SIMBIoTA-ML: Light-weight, Machine Learning-based Malware Detection for Embedded IoT Devices

Dorottya Papp[1][a], Gergely Ács[1][b], Roland Nagy[1][c], and Levente Buttyán[1,2][d]

[1]*CrySyS Lab, Budapest University of Technology and Economics, Műegyetem rkp. 3., H-1111 Budapest, Hungary*
[2]*ELKH-BME Information Systems Research Group, Műegyetem rkp. 3., H-1111 Budapest, Hungary*
{*dpapp, acs, roland.nagy, buttyan*}*@crysys.hu*

Keywords: IoT, embedded systems, malware detection, machine learning.

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 present SIMBIoTA-ML, a light-weight antivirus solution that enables embedded IoT devices to take advantage of machine learning-based malware detection. 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, a property that is advantageous in real-time applications, notably in the domain of cyber-physical systems.

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 a recent paper (Tamás. et al., 2021), we proposed SIMBIoTA (SIMilarity Based IoT Antivirus), an effective and efficient antivirus solution for such devices. The operating principles of SIMBIoTA are similar to those of traditional signature-based antivirus solutions, but SIMBIoTA 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, and it is different from cryptographic hashes, as it is designed to maximize collisions. This means that small variations in the input do not alter the TLSH output

[a] https://orcid.org/0000-0002-9976-614X
[b] https://orcid.org/0000-0003-4437-0110
[c] https://orcid.org/0000-0003-2305-3271
[d] https://orcid.org/0000-0003-4233-2559

significantly. In other words, similar inputs result in similar TLSH hash values, and SIMBIoTA takes advantage of this feature. More specifically, in case of SIMBIoTA, 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 an unknown file is similar to that of a known malware, the unknown file is detected as malware. The main advantage of SIMBIoTA is 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), SIMBIoTA 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 SIMBIoTA. 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. More specifically, our contributions, in this paper, are the following:

- We introduce SIMBIoTA-ML, which replaces SIMBIoTA'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 SIMBIoTA-ML, as well as its storage requirements and running time.

- We compare SIMBIoTA-ML's measurement results to those of SIMBIoTA and discuss the advantages and disadvantages of both solutions. Specifically, we find that SIMBIoTA has lower storage requirements and false positive detection rate, but SIMBIoTA-ML outperforms SIMBIoTA in terms of true positive detection rate even for new, previously unseen malware samples. We also show that SIMBIoTA'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 SIMBIoTA to decide whether an unknown file is malicious or not. By contrast, we show that SIMBIoTA-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.

The paper is structured as follows: Section 2 provides background information on malware detection approaches and SIMBIoTA. Section 3 discusses SIMBIoTA-ML and our changes to SIMBIoTA's architecture in order to use machine learning. The performance of SIMBIoTA-ML is evaluated in Section 4. Finally, Section 5 concludes the paper.

2 RELATED WORK

In this section, we provide background information on machine learning-based malware detection, 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[1], 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; Gibert 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), their 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;

---

[1] https://www.sophos.com/en-us/medialibrary/pdfs/technical-papers/sophoslabs-2019-threat-report.pdf (accessed: Feb 28, 2022)

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 SIMBIoTA

SIMBIoTA 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. SIMBIoTA 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 described above. 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 light-weight algorithm uses them to detect malware based on binary similarity. Therefore, SIMBIoTA requires resource-constrained embedded IoT devices to store only a small database with a few TLSH hash values.

In (Tamás. et al., 2021), SIMBIoTA 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[2], the samples were also put into so-called "weekly batches", i.e., sets of samples that were first submitted to Virus-Total 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 SIMBIoTA'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, SIMBIoTA 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 SIMBIoTA. Second, as we show in this paper, the bigger the database of similarity hash values, the longer it takes for SIMBIoTA 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 SIMBIoTA is hard to design for. Last, even though a true positive detection rate of 90% on average for new, previously 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 SIMBIoTA's architecture

---

[2]https://www.virustotal.com/ (accessed: Jan 8, 2022)

to enable embedded IoT devices to take advantage of machine learning-based malware detection. We call the resulting system SIMBIoTA-ML. 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 SIMBIoTA with machine learning-based malware detection. We discuss our modifications to SIMBIoTA's architecture in Section 3.1 and our design choices for machine learning in Section 3.2. We call the resulting antivirus solution SIMBIoTA-ML.

