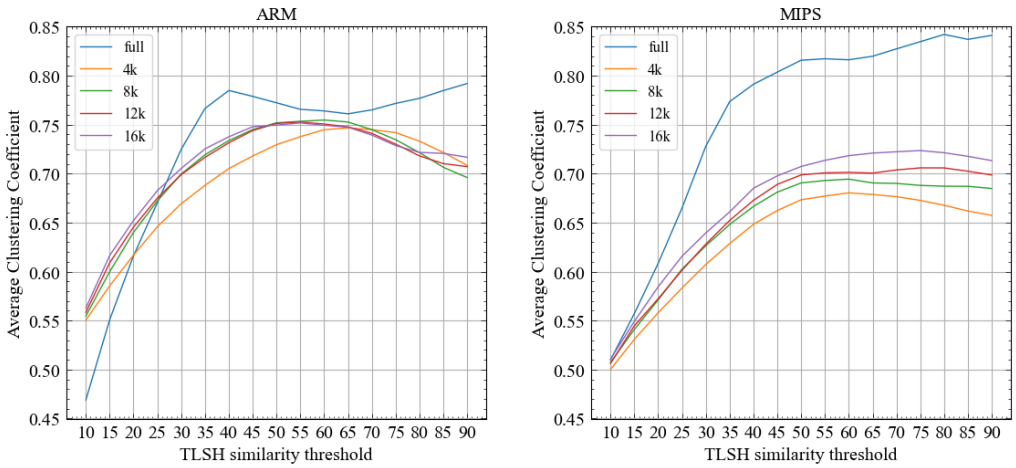

Figure 3.2: Average clustering coefficient as a function of the TLSH similarity threshold in the case of ARM (left) and MIPS (right) architectures. Different curves belong to different particle sizes, including the case where the particle size and the file size are equal (full).

3.3 Detection threshold

A suspicious sample is considered malicious if it contains at least a threshold number of particles that are similar to known malware particles. If this threshold is set to 1, the true positive detection rate (TPR) of malware will be as high as possible, but the unwanted effect may occur that even benign files are considered malicious (e.g. a benign and a malicious program use the same statically compiled library, therefore, both contain the same sequence of bytes). The consequence is that the larger the detection threshold is, the lower the false positive rate (FPR) and, unfortunately, the lower the TPR will be. So, we choose the smallest possible value where the FPR is still below 1%, which is 2 in the case of ARM samples and 4 in the case of MIPS samples (see Chapter 4).

3.4 Optimal configurations

At this point we have 4 possible particle size and TLSH similarity threshold configurations, but which of them is the best?

Before answering this question let's take a look at Figure 3.3, where we examine the number of similar particles between malware samples with the configuration of 4k particle size and 65 similarity threshold for ARM samples. For that, we divide the 2000 malware samples to 10% train and 90% test set, we split to particles the items in the train set and we build the graph from their TLSH hashes. Finally, we iterate over the elements of the test set, split each file, and count how many similar particles there are between them and the particles in the graph. In other words, we simulate the operation of PATRIoTA on a small data set and repeat this simulation ten times (just like in Chapter 4, in the case of the large data set). There are ca. 40 samples in Figure 3.3 that do not contain any similar malware particles to the particles in the train set, therefore, we cannot detect these. Furthermore, in Figure 3.3 we see how many malware samples would not be detected depending on the selected detection threshold. For instance, if we required that at least 3 particles of the

