

# Similarity-based Malware Detection on Embedded IoT Devices: Performance Evaluation and Robustness Against Evasion of Detection

Roland Nagy<sup>a</sup>, József Sándor<sup>b</sup>, Dorottya Papp<sup>c</sup>, Gergely Ács<sup>d</sup>, Levente Buttyán<sup>e</sup>

*Laboratory of Cryptography and System Security (CrySys Lab)  
Budapest University of Technology and Economics  
Műegyetem rkp. 3., H-1111 Budapest, Hungary  
{roland.nagy, jozsef.sandor, dpapp, acs, buttyan}@crysys.hu*

**Keywords:** IoT, embedded systems, malware detection, machine learning, adversarial examples.

**Abstract:** Embedded devices are increasingly connected to the Internet to provide new and innovative applications in many domains. However, these devices can also contain security vulnerabilities, which allow attackers to compromise them using malware. In this paper, we advocate the approach of similarity-based malware detection for embedded IoT devices, and we present SIMBioTA-ML, a light-weight, similarity-based antivirus solution that takes advantage of machine learning techniques. We show that SIMBioTA-ML can respect the resource constraints of embedded IoT devices, and it has a true positive malware detection rate of ca. 95%, while having a low false positive detection rate at the same time. In addition, the detection process of SIMBioTA-ML has a near-constant running time, which allows IoT developers to better estimate the delay introduced by scanning a file for malware. This property is advantageous in real-time applications, notably in the domain of cyber-physical systems. We also study the robustness of SIMBioTA-ML against two adversarial malware creation strategies aiming at the evasion of similarity-based detection and we show that it is robust to one of the strategies while, unfortunately, it can be completely evaded by the other.


## 1 INTRODUCTION


Embedded devices are special-purpose devices designed to carry out a well-defined set of tasks. Nowadays, these devices are increasingly developed with networking capabilities and are often connected to the Internet. This technological advancement led to what is now known as the Internet of Things (or IoT for short), and embedded devices with networking capabilities are also called embedded IoT devices.


The Internet of Things has enabled a wide range of new and innovative applications in many modern-day application domains, including healthcare, transportation and agriculture. Unfortunately, embedded IoT devices can have security weaknesses (just like other types of computers). Such weaknesses include insecure open ports, default or hard-coded passwords, and software vulnerabilities. Open ports and weak passwords allow attackers to easily gain access to the


device, while software vulnerabilities, notably those in the operating system of the device, allow for a wide range of malicious activities. Moreover, IoT devices in certain application domains are desirable targets for attacks, because they handle sensitive personal and business-related data, or control critical processes. Another reason for attackers to compromise IoT devices is to build a large-scale attack infrastructure and leverage the combined computing power of millions of such compromised devices. Consequently, there has been a rise in the number of malware targeting embedded IoT devices. One of the most infamous examples is Mirai (Antonakakis et al., 2017), 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 (Cozzi et al., 2020).


Detection of malware on embedded IoT devices is a challenging problem. In (Tamás et al., 2021), we proposed SIMBioTA (SIMilarity Based IoT Antivirus), an effective and efficient antivirus solution for such devices. The operating principles of SIM-

<sup>a</sup>  <https://orcid.org/0000-0003-2305-3271>

<sup>b</sup>  <https://orcid.org/0000-0001-8555-5475>

<sup>c</sup>  <https://orcid.org/0000-0002-9976-614X>

<sup>d</sup>  <https://orcid.org/0000-0003-4437-0110>

<sup>e</sup>  <https://orcid.org/0000-0003-4233-2559>

BIoTA are similar to those of traditional signature-based antivirus solutions, but SIMBIO TA uses TLSH hash values of known malware instead of raw binary signatures for detection purposes. TLSH (Oliver et al., 2013) is a similarity hash algorithm, which means that small variations in the input do not alter the TLSH output significantly. In other words, similar inputs result in similar TLSH hash values, and SIMBIO TA takes advantage of this feature. More specifically, in case of SIMBIO TA, embedded IoT devices store only a few TLSH hash values of known malware, and they compare the TLSH hash values of new files to these stored hashes. If the TLSH hash of a new file is similar to that of a known malware, then the new file is detected as malware. The main advantages of SIMBIO TA are its light-weight requirements for storage, computation, and bandwidth, as well as its remarkable detection capabilities. Indeed, according to the experiments reported in (Tamás et al., 2021), SIMBIO TA achieved a true positive detection rate of ca. 90%, even for previously unseen malware, and a false positive detection rate of 0%.

In this paper, we also use TLSH hash values for malware detection on IoT devices, but in a manner different from that of SIMBIO TA. Our key observations are that TLSH hash values can serve as compact representations of binary files and, thanks to their well-defined structure, they can be used as feature vectors for training machine learning models, which can then be used for malware detection. We show that this approach can result in interesting trade-offs in terms of detection performance and resource usage on embedded devices.

This paper is an extended version of (Papp et al., 2022), which was presented at the 7th IoTBDS conference<sup>1</sup>. Compared to the conference abstract, this paper contains significant amount of new contributions on studying and comparing the robustness of SIMBIO TA-ML and SIMBIO TA against adversarial strategies aiming at creating malware samples that evade similarity-based malware detection.

More specifically, our contributions in this paper are summarized as follows:

- We introduce SIMBIO TA-ML, which replaces SIMBIO TA’s database of TLSH hash values with a random forest classifier trained on TLSH hashes of malware and benign files.
- We measure the true and false positive detection rates of SIMBIO TA-ML, as well as its storage requirements and running time.

---

<sup>1</sup>7th International Conference on Internet of Things, Big Data and Security (IoTBDS), held online on April 22-24, 2022. More information: <https://iotbds.scitevents.org/?y=2022> (accessed: Aug 11, 2022)

- We compare SIMBIO TA-ML’s measurement results to those of SIMBIO TA and discuss the advantages and disadvantages of both solutions. Specifically, we find that SIMBIO TA has lower storage requirements and false positive detection rate, but SIMBIO TA-ML outperforms SIMBIO TA in terms of true positive detection rate even for new, previously unseen malware samples. We also show that SIMBIO TA’s database of TLSH hash values increases over time, which has an impact on its detection time. Specifically, the larger the database is, the longer it takes for SIMBIO TA to decide whether a new file is malicious or not. By contrast, we show that SIMBIO TA-ML has a near-constant running time, which allows for better estimation of the delay introduced by the antivirus solution, and this can be an advantage in case of real-time applications in the domain of cyber-physical systems.
- We study and compare the robustness of SIMBIO TA-ML and SIMBIO TA against two adversarial strategies for creating malware samples 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 to the original ones. The first strategy adds a chunk 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 remains similar to the benign file (and hence dissimilar to the original malware sample). We show by measurements that SIMBIO TA-ML is robust against the first strategy, but it can be misled by the second one, while SIMBIO TA has poor robustness against both strategies.

The paper is structured as follows: Section 2 provides background information on machine learning-based malware detection, its robustness against adversarial evasion strategies, and the operation of SIMBIO TA. In Section 3, we introduce SIMBIO TA-ML and discuss our changes to SIMBIO TA’s architecture in order to use machine learning. The detection performance and resource consumption of SIMBIO TA-ML is evaluated and compared to that of SIMBIO TA in Section 4. The evaluation and comparison of the robustness of SIMBIO TA-ML and SIMBIO TA against adversarial strategies aiming at creating malware samples that evade similarity-based malware detection is presented in Section 5. Finally, Section 6 concludes the paper.

## 2 RELATED WORK

In this section, we provide background information on machine learning-based malware detection and its robustness to adversarial evasion strategies, and we summarize the operation of SIMBioTA.

### 2.1 Malware detection with machine learning

Traditionally, antivirus products rely on signatures and heuristic rules that try to capture complex static patterns in known malware samples. One problem with this approach is that, like any method relying on static features of binaries, it can be evaded by packing, encryption, obfuscation, and code polymorphism. These techniques modify a malware sample's binary form in such a way that it cannot be detected by the same signature or heuristic rule, while, at the same time, its behavior remains the same. Another problem, which is more important for our present work, is that creating signatures and heuristic rules requires expert knowledge, and often necessitates reverse engineering techniques. As a result, it is a time consuming and tedious task. Hence, signature-based and heuristic approaches have a hard time keeping up with the constantly evolving threat landscape<sup>2</sup>, and their reliance on expert knowledge is a scalability bottleneck for antivirus companies.

In response, significant research effort has been dedicated to automate malware detection using machine learning (Ye et al., 2017; Ucci et al., 2019; Gilbert et al., 2020). Machine learning requires features, which are usually automatically extracted using static and dynamic program analysis techniques (Soliman et al., 2017). Features can be derived from a variety of sources, including the samples' instructions (Dovom et al., 2019; Takase et al., 2020), control-flow (Alasmary et al., 2019), invoked API functions and system calls (Abbas and Srikanthan, 2017; Shobana and Poonkuzhali, 2020), and messages sent over the network (Meidan et al., 2018; Goyal et al., 2019). Feature extraction can result in thousands of features, some of which may be redundant and can be eliminated with data mining techniques.

For efficient malware detection, machine learning-based approaches require lots of benign and malicious samples to train on. These samples are often collected from so-called intelligence networks. Nowadays, users' machines run only a client-side antivirus component, which may perform local

detection, but it can also request a server's assistance during the detection process. This setup is also known as cloud-based malware detection. The client-side component sends suspicious samples to a server in the cloud, which performs a more in-depth analysis, e.g., by executing the sample in a sandbox, makes a decision, and informs the client. At the same time, the server collects these submitted samples, which can then be used for training machine learning models.