3.1 Architectural overview

The original architecture of SIMBIoTA 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 SIMBIoTA is to replace the dominating set construction by 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 SIMBIoTA'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 $q1$, $q2$, and $q3$ 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[3]:

- the first byte is a checksum value;

- the second byte stores the so-called L value, which is calculated from the length of the original byte sequence;

---

[3]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.
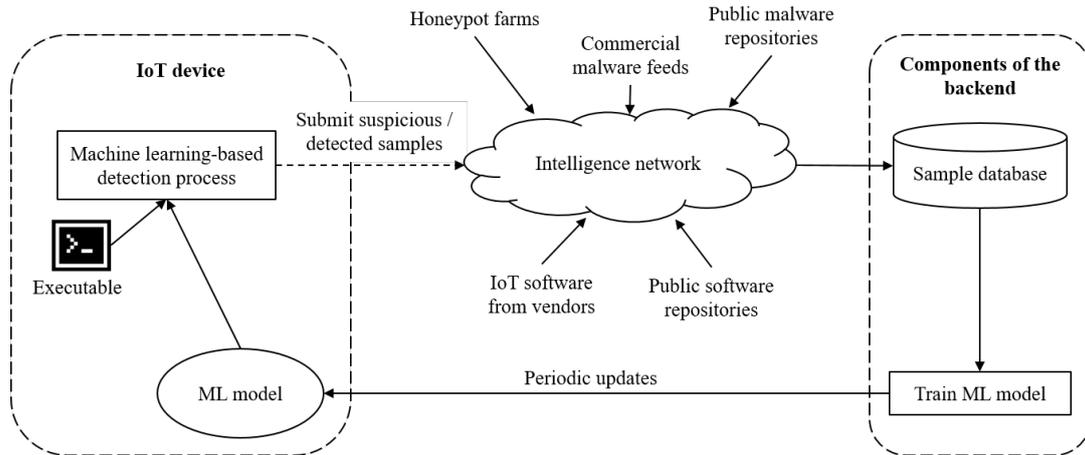
Figure 1: Architecture of SIMBIoTA-ML

- the 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 EVALUATION

In this section, we compare SIMBIoTA-ML to SIM-BIoTA and discuss their advantages and disadvantages. Specifically, we discuss the experiment design and the used data set in Section 4.1. Sections 4.2 and 4.3 present the true positive and false positive detection rates, respectively. We compare the two solutions' storage requirements in Section 4.4 and their running times in Section 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[4] 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.

---

[4]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 SIMBIoTA. The timeline of the experiment is between January 1st, 2018 and September 15th, 2019, divided into weeks. We assume that both SIMBIoTA and SIMBIoTA-ML receive updates for their detection methods at the beginning of each week. Malicious samples are organized into weekly batches based on the date they were first seen, 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.

