Increasing the Robustness of a Machine Learning-based IoT Malware Detection Method with Adversarial Training

József Sándor
CrySyS Lab
Budapest University of Technology and Economics
Budapest, Hungary
jsandor@crysys.hu

Roland Nagy
CrySyS Lab
Budapest University of Technology and Economics
Budapest, Hungary
rnagy@crysys.hu

Levente Buttyán
CrySyS Lab
Budapest University of Technology and Economics
Budapest, Hungary
buttyan@crysys.hu

ABSTRACT

We study the robustness of SIMBIoTA-ML, a recently proposed machine learning-based IoT malware detection solution against adversarial samples. First, we propose two adversarial sample creation strategies that modify existing malware binaries by appending extra bytes to them such that those extra bytes are never executed, but they make the modified samples dissimilar to the original ones. We show that SIMBIoTA-ML is robust against the first strategy, but it can be misled by the second one. To overcome this problem, we propose to use adversarial training, i.e., to extend the training set of SIMBIoTA-ML with samples that are crafted by using the adversarial evasion strategies. We measure the detection accuracy of SIMBIoTA-ML trained on such an extended training set and show that it remains high both for the original malware samples and for the adversarial samples.

CCS CONCEPTS

• **Security and privacy** → **Malware and its mitigation**; • **Computing methodologies** → **Machine learning**.

KEYWORDS

Internet-of-Things; malware detection; machine learning; adversarial examples; adversarial training

ACM Reference Format:
József Sándor, Roland Nagy, and Levente Buttyán. 2023. Increasing the Robustness of a Machine Learning-based IoT Malware Detection Method with Adversarial Training. In *Proceedings of the 2023 ACM Workshop on Wireless Security and Machine Learning (WiseML '23), June 1, 2023, Guildford, United Kingdom.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3586209.3591401

1 INTRODUCTION

The Internet-of-Things (IoT) consists of embedded computers connected to the Internet. In the recent past, there has been a dramatic increase in the number of deployed embedded IoT devices, and exciting new IoT applications emerged in various domains, including manufacturing, transportation, healthcare, and agriculture. Unfortunately, with the proliferation of IoT devices, the number of known attacks against them has also increased. These attacks include infection by malware. The best-known example for this is probably Mirai [3], a malware that infected hundreds of thousands of IoT devices and launched one of the largest distributed denial-of-service attacks against Internet-based services in 2016. Since then, the IoT threat landscape has been extended with other malware families as well, such as Gafgyt, Tsunami, and Dnsamp [6].

Malware detection is an essential part of modern defense mechanisms used in computer-based systems. Unfortunately, no matter how good a malware detection system is, attackers constantly work on methods to evade it. In particular, they want to construct malware samples that have the same function as older samples, but not recognized by detectors. Such kind of samples are called adversarial examples. The degree of resistance against adversarial examples defines the robustness of the malware detection system. History shows that traditional signature-based and heuristic malware detection are not robust against adversarial samples, which explains the large number of polymorphic malware. And it has been shown in the literature [4, 16] that, unfortunately, machine learning-based (ML-based) malware detectors can also be misled easily.

In this paper, we examine the robustness of SIMBIoTA-ML, which is a recent ML-based IoT malware detection solution [13]. We design two adversarial example creation strategies that 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 chunks of the original sample to the malware and ensures that a certain target difference is achieved by doing so. The second strategy embeds a malware into a known benign file and ensures that the resulting sample becomes similar to the benign file (and hence dissimilar to the original malware sample). We show that SIMBIoTA-ML is robust against the first strategy, but it can be misled by the second one. To overcome this problem, we propose to use adversarial training as the main contribution of this paper. Adversarial training has been used in the image recognition domain to increase the robustness of ML-based models against adversarial examples. We adopt this approach in the domain of malware detection and demonstrate its effectiveness. Adversarial training in our case means that the training set of the malware detector algorithm is extended with samples that are crafted by using the adversarial evasion strategies that we proposed. We measure the detection accuracy of SIMBIoTA-ML trained on such an extended training set and show that it remains high both for the original malware samples and for the adversarial

WiseML '23, June 1, 2023, Guildford, United Kingdom

ACM ISBN 979-8-4007-0133-7/23/06…$15.00
https://doi.org/10.1145/3586209.3591401

samples. The price that we have to pay for this robustness is the increased training time and the increased size of the detection model, however, we argue that both are bearable in practice.

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

2 SIMBIOTA-ML

Although, IoT malware detection is a challenging problem, there have been solutions proposed in the literature, like [17]. In this work, we are interested in SIMBIoTA-ML (SIMilarity Based IoT Antivirus with Machine Learning) [13], an IoT malware detection method that has been proposed recently, and that achieved a remarkable malware detection performance, while remaining resource efficient at the same time. Before presenting the operation of this solution, it is useful to review the concept of similarity hashes, as it forms the basis of SIMBIoTA-ML.