Cloud-based malware detection coupled with machine learning has also been proposed for embedded IoT devices (Sun et al., 2017; Hussain et al., 2020). This is an advantageous combination for embedded IoT devices, because resource-heavy analysis is performed in the cloud and the resource-constrained devices need to run only a light-weight client-side component. The client-side component either forwards all files to the cloud for analysis or applies a pre-trained machine learning model to detect malware. Proposed machine learning models include light-weight convolutional neural networks (Su et al., 2018), recurrent neural networks (HaddadPajouh et al., 2018), random forest classifiers (Takase et al., 2020), fuzzy and fast fuzzy pattern trees (Dovom et al., 2019). Many existing works use static features (Ngo et al., 2020), including function call graphs (Nguyen et al., 2020), grey scale images of binaries (Karanja et al., 2020), strings (Hwang et al., 2020), and instruction opcodes (Nakhodchi et al., 2020).

### 2.2 Evasion of ML-based malware detection

Adversarial evasion strategies against machine learning-based classifiers were initially created in the image recognition domain, but soon later, they were also adopted in the field of ML-based malware detection. A comprehensive survey on the topic can be found in (Aryal et al., 2022). Adversarial examples, in this case, are malware samples crafted in such a way that they evade ML-based malware detection. They are typically created by perturbing existing malware samples in such a way that their original functionality is preserved, but the features on which the ML-based detector works are sufficiently changed to induce misclassification. Perturbation of existing samples can mean modifying bytes in, adding bytes to, or deleting bytes from the samples. Two specific methods are called *append* and *slack attacks* in (Suciu et al., 2019), where *append attacks* add bytes to the end of an existing malware binary, whereas *slack attacks* add or modify bytes in the slack regions of a binary, which are gaps between

<sup>2</sup><https://www.sophos.com/en-us/medialibrary/pdfs/technical-papers/sophoslabs-2019-threat-report.pdf> (accessed: Feb 28, 2022)

neighboring sections of an executable file. In this paper, we will consider only append attacks because of their simplicity.

Depending on the knowledge available to the attacker, we can distinguish between, so called, white-box and black-box attacker models (Aryal et al., 2022). In the white-box model, the attacker has full knowledge about the ML model which he wants to create adversarial examples for, including even its training data and tuned hyper-parameters. In the black-box model, on the other hand, the attacker only has access to the input and the output of the model, and he has no information about the model and its training data and parameters. In practice, the attacker may also have partial knowledge about the system under attack, which is sometimes called a grey-box model. In this paper, we follow a black-box approach: apart from knowing the general concepts of our similarity-based malware detectors, the attacker is assumed to have no information on any particular details, parameters, or training data.

As for generating bytes to be added to a malware binary in order to create an adversarial example, there have been many strategies proposed (see Section 7 of (Aryal et al., 2022)). In gradient-based approaches (Anderson et al., 2017; Kolosnjaji et al., 2018; Kreuk et al., 2019; Suciú et al., 2019), the gradient of the change of the output or some internal parameter of the model is used to determine the required change on the input that pushes the sample towards the benign class the most. Program obfuscation techniques have also been considered (Park et al., 2019; Song et al., 2021), as they can modify binaries without changing their functionality. Reinforcement learning-based approaches have been proposed in (Anderson et al., 2017; Anderson et al., 2018), where an agent is provided with a set of functionality preserving operations on binaries, and reinforcement learning is used to derive, in an iterative manner, the sequence of required operations on a given malware sample that transforms it to a detection evading sample. Finally, Generative Adversarial Networks (GAN) and Recurrent Neural Networks (RNN) have also been used in this context; the most prominent examples are (Hu and Tan, 2017b) and (Hu and Tan, 2017a), respectively.

### 2.3 SIMBIO TA

SIMBIO TA was proposed in (Tamás et al., 2021). It is a light-weight antivirus solution with limited requirements for storage, computation, and bandwidth, hence suitable for embedded IoT devices. SIMBIO TA relies on a large malware database maintained on a backend server. This malware database is assumed

to be continuously updated with samples obtained from an intelligence network as mentioned in Subsection 2.1. The server computes the TLSH hash values of the samples in its database, and pushes a subset of these TLSH hashes to the client-side antivirus component on the embedded IoT devices, where a lightweight algorithm uses them to detect malware based on binary similarity. Therefore, SIMBIO TA requires resource-constrained embedded IoT devices to store only a small database with a few TLSH hash values.

In (Tamás et al., 2021), SIMBIO TA was evaluated on a total of 47,937 malicious samples and a total of 14,119 benign samples for the ARM and MIPS architectures. In the experiments, the set of samples was divided into two groups: the samples known to the backend via the intelligence network, and the samples found only in the wild. The samples known to the backend were used to construct the database of TLSH hash values. Based on the metadata of malicious samples available in VirusTotal<sup>3</sup>, the samples were also put into so-called “weekly batches”, i.e., sets of samples that were first submitted to VirusTotal on the same week. At the beginning of each week, the database of TLSH hashes were updated and the detection performance was measured in two ways. First, we checked the true positive detection rate for all samples in previous weeks’ weekly batches. Second, we also submitted samples from the wild of the next two weeks’ weekly batch to see SIMBIO TA’s detection performance for new, previously unseen malware samples. The experiments measured a false positive detection rate of 0%, a true positive detection rate above 90% for samples of previous weeks’ weekly batches, and a true positive detection rate of ca. 90% for the next two weeks’ weekly batches. Throughout the experiments, fewer than 200 bytes were necessary to update the TLSH hashes stored on the embedded IoT device. By the end of the experiments, the storage requirement on the embedded IoT device was 10 kB in the case of ARM and 6.5 kB in the MIPS case.

Despite its remarkable features, SIMBIO TA has a number of limitations as well. First, similar to other malware detection solutions relying on static features, analyzing obfuscated or encrypted samples is challenging for SIMBIO TA. Second, as we show in this paper, the bigger the database of similarity hash values, the longer it takes for SIMBIO TA to decide whether a given file is malicious or not. This can be a challenge in IoT environments where embedded devices must comply with real-time requirements, because the run time delay introduced by SIMBIO TA is hard to design for. Last, even though a true positive detection rate of 90% on average for new, previously

<sup>3</sup><https://www.virustotal.com/> (accessed: Jan 8, 2022)

unseen malware samples is surprisingly good, existing literature suggests that machine learning-based malware detection approaches can achieve even better results.

In this paper, we modify SIMBIO TA’s architecture to enable embedded IoT devices to take advantage of machine learning-based malware detection. Specifically, we replace the database of TLSH hash values with a random forest classifier trained on TLSH hashes of known malware and benign files. We show that this modification can increase the true positive detection rate by 5% on average, even for new, previously unseen malware samples. We also show that our trained random forest classifier has a near-constant run time, which allows IoT system developers to better estimate and design for the delay introduced by the antivirus solution.

### 3 ARCHITECTURE AND DESIGN OF SIMBIOTA-ML

We now discuss our proposed solution to improve SIMBIO TA with machine learning. We discuss our modifications to SIMBIO TA’s architecture in Subsection 3.1 and our design choices for machine learning in Subsection 3.2. We call the resulting ML-based antivirus solution SIMBIO TA-ML.

#### 3.1 Architectural overview

The original architecture of SIMBIO TA consists of both client-side and server-side components. Client-side components are located on embedded IoT devices and are responsible for protecting devices from malware via a detection process. The detection process takes as input the unknown file to be checked and a database containing TLSH hash values of known malware samples. The unknown file’s TLSH hash value is then compared to the TLSH hash values in the database in a pairwise manner. If the unknown file is determined to be similar to a known malware sample, it is considered malicious.

The task of server-side components is to keep the database of TLSH hash values up-to-date. These components are located on a backend server. The backend maintains a malware database, which receives malicious samples from honeypots, malware feeds, and malware analysis sandboxes via the intelligence network. Samples in the malware database are represented in a graph, where nodes are the TLSH hash values of the samples, and an edge connects two nodes if the corresponding TLSH hashes are similar enough according to some similarity metric. The

backend then computes a dominating set over this graph and the TLSH hash values of the nodes in the dominating set are sent to the client-side as an update.

Our main improvement to SIMBIO TA is to replace the dominating set construction with machine learning. The modified architecture is shown in Figure 1. On embedded IoT devices, we replace the database of TLSH hash values with a machine learning model. Therefore, the modified detection process takes as input the unknown file to be checked and the machine learning model. The modified detection process applies the machine learning model to the unknown file to decide whether the file is malicious or not. The machine learning model is trained on the backend using both malicious and benign samples. Therefore, we keep SIMBIO TA’s intelligence network and require it to supply the backend with benign samples as well. Benign samples could be received from IoT vendors or from public software databases.

#### 3.2 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. Dynamic program analysis, i.e., monitoring a program’s execution, however, leads to degraded performance, which is a challenge in the IoT setting. Therefore, we need features whose extraction is lightweight and can be done statically.

TLSH (Oliver et al., 2013) hash values can be considered static features because their calculation involves only the processing of the raw bytes in the program file. Moreover, TLSH has a light-weight calculation time in the range of milliseconds, which makes it suitable in the context of malware detection on IoT devices. More specifically, computing a TLSH hash value involves the following steps:

1. Process the raw byte string using a sliding window of size 5 to populate an array of bucket counts.
2. Calculate quartile points  $q_1$ ,  $q_2$ , and  $q_3$  based on the buckets’ values.
3. Construct the hash value’s header based on the quartile points.
4. Construct the hash value’s body.

The first three bytes of the resulting TLSH hash value is a header with following parts<sup>4</sup>:

- the first byte is a checksum value;

<sup>4</sup>The TLSH implementation at <https://github.com/trendmicro/tlsh> (accessed: Jan 9, 2022) appends two extra bytes to the beginning of the header for versioning purposes.