SIMBIoTA-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 SIMBIoTA-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 SIMBIoTA-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 SIMBIoTA. For SIMBIoTA-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 SIMBIoTA-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 SIMBIoTA and SIMBIoTA-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 SIMBIoTA and the right-hand side shows the performance of SIMBIoTA-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, SIMBIoTA-ML consistently outperforms SIMBIoTA 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 SIMBIoTA and SIMBIoTA-ML. The left-hand side depicts the performance of SIMBIoTA and the right-hand side shows the performance of SIMBIoTA-ML. SIMBIoTA's performance varies in time and it is only by the second half of the experiment that its performance reaches 90-95%. SIMBIoTA-ML also shows
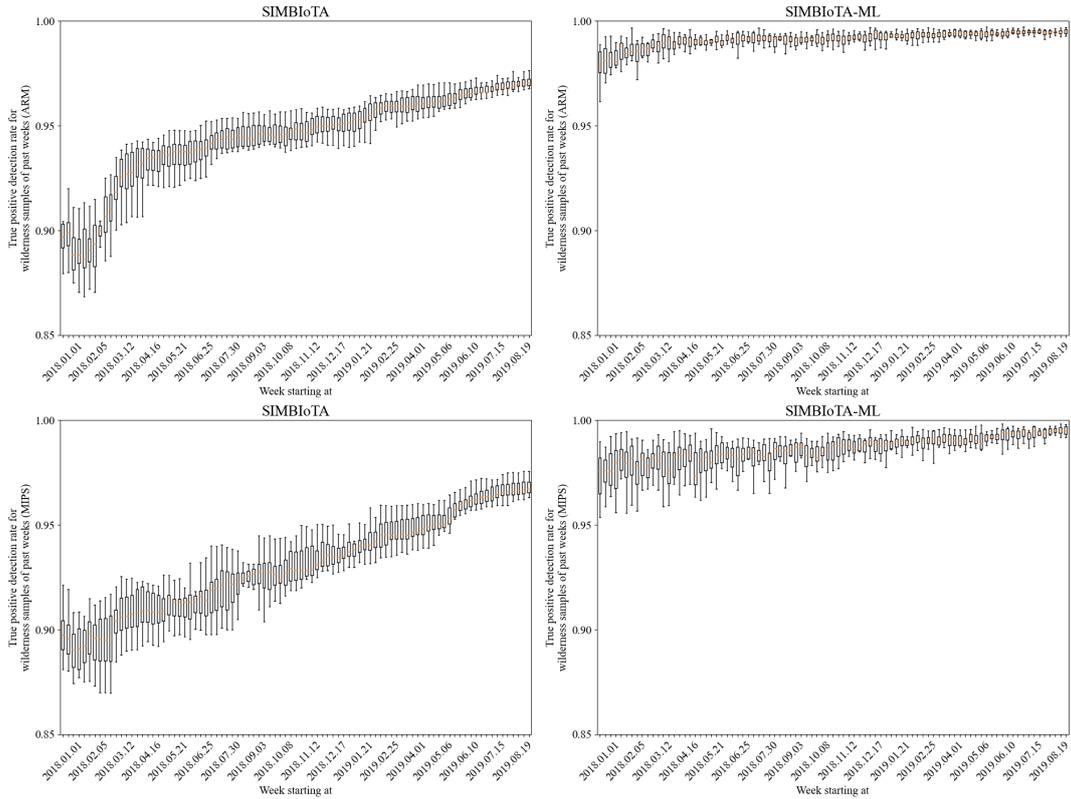
Figure 2: Box plot of the true positive detection rate for samples of the past for SIMBIoTA and SIMBIoTA-ML