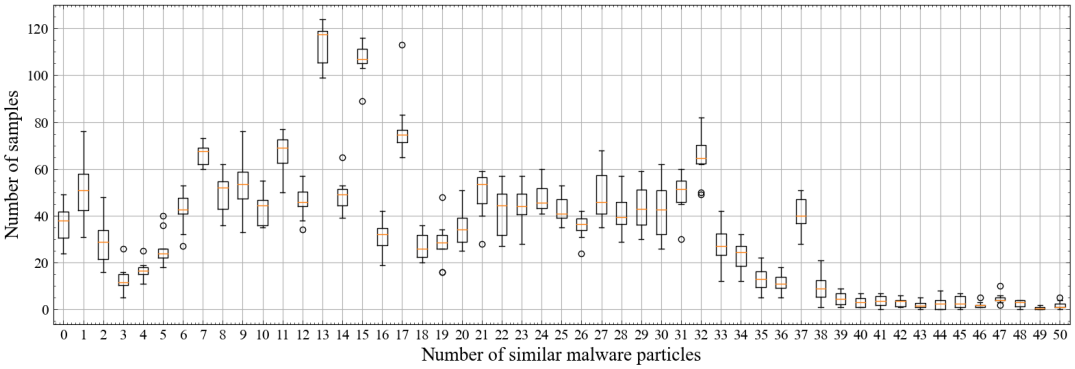

Figure 3.3: The x axis shows the number of particles in a sample from the test set that are similar to items in the train set (from 0 to 50), while the y axis shows the number of these samples in the case of 4k particle size and 65 TLSH similarity threshold configuration for ARM samples.

file should be similar to some known malware particle to detect the file as malicious (i.e., detection threshold 3), then ca. 40+50+30=120 malware samples would not be detected.

To select the best particle size and TLSH similarity threshold configuration, we examine how it changes the TPR and FPR values depending on the detection threshold. Figure 3.4 shows the ROC (Receiver Operating Characteristic) curves of the different configurations, where each jump in the step function corresponds to a certain detection threshold value between 1 and 10. According to our expectations, as the detection threshold increases, the FPR decreases, but so does the TPR. We choose the configuration with the highest AUC (Area Under the ROC Curve) value. This is 4k particle size and 65 TLSH similarity threshold for ARM samples and 8k particle size and 60 TLSH similarity threshold for MIPS samples.

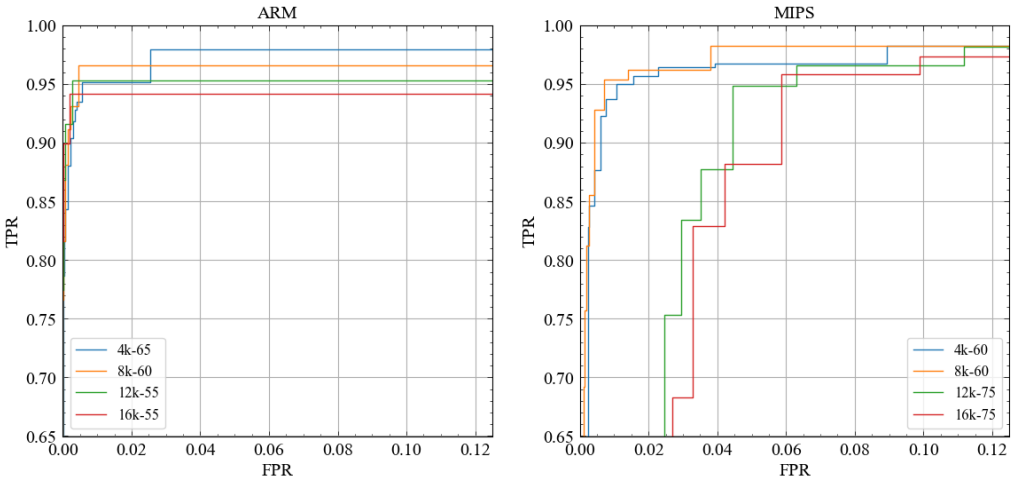

Figure 3.4: ROC curves of different configurations, where each jump in the step function corresponds to a certain detection threshold value between 1 and 10 in the ARM and MIPS cases.

Chapter 4

Evaluation

In Chapter 3, we presented PATRIoTA, including its architecture and operating principles, as well as the selection of its parameters (particle size, similarity threshold, and detection threshold), as the main contributions of this paper. It is time to evaluate PATRIoTA's performance, especially its ability to detect adversarial samples. However, before doing that, we present the data set and methodology used for the performance measurements.

4.1 Experiment design

In this work, we perform all experiments using the same dataset as used for the evaluation of SIMBIoTA [28]. 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.

As a first step for testing PATRIoTA, we split the malware samples into a 10% train set and a 90% test set. To do this, we use K-folds cross-validation [19], which is a reliable and frequently used model checking technique. We use K-folds with 10 folds and repeat each measurement 10 times, where the samples of each fold belong to the train set once, and the test set consists of the samples of the other 9 folds. This ensures that each malware appears exactly once in the train set and 9 times in the test set.