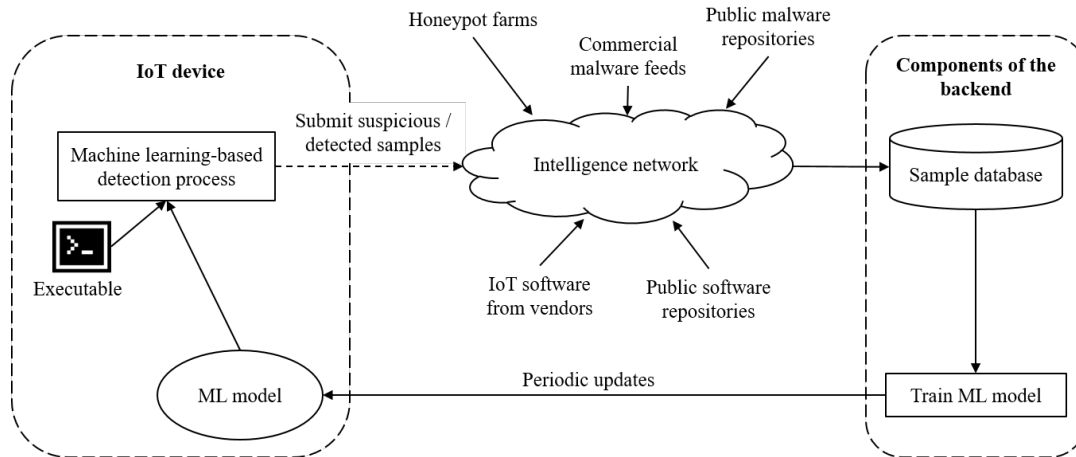


Figure 1: Architecture of SIMBioTA-ML

- the second byte stores the so-called  $L$  value, which is calculated from 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:

```
82 A4 02 13 79 E2 86 B1 E7 65 18 ...
```

The first byte of the header is a checksum, which has the value of hexadecimal 82 in our example. This is followed by the  $L$  value, which is hexadecimal A4 in this case. Next come the  $Q1$  and  $Q2$  ratios, which are hexadecimal 0 and 2, 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 1, in the example, encodes the 2-bit values 00 and 01 of the first two buckets. Similarly, the next hexadecimal number 3 encodes the 2-bit values 00 and 11 of the next two buckets, etc.

We transform the TLSH hash value into 131 features by splitting the hash value into smaller parts. Specifically, we take from the header the  $L$  value, the  $Q1$  ratio, and the  $Q2$  ratio. We then split the bytes representing buckets into bit pairs, which gives us 128 2-bit features for the 128 buckets. We train a random forest classifier over these extracted features. Choosing a random forest classifier is advantageous because it automatically filters non-predictive features.

## 4 PERFORMANCE EVALUATION

In this section, we compare the performance of SIMBioTA-ML to that of SIMBioTA and discuss their advantages and disadvantages. Specifically, we discuss the experiment design and the used data set in Subsection 4.1. Subsections 4.2 and 4.3 present the true positive and false positive detection rates, respectively. We compare the two solutions' storage requirements in Subsection 4.4 and their running times in Subsection 4.5.

### 4.1 Experiment design

We perform all experiments using the same data set as used for the evaluation of SIMBioTA. 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, which we extended with 4,727 benign ARM samples and 9,392 benign MIPS samples for the purpose of our study. 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). We made CUBE-MALIoT-2021 publicly available<sup>5</sup> for use by the IoT malware research community. To the best of our knowledge, such a large dataset containing raw binaries of IoT malware was not previously available publicly, and we hope that CUBE-MALIoT-2021 will become a *de facto* benchmark dataset in IoT malware detection, in order to satisfy the need for the comparability and reproducibility of results of different research groups.

<sup>5</sup><https://github.com/CrySyS/cube-maliot-2021> (accessed: Jan 9, 2022)

We also follow the same experiment design as used in (Tamás et al., 2021) for SIMBIO TA. The timeline of the experiment is between January 1st, 2018 and September 15th, 2019, divided into weeks. We assume that both SIMBIO TA and SIMBIO TA-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, and each weekly batch is further divided into two groups. The first group, which contains 10% of that weekly batch’s samples and is called the *intelligence part*, is made available to the backend for processing. These samples represent the knowledge obtained by the antivirus company from the intelligence network. The second group, called the *wilderness part*, contains 90% of that weekly batch’s samples, and it is assumed to exist only in the wild and is never revealed to the backend. The wilderness parts of weekly batches are used to evaluate the antivirus solutions’ true positive detection rate.

SIMBIO TA-ML also requires benign samples in order to train the machine learning model on a balanced data set. However, we have no metadata available for benign samples. Therefore, we randomly assign benign samples to be part of either the *training* or *test* sets for each architecture. In the case of ARM, the *training* set contains 2,921 benign ARM samples, and for MIPS, the corresponding *training* set contains 1,872 MIPS samples. Each week, we randomly select the same number of benign samples from the training sets as the number of malicious samples in the intelligence part of that weekly batch. Selected benign samples are sent to SIMBIO TA-ML’s backend for training the machine learning model. Samples in the test sets are never revealed to the backend and are used to measure false positive detection rates.

Note that our experiment design results in SIMBIO TA-ML’s backend having less training data available than what is usually the case in machine learning. Researchers often use 80% of their data sets for training purposes and use the remaining 20% as the testing set. In our case, however, the backend can only train on 10% of the malicious samples such that we can compare its performance to that of SIMBIO TA. For SIMBIO TA-ML’s backend to have a balanced data set, it has access to 61.78% of the benign ARM samples and 19.93% of the benign MIPS samples.

The random forest classifier trained on the backend for SIMBIO TA-ML also needs to be configured. Specifically, the number of decision trees that make up the random forest has to be specified. This number represents a trade-off between the detection capability of the machine learning model and the memory

required to apply the model on the embedded IoT device. The more decision trees there are in the model, the better the detection capability is. However, having more decision trees also increases the model size, increasing the amount of memory the embedded IoT device must have in order to apply the model. We set the number of decision trees to 10, which gave us a good trade-off between the two conflicting requirements.

Our method of assigning benign samples to the training and test sets introduces randomness into the experiment. To balance this randomness, we repeat the experiment 12 times and use traditional box plots to present the results. The data points of our box plots show the results of the 12 runs of our experiment for each week.

## 4.2 True positive detection rate

We measured the true positive detection rate of SIMBIO TA and SIMBIO TA-ML with the wilderness parts of weekly batches. In order to measure the performance for existing malware, we submit the wilderness parts of all previous weekly batches to the embedded IoT device for detection. We also measure the performance of new, previously unseen malware by submitting the wilderness part of the current weekly batch to the detection process. Note that we assume embedded IoT devices to receive updates to their detection processes at the beginning of each week. Therefore, the wilderness part of the current weekly batch contains samples that can be considered coming from the future.

The measured true positive detection rate for samples of the wilderness parts of previous weekly batches is shown in Figure 2. The left-hand side of the Figure shows the performance of SIMBIO TA and the right-hand side shows the performance of SIMBIO TA-ML. Both antivirus 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, SIMBIO TA-ML consistently outperforms SIMBIO TA by having a true positive detection rate above 95% throughout the measurement.

Figure 3 shows the true positive detection rate for the wilderness parts of current weeks for both SIMBIO TA and SIMBIO TA-ML. The left-hand side depicts the performance of SIMBIO TA and the right-hand side shows the performance of SIMBIO TA-ML. SIMBIO TA’s performance varies in time and it is only by the second half of the experiment that its performance reaches 90-95%. SIMBIO TA-ML also shows

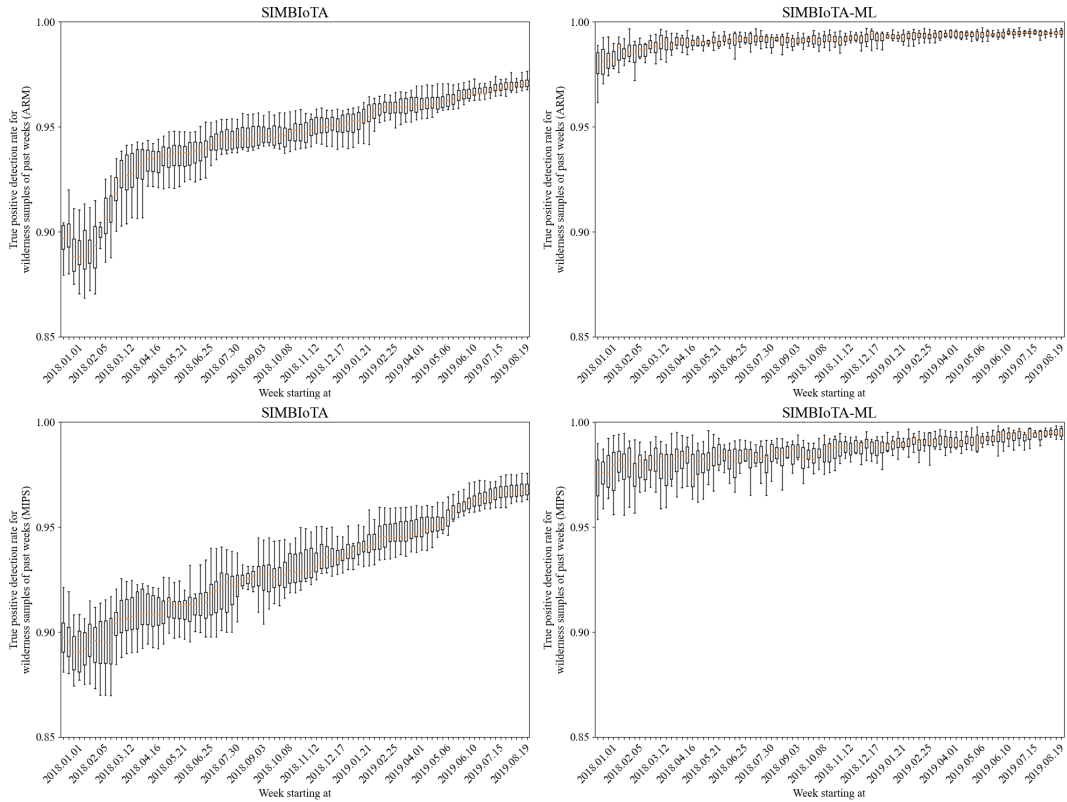


Figure 2: Box plot of the true positive detection rate for samples of the past for SIMBIO TA and SIMBIO TA-ML

variations in its true positive detection rate but the variation is smaller than that of SIMBIO TA, and performance stays above and around 95% for the majority of the experiment. Therefore, we conclude that SIMBIO TA-ML outperforms SIMBIO TA in this regard as well.