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

4.3 False positive detection rate

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

In our experiments, SIMBIoTA 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 SIMBIoTA-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, SIMBIoTA-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 Section 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 SIMBIoTA-ML's false positive detection rate is higher than that of SIMBIoTA, it is still acceptable for malware detection.
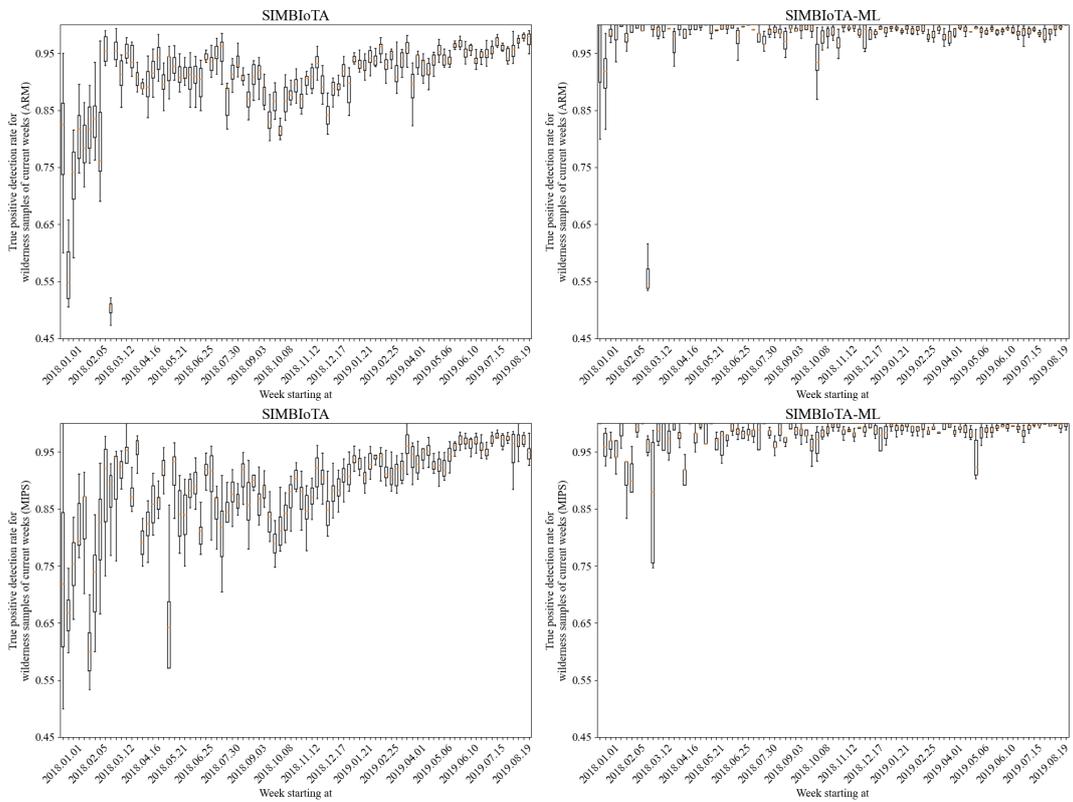
Figure 3: Box plot of the true positive detection rate for previously unseen samples for SIMBIoTA and SIMBIoTA-ML
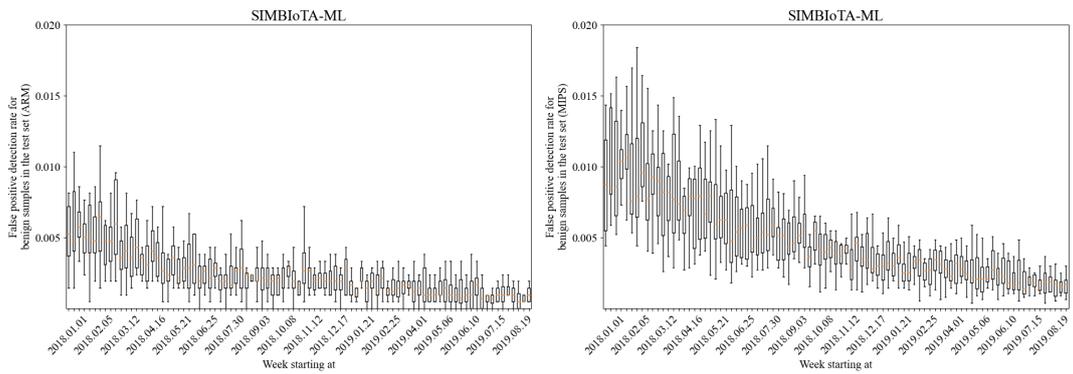

Figure 4: Box plot of the false positive detection rate for benign samples in the test set for SIMBIoTA-ML