PATRIoTA does not need benign samples for training, so we add benign samples only to the test set. Moreover, we extend the test set with adversarial samples for measuring the robustness of the system. These adversarial samples are created using the two strategies mentioned in Section 2.2: Chunker and Disguiser. We create these adversarial samples from the malware binaries in the test set, simulating that an attacker has malware samples unknown to the antivirus company and can create adversarial samples from them. Table 4.1 shows the exact number of samples in the train and test set.

With the presented construction, we simulate the operation of PATRIoTA: we build the model properly from the train samples, and then give samples from the test set to the model for detection (see Chapter 3). Furthermore, since PATRIoTA was inspired by SIMBIoTA, we compare their performances in all aspects. Indeed, we train and test the

[1] `https://github.com/CrySyS/cube-maliot-2021` (accessed on October 17, 2023)

Table 4.1: Number of samples in the train and test set, in the ARM and MIPS cases.

ARM

	Malware	Benign	Chunker	Disguiser
Train	2,921	–	–	–
Test	26,288	4,727	24,285	26,288

MIPS

	Malware	Benign	Chunker	Disguiser
Train	1,872	–	–	–
Test	16,843	9,392	13,862 - 13,916	16,813 - 16,819

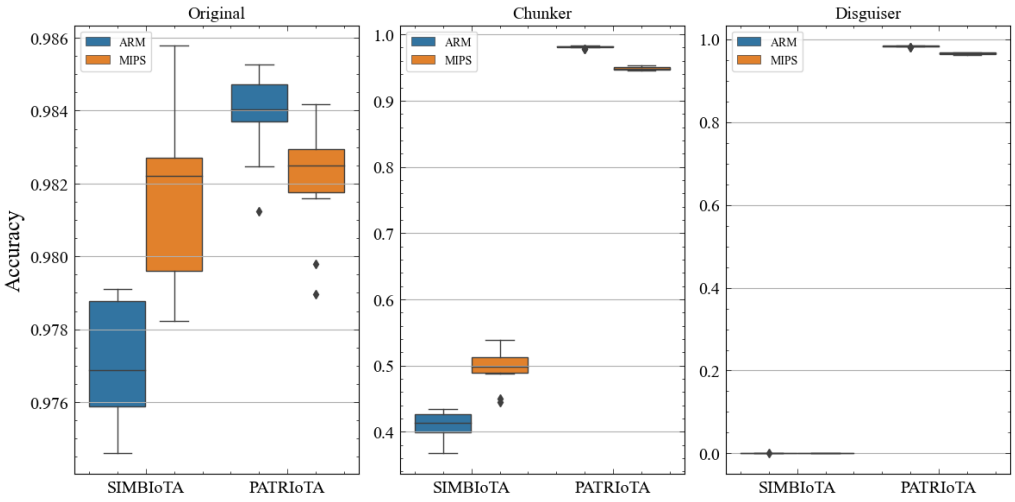

Figure 4.1: Comparison of the detection accuracy of SIMBIoTA and PATRIoTA on unmodified malicious and benign samples (Original), adversarial samples created with the Chunker strategy, and adversarial samples created with the Disguiser strategy, in the ARM and MIPS cases.

two systems on the same samples and measure the same performance metrics. In the next 3 subsections, we present the results of the performed simulation.

4.2 Detection capability

Using the experimental setup presented in the previous subsection, we measure the detection accuracy of SIMBIoTA and PATRIoTA in 3 different cases: on unmodified malware and benign files, on adversarial samples of the Chunker strategy, and on adversarial samples of the Disguiser strategy. Our goal is to achieve the highest possible accuracy in all 3 cases, while keeping the FPR of benign files below 1%. To do this, we try PATRIoTA with different detection thresholds (i.e., minimum number of particles of a file that need to be similar to known malware particles in order for the file to be classified as malware). The smaller the detection threshold is, the higher the TPR will be, but the FPR will increase too. According to our measurements, the optimal value for the detection threshold is 2 for ARM samples and 4 for MIPS samples, as for smaller values, the FPR exceeds 1%.

Table 4.2: Storage requirement of SIMBIoTA and PATRIoTA on the client side in the ARM and MIPS cases.