### 4.3 False positive detection rate

In order to measure the false positive detection rate of SIMBIO TA and SIMBIO TA-ML, we conduct the following experiment. In the case of SIMBIO TA, the backend does not need benign samples due to the antivirus solution’s design. Therefore, we submit all benign samples to SIMBIO TA for detection. In the case of SIMBIO TA-ML, however, the backend requires benign samples in order to train the machine learning model on a balanced dataset. As a result, SIMBIO TA-ML’s backend has access to the benign samples in the training set. Therefore, we only submit benign samples from the test set to SIMBIO TA-ML’s detection process.

In our experiments, SIMBIO TA did not detect any benign samples as malicious, hence achieved a false positive rate of 0, which is consistent with the results

reported in (Tamás et al., 2021). The same cannot be said for SIMBIO TA-ML, however. Machine learning classifiers have the tendency to sometimes misclassify inputs and our random forest classifier is no exception. The weekly false positive detection rate on benign samples is shown in Figure 4. In the case of benign ARM samples, SIMBIO TA-ML’s false positive detection rate stays below 1% on average throughout the experiment. For benign MIPS samples, the false positive detection rate goes slightly above 1% on average at the beginning of the experiment. It then steadily decreases as more and more benign MIPS samples are revealed to the backend. As we discussed in Subsection 4.1, our experiment design provides less training data to the backend than what is usually recommended in literature. This is especially the case for benign MIPS samples, because the data set is divided into 19.93%-80.07% for training and testing, respectively. Taking this into consideration, we conclude that while SIMBIO TA-ML’s false positive detection rate is higher than that of SIMBIO TA, it is still acceptable for malware detection.



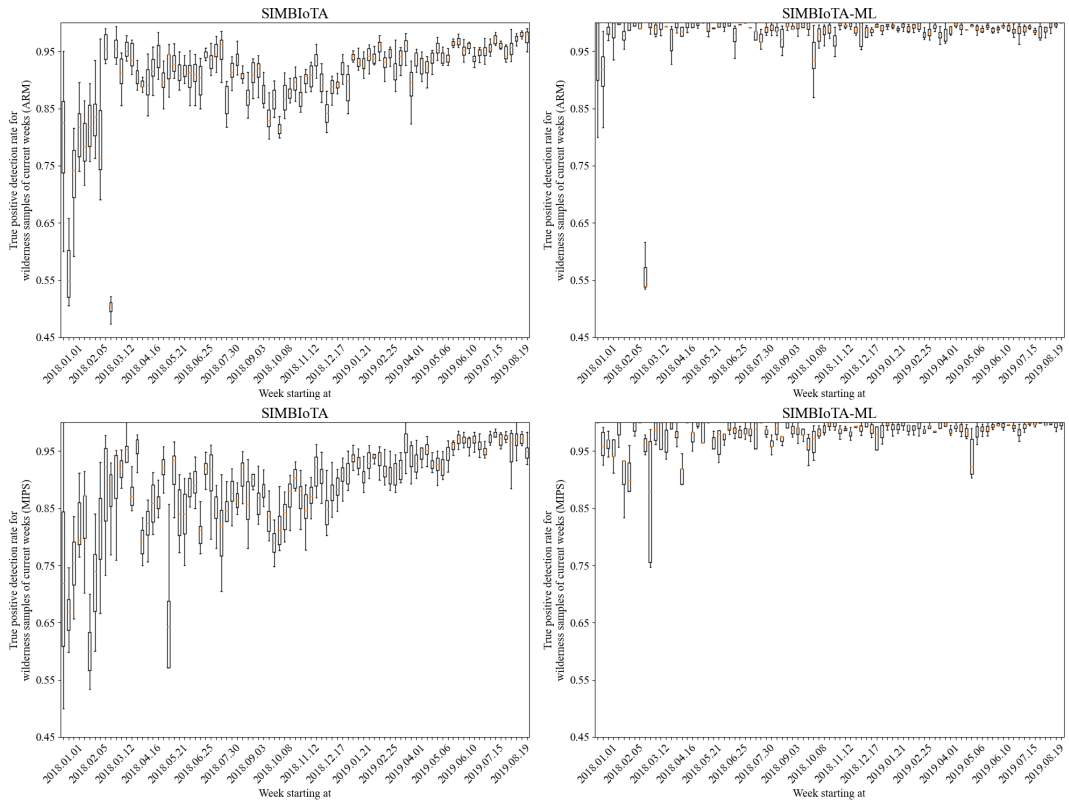


Figure 3: Box plot of the true positive detection rate for previously unseen samples for SIMBIO-TA and SIMBIO-TA-ML

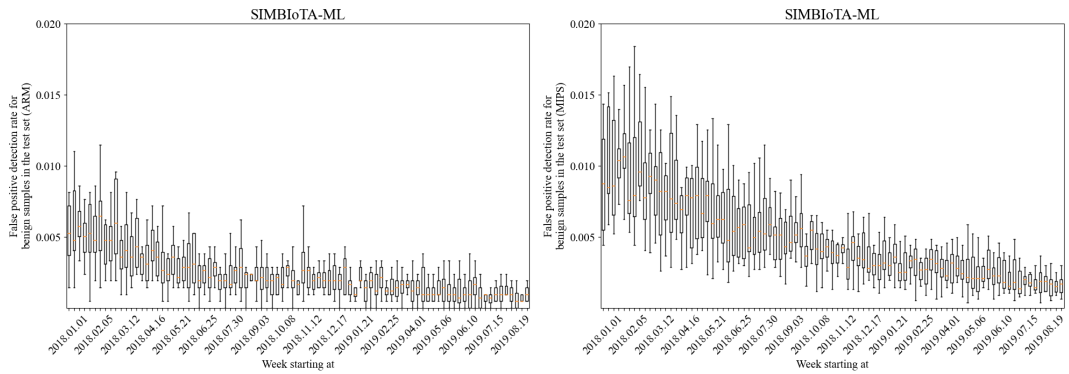


Figure 4: Box plot of the false positive detection rate for benign samples in the test set for SIMBIO-TA-ML

## 4.4 Storage requirement

Throughout our experiments, we measured the amount of storage necessary to hold SIMBIO TA’s database of similarity hashes and SIMBIO TA-ML’s machine learning model. In the case of SIMBIO TA, each similarity hash is 35 bytes, therefore, the total amount of storage necessary is 35 times the number of entries in the database. In the case of SIMBIO TA-ML, our implementation for the random forest classifier uses the scikit-learn<sup>6</sup> Python module. In order to measure the amount of storage necessary to hold the model, we used the pickle<sup>7</sup> 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.

The storage requirements for both SIMBIO TA and SIMBIO TA-ML are shown in Figure 5. While the storage requirements of both antivirus solutions increase over time, SIMBIO TA-ML’s requirements are orders of magnitude higher, going from ca. 40 KB to ca. 150 KB by the end of our experiment. By contrast, SIMBIO TA’s database of similarity hashes require less than 10 KB of storage throughout the experiment. Therefore, we may conclude that SIMBIO TA-ML is not fit for very low-end embedded devices, which typically have only tens of kilobytes of RAM and a few hundred kilobytes of Flash memory (Ojo et al., 2018). However, such devices usually do not have an operating system and they do not handle files, therefore, they are not really in the scope of our work. On the other hand, middle-range and high-end embedded devices with megabytes of memory available would be able to use SIMBIO TA-ML.

## 4.5 Run time performance

The last aspect by which we compare SIMBIO TA and SIMBIO TA-ML is their run time performance. Specifically, we measure the time it takes for both solutions’ detection process to decide whether a submitted file is malicious or not. We performed this measurement on a non-real time Linux operating system, therefore, small fluctuations in the measurements are possible due to task scheduling in the system.

The run time performance of SIMBIO TA and SIMBIO TA-ML for determining that a submitted file is malicious is shown in Figure 6. SIMBIO TA’s performance microseconds as it only needs to calculate the difference between TLSH hashes and compare the

result to a threshold value. However, SIMBIO TA has to do the comparison in a pair-wise fashion, i.e., it has to compare the TLSH hash value of the unknown file to each similarity hash value in its database individually. It is therefore not surprising that as the size of the database increases, so does the run time of the detection process. This is also the explanation for the growing difference between the minimum and maximum run time we measured. Depending on where the similar hash value is located in the database of similarity hashes, SIMBIO TA’s detection process needs to perform a different number of comparisons before a decision can be made. Unfortunately, in application areas where the delay caused by an antivirus product is of importance, e.g., due to real time requirements, this is an undesirable feature.

SIMBIO TA-ML requires more time to apply the machine learning model: its run time performance is a little above 1 ms. While this would result in a larger delay in real systems than that caused by SIMBIO TA, this delay is near constant. This is advantageous from the system operator’s standpoint because this delay is easy to take into consideration during system design and operation.

The run time performance of SIMBIO TA and SIMBIO TA-ML for determining that a submitted file is benign is shown in Figure 7. The run time delay that SIMBIO TA’s detection process would cause on a real system is even higher in this case. The reason for this is that in order for SIMBIO TA’s detection process to make a decision about an unknown benign file, it has to compare the file’s TLSH hash value to all the similarity hash values in its database. SIMBIO TA-ML’s detection process, however, always applies the same machine learning model to every file, therefore, the run time performance is the same for both malware and benign files.