4.4 Storage requirement

Throughout our experiments, we measured the amount of storage necessary to hold SIMBIoTA's database of similarity hashes and SIMBIoTA-ML's machine learning model. In the case of SIMBIoTA, 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 SIMBIoTA-ML, our implementation for the random forest classifier uses the scikit-learn[5] Python module. In order to measure the amount of storage necessary to hold the model, we used the pickle[6] 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 SIMBIoTA and SIMBIoTA-ML are shown in Figure 5. While the storage requirements of both antivirus solutions increase over time, SIMBIoTA-ML's requirements are orders of magnitude higher, going form ca. 40 KB to ca. 150 KB by the end of our experiment. By contrast, SIMBIoTA's database of similarity hashes require less than 10 KB of storage throughout the experiment. Therefore, we may conclude that SIMBIoTA-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 SIMBIoTA-ML.

4.5 Run time performance

The last aspect by which we compare SIMBIoTA and SIMBIoTA-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 SIMBIoTA and SIMBIoTA-ML for determining that a submitted file is malicious is shown in Figure 6. SIMBIoTA's performance microseconds as it only needs to calculate the difference between TLSH hashes and compare the

---

[5]https://scikit-learn.org/stable/ (accessed: Jan 11, 2022)
[6]https://docs.python.org/3/library/pickle.html (accessed: Jan 11, 2022)

result to a threshold value. However, SIMBIoTA 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, SIMBIoTA'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.

SIMBIoTA-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 SIMBIoTA, 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 SIMBIoTA and SIMBIoTA-ML for determining that a submitted file is benign is shown in Figure 7. The run time delay that SIMBIoTA's detection process would cause on a real system is even higher in this case. The reason for this is that in order for SIMBIoTA'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. SIMBIoTA-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 CONCLUSION

In this paper, we proposed SIMBIoTA-ML, a lightweight, machine learning-based malware detection approach for embedded IoT devices. Our work was inspired by SIMBIoTA (Tamás. et al., 2021), which uses TLSH hashes to detect malware based on binary similarity of unknown files to known malicious binaries. The key difference between SIMBIoTA-ML and SIMBIoTA 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 SIMBIoTA. More specifically, we showed via an exten-
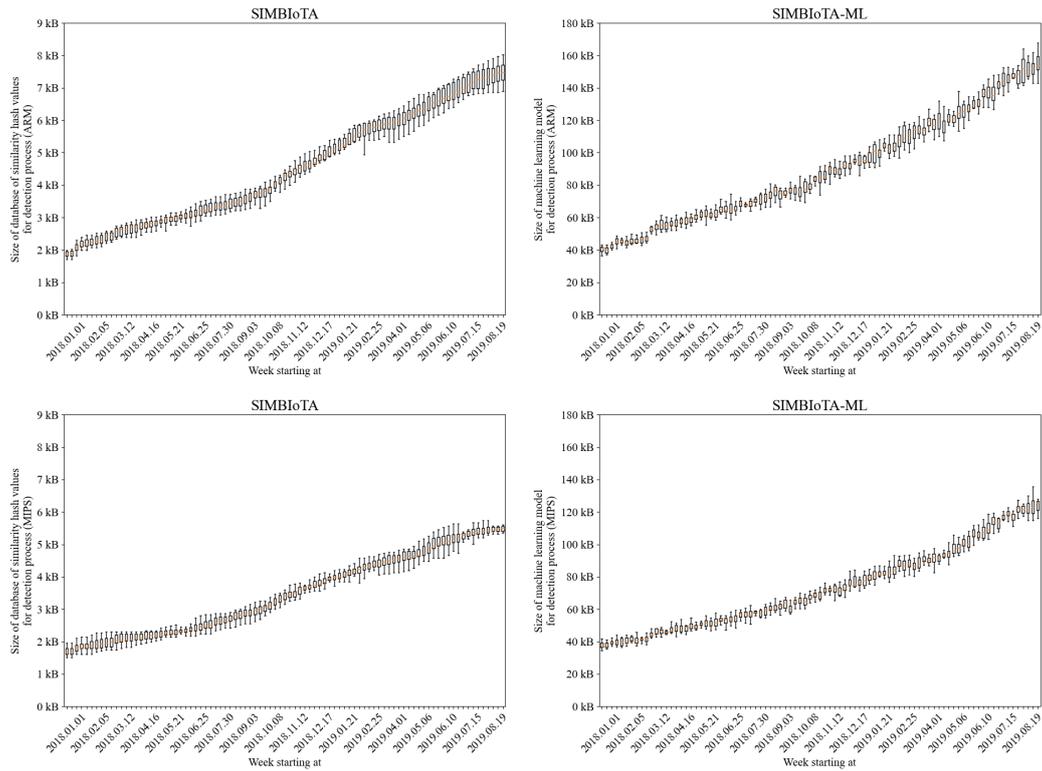
Figure 5: Box plot of the storage requirements for SIMBIoTA and SIMBIoTA-ML