	SIMBIoTA	PATRIoTA
ARM	8,365 - 9,030 B	333,585 - 371,805 B
MIPS	5,775 - 6,440 B	111,965 - 139,370 B

In Figure 4.1, we compare the detection accuracy of the two system. PATRIoTA drastically outperforms SIMBIoTA in terms of accuracy in all test cases! On the sample set of unmodified benign and malicious programs, PATRIoTA has an impressive 98.5% accuracy in the case of ARM samples and 98.2% in the case of MIPS samples. Moreover, it performs extremely well even on adversarial samples of both the Chunker and Disguiser strategies, with 98% accuracy on ARM samples and 95% on MIPS samples.

4.3 Storage requirement

IoT devices are usually limited by resources, including available memory and storage capacity. Therefore, we measure the storage space requirement of PATRIoTA on the client side (i.e., on the IoT device), which in our case is the size of the dominating set multiplied by the size of the TLSH hash. Compared to SIMBIoTA, unfortunately, the higher accuracy and robustness of PATRIoTA comes with a higher storage requirement, due to the increased number of nodes in the dominating set. In Table 4.2, we present the required memory sizes of the two system.

4.4 Run time performance

Another price we have to pay for the increased detection accuracy and robustness of PATRIoTA is the increased processing time compared to SIMBIoTA. By detection time, we mean the time that elapses from the beginning of the binary scan of any file to the decision whether it is malicious or not. In the case of SIMBIoTA, this time consists of the TLSH hash computation time of the binary and the decision time of the model. For PATRIoTA, the detection time consists of the sum of 3 components: the time required to split the binary into fixed-size particles, the sum of TLSH hash computation time of the particles, and the sum of decision times required for particles. Basically, PATRIoTA performs SIMBIoTA's detection method multiple times, more precisely for each particle, until the number of particles considered malicious reaches the value of the detection threshold parameter. Therefore, PATRIoTA requires more time for detection than SIMBIoTA, as shown in Figure 4.2.

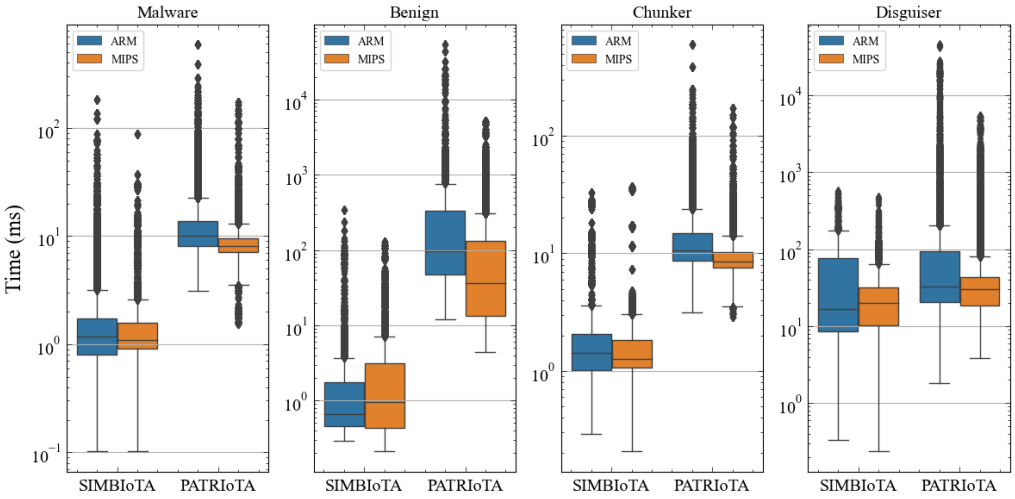

Figure 4.2: Comparison of the required detection times (on logarithmic y-axis scale) of SIMBIoTA and PATRIoTA, separately for malware samples, benign samples, adversarial samples of Chunker strategy, and adversarial samples of Disguiser strategy, in the ARM and MIPS cases.