## 5 ROBUSTNESS AGAINST EVASION OF DETECTION

Once similarity-based malware detection becomes wide-spread in practice, attackers will naturally try to construct malware samples that evade this type of detection too. This means that they would try to create malware samples that are misclassified by similarity-based malware detectors as benign files. In the field of machine learning, carefully crafted inputs that are misclassified by a previously trained ML model are called *adversarial examples* (Barreno et al., 2010). Clearly, the same concept can be extended to other types of (i.e., non-ML-based) classifiers too. Hence, it seems interesting to investigate both SIMBIO TA-ML

<sup>6</sup><https://scikit-learn.org/stable/> (accessed: Jan 11, 2022)

<sup>7</sup><https://docs.python.org/3/library/pickle.html> (accessed: Jan 11, 2022)

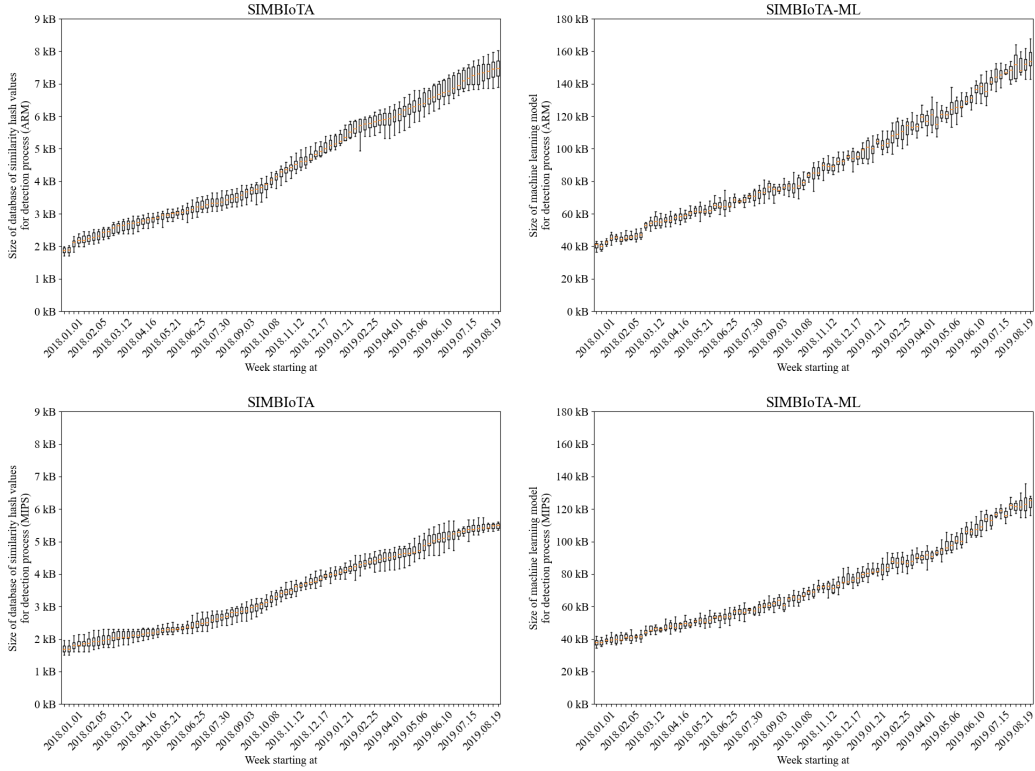


Figure 5: Box plot of the storage requirements for SIMBIOtA and SIMBIOtA-ML

and SIMBIOtA in terms of their robustness with respect to such adversarial examples.

In order to do so, in this section, we first introduce two strategies for creating adversarial examples for SIMBIOtA and SIMBIOtA-ML in Subsection 5.1 and evaluate the detection rate of SIMBIOtA and SIMBIOtA-ML on samples generated by these strategies in Subsection 5.2.

## 5.1 Strategies for creating adversarial examples

Adversarial examples for malware detection can be created in multiple ways. One approach could be to construct new malware with completely novel functionality. However, we do not consider this approach here, because constructing new malware is difficult, and hence, this is not the most economical way for attackers to create adversarial examples. In addition, the detection performance of SIMBIOtA and SIMBIOtA-ML on new malware has already been measured in Section 4, where we plotted their true positive detection rate for previously unseen samples in Figure 3.

Instead, we are interested in the approach of creat-

ing adversarial examples by modifying existing malware samples, as this seems to be a more economically viable strategy for attackers. Obfuscating existing malware samples should be mentioned here as an example for this approach, but we do not consider this either, 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, as we explained above.

In other domains (notably in case of image recognition), adversarial examples are often constructed in the feature space, which means crafting feature vectors (instead of real inputs) that are misclassified by the ML model under study. However, this approach has problems in the domain of malware detection, because the attacker would ultimately need to find or to construct functional computer programs that have the crafted feature vectors that are misclassified by the ML model, and this seems to be difficult. In addition, in the case of non-ML-based malware detection, the

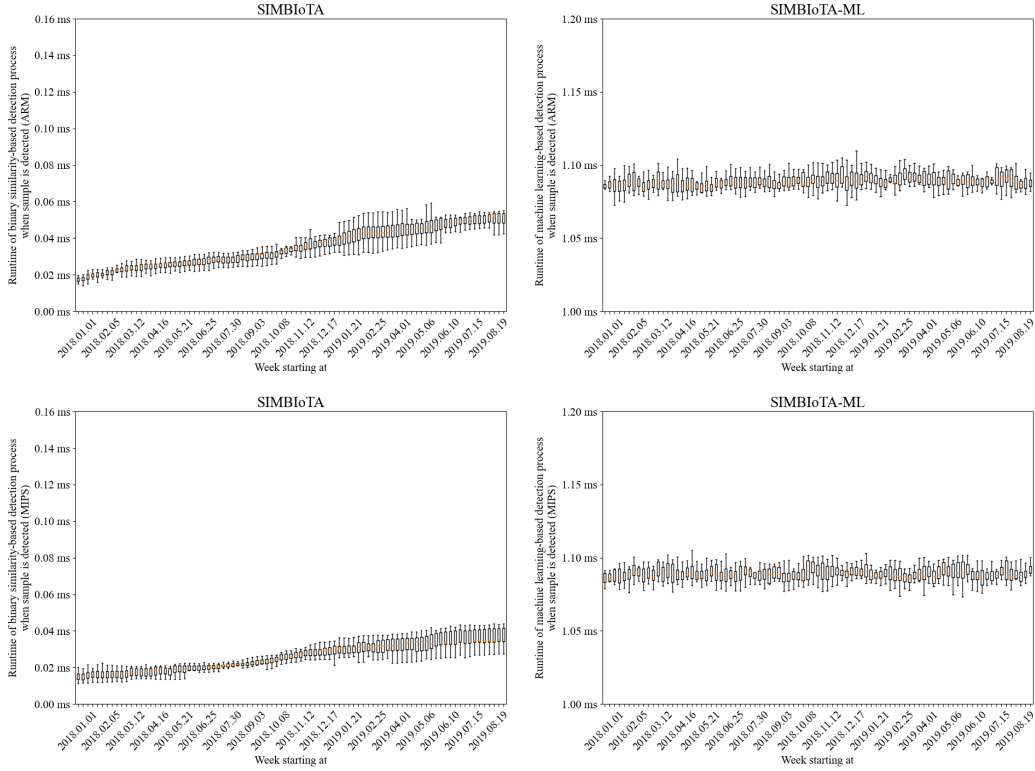


Figure 6: Box plot of the run time of the detection process for “malicious” decision for SIMBIOta and SIMBIOta-ML

concepts of the feature vector and feature space may not be well-defined. Therefore, we are interested in attacks aiming at crafting real inputs, rather than feature vectors as adversarial examples.

Even with all these constraints, there are still many ways to construct adversarial examples for malware detection by modifying malware binaries along different strategies. Therefore, we further restrict ourselves to adversarial modification strategies that result in an executable program with the same functionality as the original malware and that minimize the effort of the attacker.

More specifically, we consider here two strategies that can easily satisfy the above requirements of preserving functionality and requiring minimal effort. Both strategies concatenate extra bytes to the end of the malware binary in such a way that those extra bytes never get executed but they will be considered in the computation of the TLSH value by SIMBIOta and SIMBIOta-ML. In case of the first strategy, called *Chunker*, the extra bytes added are obtained as a chunk of the malware binary itself. In case of the second strategy, called *Disguiser*, an entire known benign binary is concatenated to the malware. One can think of strategy *Chunker* as a method that

tries to add a small amount of extra bytes to the malware binary such that *the modified sample becomes sufficiently different from the original malware*. Strategy *Disguiser*, on the other hand, can be viewed as a method to hide the malware within a potentially much larger benign file, hoping that *the resulting sample remains sufficiently similar to the benign file* (and hence, dissimilar to the original malware).

### 5.1.1 Chunker

Recall that both SIMBIOta and SIMBIOta-ML operates on the TLSH values of the inputs; SIMBIOta uses the TLSH values directly, while SIMBIOta-ML uses them as feature vectors for a random forest model. Hence, intuitively, the goal of adding extra bytes to a malware sample should be to obtain a modified sample whose TLSH value differs from that of the original sample to the extent that the modified sample is misclassified as a benign file by both algorithms.

In addition, as explained and justified in (Buttyán et al., 2022), SIMBIOta uses the value 40 as the TLSH difference threshold: if the TLSH difference between a new file and any known malware sample is smaller than 40, then the new file is classified as malware, otherwise it is classified as benign.