sive experiment on a large dataset of real IoT malware and benign files that SIMBIoTA-ML consistently achieves a higher true positive detection rate than SIMBIoTA, while, at the same time, it also has a higher, but still acceptable, false positive detection rate. In terms of storage requirements, SIMBIoTA is superior to SIMBIoTA-ML, but SIMBIoTA-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 SIMBIoTA introduces into the operation of an embedded IoT device is not constant, making it hard to design for. In contrast, SIMBIoTA-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.

ACKNOWLEDGEMENTS

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.

Alasmary, 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.

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*
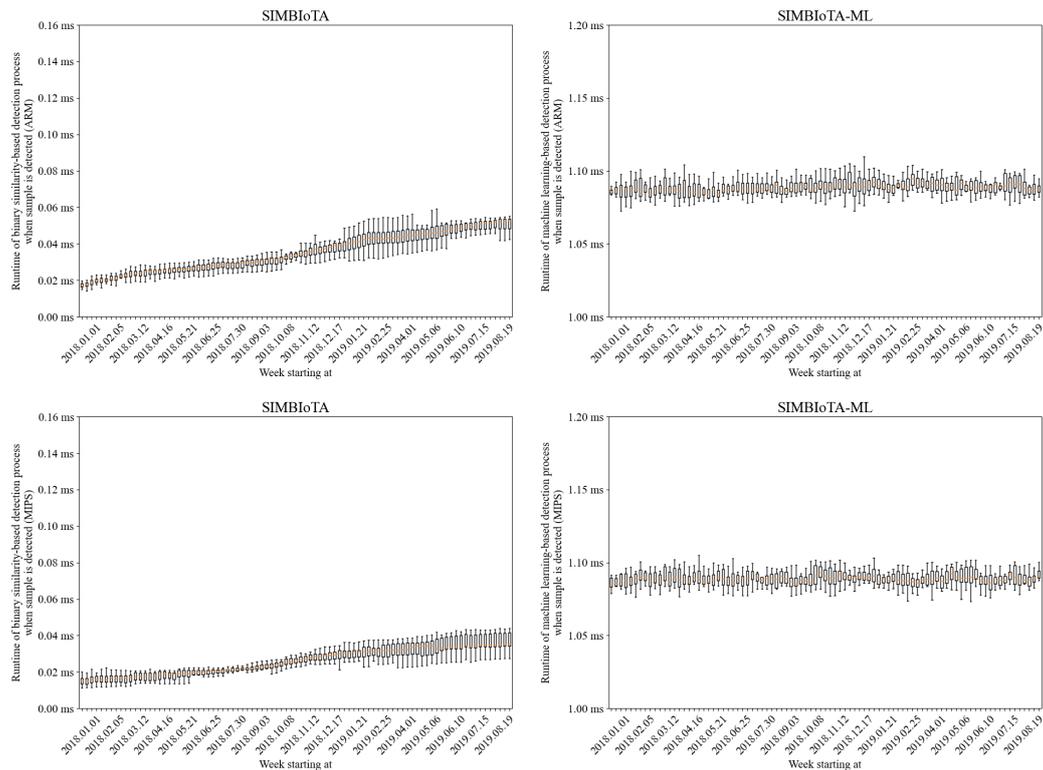
Figure 6: Box plot of the run time of the detection process for "malicious" decision for SIMBIoTA and SIMBIoTA-ML

Symposium (USENIX Security 17), pages 1093–1110, Vancouver, BC. USENIX Association.