Chapter 5

Discussion

In this work, we were concerned with increasing the robustness of binary similarity-based malware detection methods against adversarial samples that are crafted specifically to mislead a given malware detector. More specifically, we proposed PATRIoTA, a robust, similarity-based antivirus solution, which was inspired by SIMBIoTA [28]. It turns out that SIMBIoTA has a machine-learning based variant, called SIMBIoTA-ML, proposed in [18], and the robustness of SIMBIoTA-ML has already been studied in [26], where an adversarial training approach was proposed as a solution. So a natural question, at this point, is whether an adversarial training approach could have increased the robustness of SIMBIoTA as well. If so, then the need for a new approach, i.e., our PATRIoTA, would be much weaker.

It is clear what adversarial training means in case of a machine learning-based method: the training set is expanded with adversarial samples created by various known adversarial strategies. But SIMBIoTA is not a machine learning-based method. Nevertheless, we can define adversarial training quite intuitively for SIMBIoTA too: the antivirus provider extends the similarity graph of known malware samples with adversarial samples and computes the dominating set of this extended graph. One can then check the detection performance of this modified SIMBIoTA on adversarial samples to determine how robust this approach is.

We performed adversarial training of SIMBIoTA by extending the similarity graph of known malware samples with adversarial samples created from those known malware by the Chunker and Disguiser strategies introduced in [26], and computed the dominating set of the extended graph. We then measured the detection performance on adversarial samples created from malware unknown to the antivirus provider by the same Chunker and Disguiser strategies. The results we got were not so promising: adversarial ARM and MIPS samples created by the Chunker strategy were detected with 92% and 90% accuracy, respectively, while adversarial ARM and MIPS samples created by the Disguiser strategy were detected only with 86% and 67% accuracy, respectively. These results confirm the raison d'être of PATRIoTA.

In addition, while PATRIoTA was designed to be robust against adversarial samples that were created from existing malware samples by appending extra bytes to them, we have the intuition that it is also robust against other strategies that create adversarial samples that contain chunks of the original sample, as those chunks may result in particles that are similar to the particles of the original sample. In order to test this intuition, we measured the robustness of PATRIoTA against such a strategy. In particular, a very clever adversarial sample creation strategy against similarity-based malware detection was

proposed in [8] that consists in modifying a few unused portions of a malware binary (e.g., the section header tables were modified in [8]) such that the TLSH difference between the modified and the original files is maximized, while the functionality of the original binary is fully preserved, the size of the modified file remains the same as that of the original one, and even the binary content is only slightly changed. As reported in [8], it is rather easy to create adversarial samples in this way that are misclassified by SIMBIoTA: out of 2000 randomly chosen ARM malware samples, 1779 samples were suitable for such kind of modification, and 1465 samples could be created with a TLSH difference of at least 40 (the threshold used by SIMBIoTA) between the modified and original files. We tested both SIMBIoTA and PATRIoTA with those samples, and SIMBIoTA recognized only 17% of them as malware, while PATRIoTA detected a remarkable 98% of them as malware!

One may wonder whether statically linked libraries decrease the detection accuracy of PATRIoTA. Such libraries may be included in both malware and benign binaries, so actually, some portions of statically linked malware and benign samples that include the same libraries can be identical. This may lead to multiple similar particles in them, potentially above the threshold number used by PATRIoTA. In other words, benign files may contain particles resulting from linked libraries that are similar to particles seen in malware binaries using the same libraries. Such benign files may be classified as malware by PATRIoTA, which leads to an increased false positive rate. Indeed, we tested PATRIoTA on 118 statically linked ARM and 64 statically linked MIPS benign binaries and it misclassified 15% and 6% of them, respectively, as malware. This misclassification rate is not really acceptable, therefore, further research is needed to reduce it. We note that SIMBIoTA had a false positive detection rate of 0% throughout our experiments.

Chapter 6

ECOMP Framework

Recently, in our laboratory, similarity-based IoT malware detection has become an emerging topic, with a noticeable increase in the number of detection methods and attacks against them. Over time, numerous experiments have been conducted, each of which is interesting and meaningful when considered individually. However, they cannot be easily interpreted collectively. This situation called for urgent standardization.

6.1 Overview

In Chapter 4, we compared SIMBIoTA and PATRIoTA based on specific metrics, including detection capability, storage requirements, and run time performance. We implemented these measurements in our newly proposed framework, which was created for Efficient Comparisons of Malware Detector Performances (ECOMP). The ECOMP Framework simplifies and unifies the testing of various malware detection methods; it is not limited to SIMBIoTA and PATRIoTA.