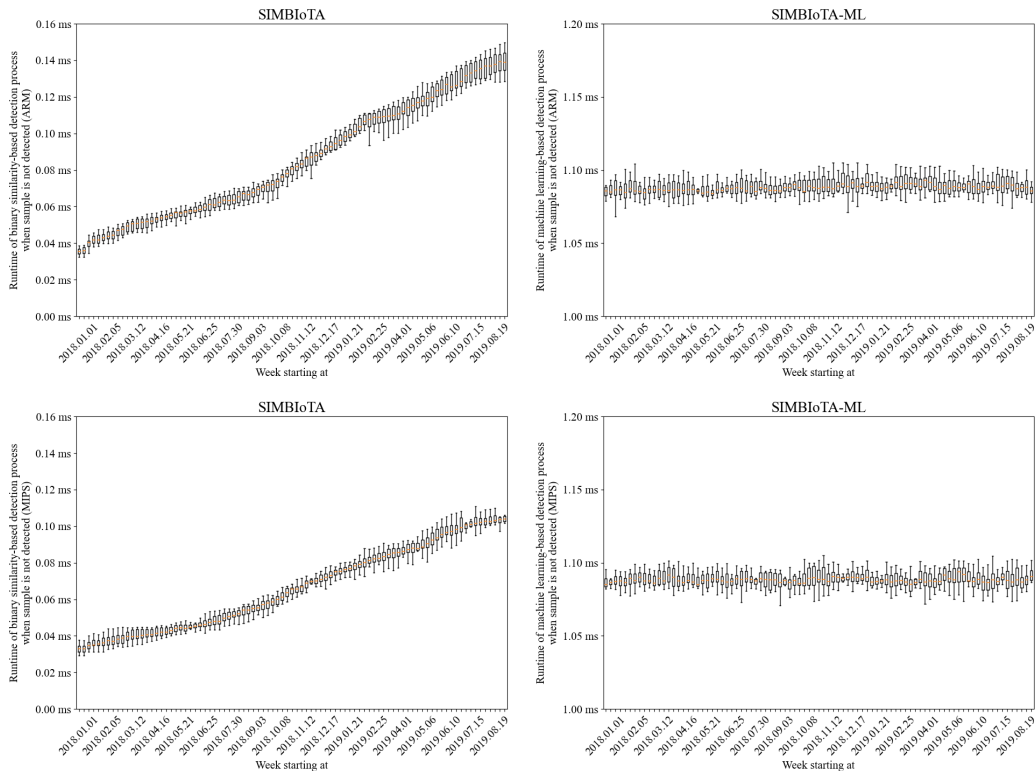


Figure 7: Box plot of the run time of the detection process for “benign” decision for SIMBIO TA and SIMBIO TA-ML

So a natural idea for adversarial example creation is to add extra bytes to a malware sample in such a way that the TLSH difference between the modified sample and the original one becomes 40 or larger. We expect that an adversarial sample created in this way would very likely mislead the SIMBIO TA classifier; however, it is less clear how SIMBIO TA-ML would cope with such an adversarial sample, as it does not make decisions directly on the TLSH difference between the new sample and previously seen malware. Our measurements, presented in Subsection 5.2 will help to answer this question.

Another question to consider is how to generate the extra bytes that are added to the malware sample to be modified. A practical requirement here could be to preserve the byte statistics of the original sample. In particular, constant or random bytes would be easily spotted by simple static analysis. In addition, even if the extra bytes are generated by a byte-entropy preserving manner, if the added content does not look like some meaningful program code, then the modified sample may appear suspicious to a static analyzer. Hence, we propose to take a chunk of the original sample itself and use it as the extra bytes concatenated to obtain the modified sample.

In order to determine how many extra bytes will likely be sufficient to drive the TLSH difference between the original sample and the modified one above 40, we did an experiment: We took a subset of size 2,000 of the ARM samples from the CUBE-MALIOT-2021 dataset (i.e., the dataset we also used for performance evaluation in Section 4). Then, we split each sample into 20 equal chunks (i.e., each chunk having the size of 5% of the sample size), and we chose the chunk whose byte entropy is the closest to the byte entropy of the original sample. Finally, we concatenated this chunk to the sample 1, 2, ..., 8 times (i.e., we increased the sample size with 5%, 10%, ..., 40%, respectively). For each case, we measured the TLSH difference between the original sample and the modified one. The obtained box plot of the results is shown in Figure 8. We note that a similar box plot was obtained for 2,000 MIPS samples randomly chosen from the CUBE-MALIOT-2021 dataset too, but we do not show that for saving space.

As expected, the more extra bytes we add in the described way to a sample, the larger the TLSH difference becomes between the original and the modified samples. More importantly, based on Figure 8, we can conclude that increasing the sample size with

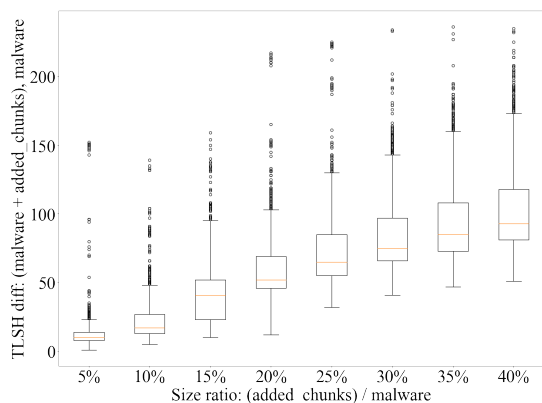


Figure 8: TLSH difference between the original sample and the adversarial example created from it by strategy Chunker as a function of the amount of bytes added.

20% (by adding the chosen chunk 4 times) results in a TLSH difference of 40 or above for the majority of the samples. However, it may happen that increasing the sample size with 20% is not sufficient, so the attacker should check the actually achieved TLSH difference. In addition, if the attacker wants to achieve a TLSH difference larger than 40, then he may need to perform an even larger increase in sample size. For instance, achieving a TLSH difference of 60 appears to need a 25% or even 30% increase in the sample size (i.e., concatenating the chosen chunk 5 or 6 times to the original sample).

To summarize, strategy Chunker works as follows: The attacker takes a malware sample to be modified, and splits the sample into 20 equal sized chunks such that each chunk has the size of 5% of the original sample size. Then, he chooses the chunk whose byte entropy is the most similar to the byte entropy of the original sample. Finally, the attacker concatenates the chosen chunk to the original sample multiple times, depending on the target TLSH difference he wants to achieve. For instance, to achieve a TLSH difference of at least 40 between the modified sample and the original one, the attacker concatenates the chosen chunk 4 times (i.e., increases the sample size with 20%). In any case, the attacker computes the resulting TLSH difference between the original sample and the modified one. If the target TLSH difference is achieved, then the attacker is done. If the TLSH difference is smaller than the target TLSH difference, then the attacker must concatenate the chosen chunk more times or re-start from another malware sample.

### 5.1.2 Disguiser

Strategy Chunker tries to modify malware samples such that they become dissimilar to their originals. In contrast to this, strategy Disguiser tries to embed mal-

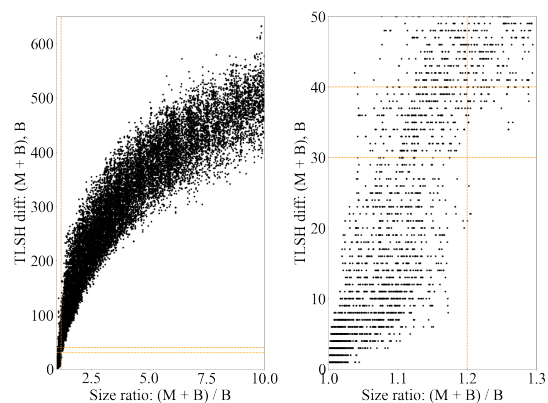


Figure 9: Illustration of the effect of the size ratio between the adversarial example and the hosting benign file on the TLSH difference between them when strategy Disguiser is used. The  $x$ -axis shows the ratio of the sizes and the  $y$ -axis shows the TLSH difference.

ware into benign binaries in such a way that the resulting files remain similar to the hosting benign files. In practice, the embedding is performed by concatenating the benign binary to the malware binary, hence when executed, the malware will run and control is never passed to the benign part.

One issue to consider in this case is the ratio of the sizes of the malicious and the benign parts. Intuitively, if a small benign file is concatenated to a large malware, then the malicious part may dominate, and the TLSH value of the result will not be similar to the TLSH value of the benign part. So the attacker may want to embed small malware samples into large benign binaries. However, this increases the size of the adversarial samples that he ultimately needs to deliver to victims. Hence, ideally, the attacker would like to minimize the size of the benign file in which his malware is embedded, subject to the constraint that it is still ensured that the TLSH value of the resulting adversarial sample is similar to the TLSH value of the hosting benign file.

To understand the effect of the ratio of the sizes of the benign and malware parts on the TLSH difference between the adversarial example and its benign part, we performed an experiment: We randomly chose 100 malware samples and 300 benign files from the dataset used in the performance evaluation in Section 4, we paired these malware and benign binaries in all possible ways, and we created the scatter plot shown in Figure 9. Each point in this plot represents a pair  $(M, B)$  of a malware  $M$  and a benign file  $B$ . Furthermore, the  $x$  coordinate of a point is the ratio of the sizes of  $M + B$  and  $B$ , where  $M + B$  denotes the sizes of  $M$  and the benign file  $B$  concatenated, and the  $y$  coordinate is the TLSH difference between  $M + B$  and  $B$ .

As in this case, the attacker is interested in a small TLSH difference between  $M + B$  and  $B$ , the interesting part of the figure is the bottom left corner of the scatter plot. For this reason, we enlarged that part on the right side of Figure 9. As it can be clearly seen, when the ratio of the sizes of  $M + B$  and  $B$  is above 1.2, the TLSH difference tends to get above the threshold 40. This means that we can only hope for keeping the TLSH difference low (i.e. below 40, or even below 30) when the malware part remains smaller than 20% of the benign part.

Given this observation, strategy Disguiser can be summarized as follows: The attacker takes a malware sample and chooses a known benign file such that the ratio of the sizes of the malware and the benign file is smaller than 20%. The malware and the benign parts are concatenated, and the attacker computes the TLSH difference between the resulting sample and the hosting benign file. If this TLSH difference is smaller than 40, then the attacker is done. Otherwise, the attacker chooses another benign file of appropriate size, and repeats the above step, until finally he succeeds in constructing an adversarial example that is sufficiently similar to a benign file.