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.

Hussain, F., Hussain, R., Hassan, S. A., and Hossain, E. (2020). Machine learning in IoT security: Current so-

lutions 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*, 9:100153.

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*, pages 195–202. Springer International Publishing, Cham.

Ngo, Q.-D., Nguyen, H.-T., Le, V.-H., and Nguyen, D.-H. (2020). A survey of iot malware and detection methods 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 Seitanidis, I. N. (2018). A review of low-end, middle-end, and high-end IoT devices. *IEEE Access*, 6:70528–70554.
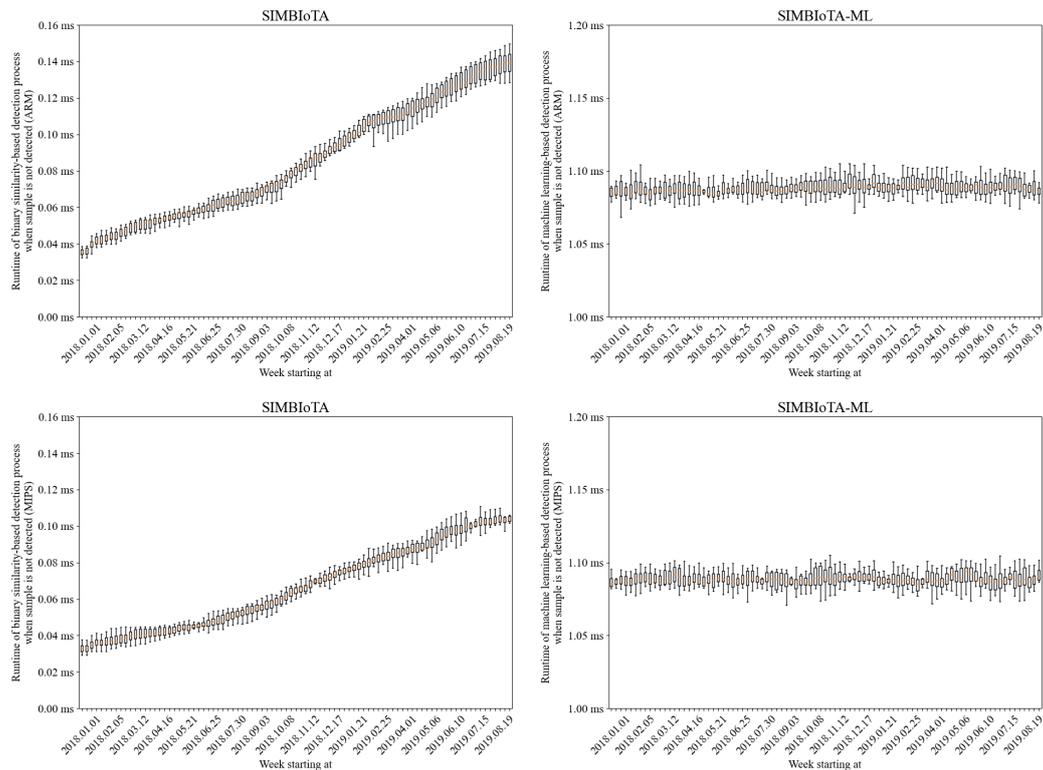
Figure 7: Box plot of the run time of the detection process for "benign" decision for SIMBIoTA and SIMBIoTA-ML

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.

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 (ICITIIT)*, 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.

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.

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).