Malware detection methods usually use a detection model, which can be a classical machine-learning model, like a Random Forest Classifier [18], or simply, a set of TLSH hashes [28]. A complete workflow for testing the performance of a malware detection solution typically involves two steps:

1. Training a model: This step requires using a training data set to train the malware detection model.
2. Testing the model: In this step, the trained model is evaluated using a separate testing data set, which should be different from the training data set to assess the model's generalization performance.

6.2 Modules

The framework divides the training and testing steps into four modules, each with a predefined interface. To implement the specific functionality for a detection method, we inherit from the module's abstract class. These modules can also optionally measure various performance metrics, such as running time, during their operations. Figure 6.1 shows the framework's modules and their relations with each other, in the following we present what we see in the figure in more detail.

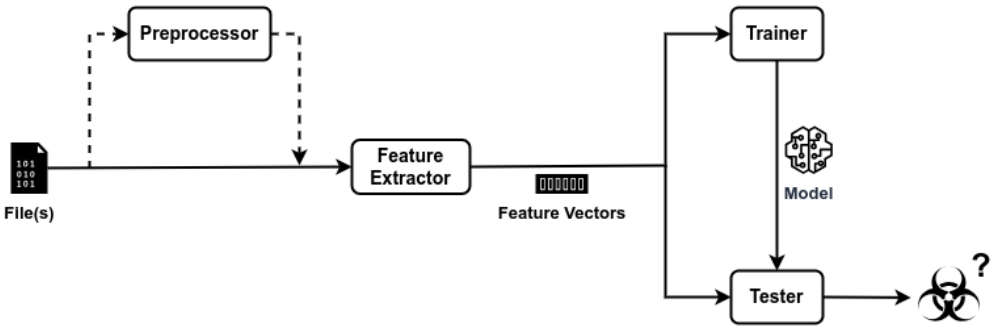

Figure 6.1: High-level overview of the ECOMP framework's modules and their relations with each other.

To test a malware detection method, we require both malware and benign samples. Typically, these input data are available as raw binaries, although some methods may demand a different format. *Preprocessor* modules are designed for this purpose: they take file(s) as input, perform necessary operations, and save the results. In the case of SIMBIoTA, which uses raw binaries as input, *Preprocessor* is unnecessary. However, it can be valuable for other malware detection methods, such as those that compute Control Flow Graphs (CFGs) from executable binary files [20].

In order to train and test a model we need to extract feature vectors from the input data. *Feature Extractor* modules do exactly this, they take file(s) as input then produce the feature vectors from it, which will be used during the training and testing process. In the case of SIMBIoTA the features are the TLSH hashes themselves, so *Feature Extractor* module of SIMBIoTA calculates the TLSH hashes of the provided input files while measuring the processing time as well.

At this point we have everything to actually train our detection method's model. *Trainer* module provides the trained malware detection model after fitting the training feature vectors to the corresponding target labels. In the case of SIMBIoTA the trained model is the dominating set of the similarity graph built from the TLSH hashes of the training sample set.

The last phase of the workflow is testing the trained detection model. To do this, we created *Tester* module, which uses the model created by the *Trainer* module on the test feature vectors and it returns the predicted labels (or the probabilities of each target labels) to the given features. *Tester* module of SIMBIoTA for each scanned file examines how close the TLSH hash value of the scanned file is to the TLSH hash values of the samples belonging to the dominating set. It detects the scanned file as malware if it is similar to any of the samples in the dominating set, and as benign otherwise.

Different circumstances call for different approaches in orchestrating the four modules. In the context of ECOMP framework we have two options: either write code to configure and use the modules, or use the modules from the command line by passing parameters.

6.3 Discussion

Driven by our intention to standardize the performance measurements of IoT malware detection methods, we managed to create a truly flexible system. Indeed, the ECOMP framework offers a set of interfaces that establish the required constraints for uniformity while granting researchers the flexibility to carry out their desired malware detection

and performance measurement tasks. The framework also supports unit testing, which comes with its every well-known advantages. Furthermore, it gives a concise and stylish visualization template for comparing different performance metrics (see the figures in Section 4). Finally, the ECOMP framework is implemented in Python 3.10 and it is available among the CrySyS Lab's private GitLab repositories. As this framework serves educational purposes, there is also a useful tutorial available in the repository, which provides a more precise, step-by-step guide on how to properly use the ECOMP framework.