## 5.2 Accuracy measurements

Now we study the robustness of SIMBIO TA and SIMBIO TA-ML against the above described strategies for creating adversarial examples. We measure robustness by creating adversarial examples using those strategies and presenting them to SIMBIO TA and SIMBIO TA-ML, already updated and trained, respectively. We record the classification results (malware or benign) for the presented adversarial examples and compute the accuracy (i.e., the ratio between the number of correct detections and the total number of samples presented) of both SIMBIO TA and SIMBIO TA-ML. Note that in this case, accuracy equals 1 minus the misclassification rate of the algorithms. An accuracy close to 1 means that large part of the adversarial examples presented are correctly detected as malware. When the accuracy is close to 0, the classifier is misled by the adversarial examples, as most of them are not detected as malware.

For the purpose of the robustness measurement, we divided both the ARM and the MIPS samples in the CUBE-MALIOT-2021 dataset into three groups depending on the sample size. The first group contains all samples with size smaller than 50 KB, the second group contains all samples with size between 50 KB and 120 KB, and the third group contains all samples with size larger than 120 KB. It turns out that each of these groups contains roughly one third of the

total number of samples in both the ARM and the MIPS cases. We call these groups of malware samples the set of small (S), medium (M), and large (L) samples, respectively.

Then, for studying the robustness against strategy Chunker, we randomly chose 4,000 samples from each group and applied strategy Chunker on them with target TLSH difference values 40 and 60. The reason to also use the target TLSH difference of 60, besides the difference threshold 40 used by SIMBIO TA, was that a larger target difference results in an adversarial example that differs even more from the original sample. Table 1 shows the number of resulting adversarial examples in each case both for the ARM and the MIPS samples.

ARM			
Target TLSH diff	S	M	L
40	3,933	3,655	3,630
60	3,814	3,440	3,357
MIPS			
Target TLSH diff	S	M	L
40	2,051	3,215	3,189
60	2,040	2,854	2,721

Table 1: 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.

As for strategy Disguiser, we randomly chose 100 samples from each group of malware (small, medium, large) and we also randomly chose 300 benign files from the same dataset that was used for the performance evaluation of SIMBIO TA-ML in Section 4. We combined each malware sample with each benign file that satisfied the size constraint (i.e., the size of the malware was at most 20% of that of the benign file) using strategy Disguiser with target TLSH difference values 30 and 40. In this case, besides the known TLSH difference threshold of 40, we used the smaller target difference value of 30 too, because that results in stronger similarity of the created adversarial example to its hosting benign file. Table 2 shows the number of adversarial examples obtained in this way in each case both for the ARM and the MIPS samples.

In this robustness study, we did not use the weekly analysis that we used in the performance evaluation of SIMBIO TA and SIMBIO TA-ML. Instead, for SIMBIO TA, we simply took the dominating set calculated on the last week, and for SIMBIO TA-ML, we took the random forest model trained on the last week. In other words, in both cases, we used a classifier constructed or trained on the largest amount of data available to the antivirus company from its intelligence network.

ARM			
Target TLSH diff	S	M	L
40	4,732	2,022	237
30	4,195	1,618	138
MIPS			
Target TLSH diff	S	M	L
40	10,025	3,630	1,877
30	9,265	3,240	1,630

Table 2: 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.

We presented the adversarial examples that we created to these classifiers and measured their accuracy. Similar to the performance evaluation, we repeated the whole experiment 12 times, and we show the box plot of the accuracy results obtained in the 12 runs in Figures 10 and 11 for strategy Chunker and strategy Disguiser, respectively.

As we can see in Figures 10, SIMBIO-TA-ML is more robust against strategy Chunker than SIMBIO-TA is, achieving much higher accuracies in all cases. 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 SIMBIO-TA’s) for small samples. What is surprising, though, is that the accuracy of SIMBIO-TA-ML does not decrease significantly even when the TLSH difference between the adversarial examples and the original malware samples is at least 60, while in case of SIMBIO-TA, accuracy drops significantly in those cases. The reason is that SIMBIO-TA directly compares the TLSH values of the adversarial examples to the TLSH values in its dominating set, and it uses the TLSH difference threshold of 40 for detection. Clearly, the larger the TLSH difference between the adversarial examples and known malware is, the harder it is for SIMBIO-TA to successfully detect them as malware.

Unfortunately, the robustness of SIMBIO-TA-ML is not preserved at all against strategy Disguiser: as shown in Figure 11, both SIMBIO-TA and SIMBIO-TA-ML achieve very poor accuracy on the adversarial examples constructed with this strategy. Finally, there does not seem to be any significant difference in the results in the ARM and the MIPS cases.

## 6 CONCLUSION

In this paper, which is an extended version of (Papp et al., 2022), we proposed SIMBIO-TA-ML, a light-weight, machine learning-based malware detection

approach for embedded IoT devices. Our work was inspired by SIMBIO-TA (Tamás et al., 2021), which uses TLSH hashes to detect malware based on binary similarity of new files to known malicious binaries. The key difference between SIMBIO-TA-ML and SIMBIO-TA is that we use TLSH hashes as feature vectors to train a random forest classifier, instead of directly measuring the TLSH similarity of files, and by doing so, we achieve a better malware detection performance than that of SIMBIO-TA.

More specifically, we showed via an extensive experiment on a large dataset of real IoT malware and benign files that SIMBIO-TA-ML consistently achieves a higher true positive detection rate than SIMBIO-TA does, while, at the same time, it also has a higher, but still acceptable, false positive detection rate. In terms of storage requirements, SIMBIO-TA is superior to SIMBIO-TA-ML, but SIMBIO-TA-ML can still be hosted by mid-range and high-end embedded devices with megabytes of memory. Finally, we also showed that the run time delay SIMBIO-TA introduces into the operation of an embedded IoT device is not constant, making it hard to design for. In contrast, SIMBIO-TA-ML introduces a near-constant, although somewhat increased, delay into the operation of the embedded IoT device, which is advantageous when the device has to satisfy real-time constraints.

We also studied the robustness of SIMBIO-TA-ML and SIMBIO-TA to two adversarial strategies aiming at the creation of malware samples that are likely misclassified by our similarity-based malware detectors. We found that SIMBIO-TA-ML is robust against one of the strategies, but it can be easily misled by the other, whereas SIMBIO-TA achieves poor robustness against both adversarial strategies. This means that, while SIMBIO-TA-ML is more robust than SIMBIO-TA is, there is still much room for improvement of SIMBIO-TA-ML too.

Our results also demonstrate that achieving robustness against adversarial examples is a challenging problem. Solving this open issue requires further effort from the research community. In our future work, we plan to improve the robustness of SIMBIO-TA-ML against adversarial examples by splitting samples into smaller parts both at the training and the testing phases, and by performing detection on smaller parts of a new file. We hope that, in this way, SIMBIO-TA-ML will eventually be able to detect if any part of a new file is similar to a known malware. Such an improved detection algorithm would achieve better robustness not only against strategy Disguiser, but against any strategy that hides a known malware binary in a benign file or adds extra bytes to a known malware binary.



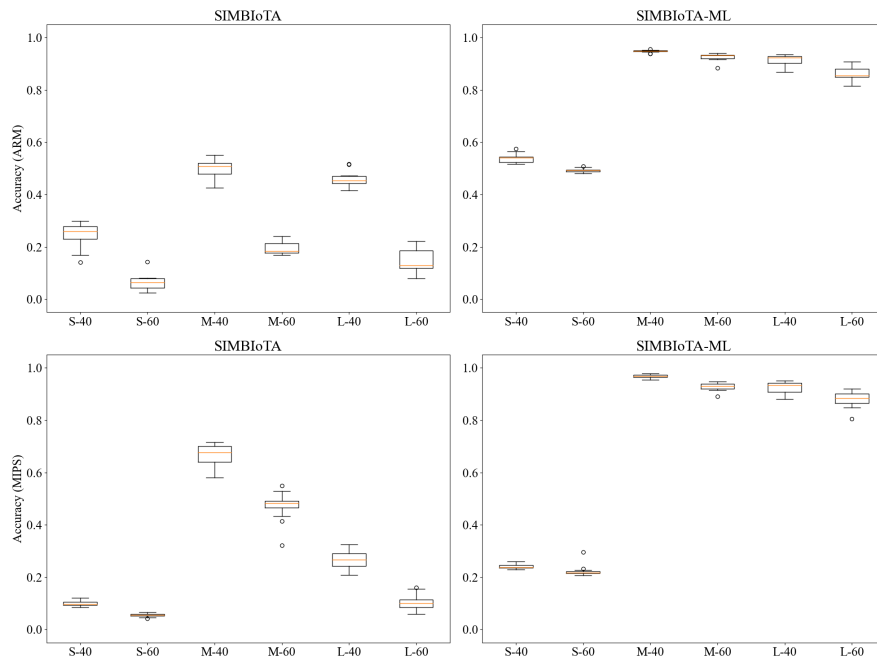


Figure 10: Comparison of the robustness of SIMBIO TA (left) and SIMBIO TA-ML (right) against adversarial examples generated with strategy **Chunker** for ARM (top) and MIPS (bottom) samples. The different cases that were analyzed are denoted by a combination of letters S, M, L (indicating the size of the malware samples from which the adversarial examples were created) and numbers 40, 60 (indicating the target TLSH difference used by the strategies). These cases are shown on the  $x$ -axis, while the  $y$ -axis shows the accuracy values.