Chapter 7

Related Work

Although SIMBIoTA, and thus PATRIoTA as well does not belong to ML-based malware detectors, adversarial examples and adversarial robustness, the main focus of this work are closely tied to machine learning. Additionally, the ECOMP framework is suitable to compare not only SIMBIoTA and PATRIoTA but ML-based malware detectors as well. Indeed, the ECOMP framework was developed with ML-based malware detectors in mind, and it has features that are particularly advantageous when the tested detector is ML-based. Therefore, we include an outlook on ML-based malware detection in this section.

ML-based malware detection solutions, unlike their traditional counterparts, are highly automated[29], thus they can keep pace with the increasing amount of malware. They use static, dynamic or hybrid program analysis techniques to extract information from samples which they use to construct feature vectors[21].

Statically obtained features could include opcode-based solutions, where the samples' instructions[27] are used to construct such a vector, gray scale images created from binaries[16] or solutions built to perform detection based on the control-flow graph of samples[20]. Dynamic analysis-based solutions can rely on API or system call traces[1] or network traffic, observed during the execution of the sample[14].

Solutions, where ML-based and cloud-based approaches are combined, can be highly advantageous in the IoT domain, since these solutions can relieve resource constrained devices from performing computationally heavy tasks[24], thus more complicated models can be used as well, like convolutional neural networks[22] and recurrent neural networks, while they can also boost the performance of more light-weight models, like random forests[27] and fuzzy pattern trees[7].

Adversarial attacks against malware detectors can also use multiple approaches[5]; here we highlight two of these, called *append* and *slack* attacks[23]. Append, as the name suggests, works by adding extra bytes to the end of samples, that will never be executed, thus they don't have any affect on the functionality of the modified sample. The strategies Chunker and Disguiser implement this approach. Slack attacks, on the other hand modify the content of so-called slack spaces in binary files. These are regions that contain no useful data, and usually exists because of alignment-related reasons: for example, the size of a section cannot be divided by the page size, but the next section's beginning must be aligned to another page, thus creating a slack space between the end of the section and the beginning of the next one. A special case of this approach is the strategy where the section header table is overwritten; as it is not required for loading and executing ELF files, it can be categorized as slack space. To use these strategies to fool the ML model,

solutions like a gradient-based approach can be used[13], or the feature extraction process can be attacked as well[9].

More advanced techniques, like program obfuscation can be used as well, to change the binary file while preserving its original functionality; to do so, one could use reinforcement learning[3], like Recurrent Neural Networks (RNNs) or Generative Adversarial Networks (GANs)[11].

Increasing the adversarial robustness of ML-based malware detectors is a logical next step in the decades old arms race between malware developers and antivirus vendors. One such attempt was to improve SIMBIoTA-ML by applying adversarial training[26], meaning that the training set was extended with adversarial examples. Our solution aims to achieve the same goal (i.e. increasing adversarial robustness), but using another approach. This method works on SIMBIoTA as well, which wasn't suitable for adversarial training. We also believe that this approach is superior, since it does not require adversarial examples and only employs more general assumptions regarding the strategy of the attacker.

Chapter 8

Conclusion

In this paper, we proposed PATRIoTA, a similarity-based IoT malware detection method, and showed that it has outstanding malware detection capabilities, while being robust against various adversarial sample creation strategies too. More specifically, we compared the performance and robustness of PATRIoTA to those of SIMBIoTA, an IoT malware detection method in a similar vein as PATRIoTA. PATRIoTA has a higher true positive detection rate, but it also has a higher false positive rate, it requires more storage capacity, and it has longer detection time than SIMBIoTA has. Its true advantage is its strong robustness against adversarial samples: indeed, SIMBIoTA can be completely misled by adversarial samples created from existing malware by appending extra bytes to them, whereas PATRIoTA detects those samples with very high accuracy. In addition, PATRIoTA proved to be robust against another adversarial sample creation strategy too that produces adversarial samples by modifying unused portions of an existing malware binary such that the modified binary becomes dissimilar to the original one, and hence, likely misclassified by SIMBIoTA.