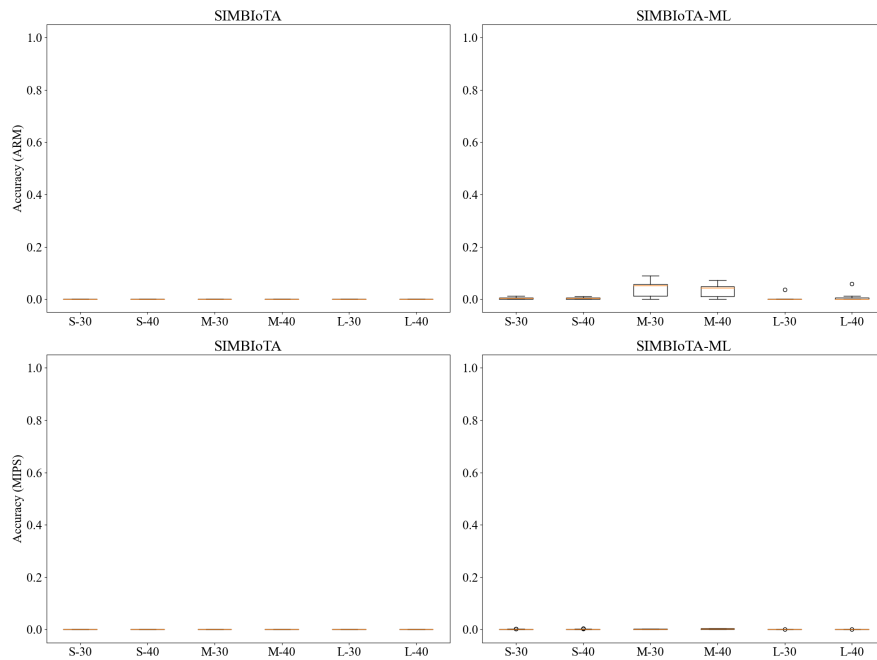


Figure 11: Comparison of the robustness of SIMBIO TA (left) and SIMBIO TA-ML (right) against adversarial examples generated with strategy **Disguiser** for ARM (top) and MIPS (bottom) samples. The different cases that were analyzed are denoted by a combination of letters S, M, L (indicating the size of the malware samples from which the adversarial examples were created) and numbers 30, 40 (indicating the target TLSH difference used by the strategies). These cases are shown on the  $x$ -axis, while the  $y$ -axis shows the accuracy values.

## 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 Office within the framework of the Artificial Intelligence National Laboratory Program. The authors would like to thank Zoltán Iuhos for his help in implementing the performance measurements.

## REFERENCES

- Abbas, M. F. B. and Srikanthan, T. (2017). Low-complexity signature-based malware detection for IoT devices. In Batten, L., Kim, D. S., Zhang, X., and Li, G., editors, *Applications and Techniques in Information Security*, pages 181–189, Singapore. Springer Singapore.
- Alasmay, H., Khormali, A., Anwar, A., Park, J., Choi, J., Abusnaina, A., Awad, A., Nyang, D., and Mohaisen, A. (2019). Analyzing and detecting emerging Internet of Things malware: A graph-based approach. *IEEE Internet of Things Journal*, 6(5):8977–8988.
- Anderson, H. S., Kharkar, A., Filar, B., Evans, D., and Roth, P. (2018). Learning to evade static PE machine learning malware models via reinforcement learning. arXiv:1801.08917.
- Anderson, H. S., Kharkar, A., Filar, B., and Roth, P. (2017). Evading machine learning malware detection. In *BlackHat USA*.
- Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J. A., Invernizzi, L., Kallitsis, M., Kumar, D., Lever, C., Ma, Z., Mason, J., Menscher, D., Seaman, C., Sullivan, N., Thomas, K., and Zhou, Y. (2017). Understanding the Mirai botnet. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1093–1110, Vancouver, BC. USENIX Association.
- Aryal, K., Gupta, M., and Abdelsalam, M. (2022). A survey on adversarial attacks for malware analysis. arXiv:2111.08223.
- Barreno, M., Nelson, B., Joseph, A., and Tygar, J. D. (2010). The security of machine learning. *Machine Learning*, 81:121–148.
- Buttyán, L., Nagy, R., and Papp, D. (2022). SIMBioTA++: Improved similarity-based IoT malware detection. In *Proceedings of the IEEE Conference on Information Technology and Data Science (CITDS)*.
- Cozzi, E., Vervier, P.-A., Dell’Amico, M., Shen, Y., Bigle, L., and Balzarotti, D. (2020). The tangled genealogy of IoT malware. In *Annual Computer Security Applications Conference (ACSAC2020)*, Austin, USA.
- Dovom, E. M., Azmoodeh, A., Dehghantanha, A., Newton, D. E., Parizi, R. M., and Karimipour, H. (2019). Fuzzy pattern tree for edge malware detection and categorization in IoT. *Journal of Systems Architecture*, 97:1–7.
- Gibert, D., Mateu, C., and Planes, J. (2020). The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications*, 153:102526.
- Goyal, M., Sahoo, I., and Geethakumari, G. (2019). 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.
- HaddadPajouh, H., Dehghantanha, A., Khayami, R., and Choo, K.-K. R. (2018). A deep recurrent neural network based approach for Internet of Things malware threat hunting. *Future Generation Computer Systems*, 85:88–96.
- Hu, W. and Tan, Y. (2017a). Black-box attacks against RNN based malware detection algorithms. arXiv:1705.08131.
- Hu, W. and Tan, Y. (2017b). Generating adversarial malware examples for black-box attacks based on GAN. arXiv:1702.05983.
- Hussain, F., Hussain, R., Hassan, S. A., and Hossain, E. (2020). Machine learning in IoT security: Current solutions and future challenges. *IEEE Communications Surveys & Tutorials*, 22(3):1686–1721.
- Hwang, C., Hwang, J., Kwak, J., and Lee, T. (2020). Platform-independent malware analysis applicable to Windows and Linux environments. *Electronics*, 9(5).
- Karanja, E. M., Masupe, S., and Jeffrey, M. G. (2020). Analysis of Internet of Things malware using image texture features and machine learning techniques. *Internet of Things*, 9:100–153.
- Kolosnjaji, B., Demontis, A., Biggio, B., Maiorca, D., Giacinto, G., Eckert, C., and Roli, F. (2018). Adversarial malware binaries: Evading deep learning for malware detection in executables. In *26th European Signal Processing Conference (EUSIPCO)*, pages 533–537.
- Kreuk, F., Barak, A., Aviv-Reuven, S., Baruch, M., Pinkas, B., and Keshet, J. (2019). Deceiving end-to-end deep learning malware detectors using adversarial examples. arXiv:1802.04528v3.
- Meidan, Y., Bohadana, M., Mathov, Y., Mirsky, Y., Shabtai, A., Breitenbacher, D., and Elovici, Y. (2018). N-BaIoT — network-based detection of IoT botnet attacks using deep autoencoders. *IEEE Pervasive Computing*, 17(3):12–22.
- Nakhodchi, S., Upadhyay, A., and Dehghantanha, A. (2020). A comparison between different machine learning models for IoT malware detection. In Karimipour, H., Srikantha, P., Farag, H., and Wei-Kocsis, J., editors, *Security of Cyber-Physical Systems: Vulnerability and Impact*, pages 195–202, Cham. Springer International Publishing.
- Ngo, Q.-D., Nguyen, H.-T., Le, V.-H., and Nguyen, D.-H. (2020). A survey of IoT malware and detection meth-

- ods based on static features. *ICT Express*, 6(4):280–286.
- Nguyen, H., Ngo, Q., and Le, V. (2020). A novel graph-based approach for IoT botnet detection. *Int. J. Inf. Sec.*, 19(5):567–577.
- Ojo, M. O., Giordano, S., Procissi, G., and Seitaniadis, I. N. (2018). A review of low-end, middle-end, and high-end IoT devices. *IEEE Access*, 6:70528–70554.
- Oliver, J., Cheng, C., and Chen, Y. (2013). TLSH – A Locality Sensitive Hash. In *2013 Fourth Cybercrime and Trustworthy Computing Workshop*, pages 7–13, Sydney NSW, Australia. IEEE.
- Papp, D., Ács, G., Nagy, R., and Buttyán, L. (2022). 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.
- Park, D., Khan, H., and Yener, B. (2019). Generation and evaluation of adversarial examples for malware obfuscation. In *18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pages 1283–1290.
- Shobana, M. and Poonkuzhali, S. (2020). A novel approach to detect IoT malware by system calls using deep learning techniques. In *2020 International Conference on Innovative Trends in Information Technology (ICITIT)*, pages 1–5.
- Soliman, S. W., Sobh, M. A., and Bahaa-Eldin, A. M. (2017). Taxonomy of malware analysis in the IoT. In *2017 12th International Conference on Computer Engineering and Systems (ICCES)*, pages 519–529.
- Song, W., Li, X., Afroz, S., Garg, D., Kuznetsov, D., and Yin, H. (2021). MAB-Malware: A reinforcement learning framework for attacking static malware classifiers. arXiv:2003.03100v3.
- Su, J., Vasconcellos, D. V., Prasad, S., Sgandurra, D., Feng, Y., and Sakurai, K. (2018). 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.
- Suciu, O., Coull, S. E., and Johns, J. (2019). Exploring adversarial examples in malware detection. In *IEEE Security and Privacy Workshops (SPW)*.
- Sun, H., Wang, X., Buyya, R., and Su, J. (2017). CloudEyes: Cloud-based malware detection with reversible sketch for resource-constrained Internet of Things (IoT) devices. *Software: Practice and Experience*, 47(3):421–441.
- Takase, H., Kobayashi, R., Kato, M., and Ohmura, R. (2020). A prototype implementation and evaluation of the malware detection mechanism for IoT devices using the processor information. *International Journal of Information Security*, 19.
- Tamás, C., Papp, D., and Buttyán, L. (2021). 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. INSTICC, SciTePress.
- Ucci, D., Aniello, L., and Baldoni, R. (2019). Survey of machine learning techniques for malware analysis. *Computers & Security*, 81:123 – 147.
- Ye, Y., Li, T., Adjeroh, D., and Iyengar, S. S. (2017). A survey on malware detection using data mining techniques. *ACM Computing Surveys*, 50(3).