We argued that edge gateways are well-positioned for performing malware detection and protecting resource constrained IoT devices behind such gateways. They can identify the transfer of executable files in the network traffic, check those executables with a malware detection mechanism, and block any traffic carrying malware. PATRIoTA can be used for such malware detection on edge gateways. Although it has a larger storage requirement than SIMBIoTA has, edge gateways also offer more storage space than typical IoT field devices do. PATRIoTA also has an increased running time, in particular when checking benign files. We believe that this does not hinder the use of PATRIoTA on edge gateways, but this requires further study and more measurements, in which the introduced ECOMP framework will be particularly useful.

In Chapter 5, we discussed that PATRIoTA may make false positive decisions on statically linked benign binaries if they include libraries that have also been used in malware. This issue also needs further study and it is on our future research agenda.

Acknowledgements

Supported by the **ÚNKP-23-2-I-BME-179** New National Excellence Program of the Ministry for Culture and Innovation from the source of the National Research, Development and Innovation Fund.

KULTURÁLIS ÉS INNOVÁCIÓS MINISZTÉRIUM

NATIONAL RESEARCH, DEVELOPMENT AND INNOVATION OFFICE HUNGARY

Furthermore, the research presented in this paper was supported by the European Union project RRF-2.3.1-21-2022-00004 within the framework of the Artificial Intelligence National Laboratory. The presented work also builds on results of the SETIT Project (2018-1.2.1-NKP-2018-00004), which was 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.

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] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose. SoK: Security evaluation of home-based IoT deployments. In *IEEE Symposium on Security and Privacy*, 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] 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.

[5] Kshitiz Aryal, Maanak Gupta, and Mahmoud Abdelsalam. A Survey on Adversarial Attacks for Malware Analysis. *ArXiv*, abs/2111.08223, 2021.

[6] Levente Buttyán, Roland Nagy, and Dorottya Papp. SIMBIoTA++: Improved Similarity-based IoT Malware Detection. In *2022 IEEE 2nd Conference on Information Technology and Data Science (CITDS)*, pages 51–56. IEEE, 2022.

[7] 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.

[8] Gábor Fuchs. Adversarial example creation strategies for defeating similarity-based IoT malware detection algorithms. BSc Thesis, Budapest University of Technology and Economics, 2023.

[9] Gábor Fuchs. Adversarial example creation strategies for defeating similarity-based IoT malware detection algorithms. Bachelor's thesis, Budapest University of Technology and Economics, 2022.

[10] S. Greengard. Deep insecurities: The Internet of Things shifts technology risk. *Communications of the ACM*, May 2019.

[11] Weiwei Hu and Ying Tan. Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN. *CoRR*, abs/1702.05983, 2017.

[12] Young Jae Kim, Chan-Hyeok Park, and MyungKeun Yoon. Film: Filtering and machine learning for malware detection in edge computing. *Sensors*, 22(6), 2022.

[13] 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.

[14] 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.

[15] Charlie Miller and Chris Valasek. Remote exploitation of an unaltered passenger vehicle. In *Black Hat USA*, 2015.

[16] Lakshmanan Nataraj, Sreejith Karthikeyan, Gregoire Jacob, and Bangalore S Manjunath. Malware images: visualization and automatic classification. In *Proceedings of the 8th international symposium on visualization for cyber security*, pages 1–7, 2011.

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

[18] 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. SCITEPRESS, 2022.

[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[20] Justin Sahs and Latifur Khan. A machine learning approach to android malware detection. In *2012 European intelligence and security informatics conference*, pages 141–147. IEEE, 2012.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] József Sándor. Robustness Against Evasion of Similarity-based IoT Malware Detection Methods. Scientific students' association report, Budapest University of Technology and Economics, 2022.

[26] József Sándor, Roland Nagy, and Levente Buttyán. Increasing the robustness of a machine learning-based IoT malware detection method with adversarial training. In *Proceedings of the 2023 ACM Workshop on Wireless Security and Machine Learning*, 2023.

[27] 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.

[28] Csongor Tamás, Dorottya Papp, and Levente Buttyán. SIMBIoTA: Similarity-based malware detection on IoT devices. In *Proceedings of the 6th International Conference on Internet of Things, Big Data and Security (IoTBDS)*, pages 58–69. SCITEPRESS, 2021.

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