2.1 The TLSH similarity hash function

Cryptographic hash functions, such as SHA256, produce completely different hash values even for similar inputs. This makes them suitable for many security applications. Similarity hash functions are, however, different: they output similar hash values for similar inputs. SIMBIoTA-ML uses the TLSH similarity hash function [12] for extracting features from executable files (so called binaries). Indeed, the TLSH hash of a binary can be computed very efficiently and in a completely automated way. The hash calculation time is in the range of milliseconds on contemporary personal computers, which means that it should be lightweight enough even for resource constrained IoT devices. The TLSH hash value is also relatively short, it can be represented in 36 bytes, which is another advantage. Moreover, one can measure the similarity of two binaries by computing their TLSH difference score, an integer reflecting the dissimilarity of the inputs. The minimum TLSH difference score is 0, which means that the two inputs are almost identical, while a higher difference score means that the inputs are more dissimilar. For more details on the calculation of the TLSH hash value and the TLSH difference score, we refer the reader to [12]. It is important to note that similarity hash functions and difference score calculation algorithms operate only on raw byte sequences as inputs, therefore, they are suitable for capturing static byte level similarity of binaries and nothing more.

2.2 Operation of SIMBIoTA-ML

SIMBIoTA-ML is an IoT malware detection solution that was proposed in [13]. It saves storage, memory, computation, and bandwidth, which resources are constrained in the IoT field. According to the architectural view in Figure 1, SIMBIoTA-ML consists of 3 major components: IoT device (i.e. client), backend (i.e. server), and intelligence network. The intelligence network (e.g. honeypot farms, commercial malware feeds, and public malware/software repositories) provides malware and benign samples for the backend. Typically, thousands of samples are collected each day. The backend trains a random forest classifier (i.e. model) using the feature vectors extracted from the malware and benign samples. The feature vectors are derived from the TLSH hash of the binaries. The training of a machine learning model could be a resource intensive task, therefore, only the fully trained model is sent to the IoT devices. If the model on the server side is updated, then the clients download the update from the server. The clients use the stored machine learning model to decide locally about any new file if it is malicious or benign. With this solution, SIMBIoTA-ML has a true positive malware detection rate of ca. 95%, while having low false positive detection rate at the same time. Furthermore, SIMBIoTA-ML has the advantage with respect to other cloud-based malware detection systems that the clients can operate using their local model even if the backend is unavailable.

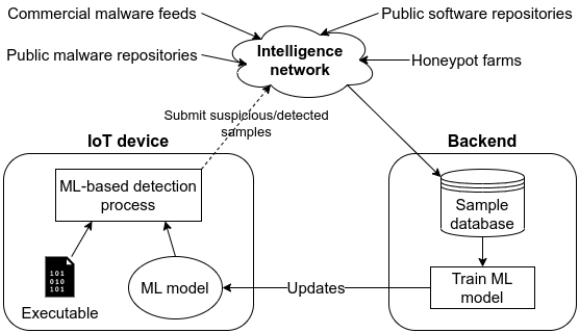

Figure 1: Architecture of SIMBIoTA-ML

3 STRATEGIES FOR CREATING ADVERSARIAL EXAMPLES

SIMBIoTA-ML effectively recognizes malicious files using an ML model trained on TLSH hash values of malware and benign binaries. But what if the attacker knows that SIMBIoTA-ML uses this technique? Could the attacker use this information to increase the chances of evading detection of his malware? In this section we are looking for answers to these questions.

In general, adversarial examples are those inputs that were specifically crafted to be misclassified by a given ML model [5]. In our case, this means the misclassification of malicious samples as benign files. To answer the questions asked at the beginning of the section, we have to think with the head of the attacker. First, we assume that we already have a fancy malware and we do not want to spoil its functionality and executability. The only problem is that SIMBIoTA-ML indicates correctly that it is malicious. However, we know that the similarity hash generation algorithms (which are also used by our detection systems) do not take into account the format of the inputs, they only consider raw sequences of bytes. Furthermore, we can even find out that this similarity hash function is the TLSH.

Our idea is that we can mislead the detection system if we could manipulate the TLSH hash value of our malware. Knowing the nature of TLSH, in order to do so, we have to modify the raw binary.

Targeted modification of the binary by manipulating the source code without spoiling the original functionality is not so trivial task. It would be much easier to add a few extra bytes to the end of the binary that actually will never be executed, but will change the TLSH hash value. However, we have to do this expansion of the binary carefully, because too much growth can be noticeable for the defenses system. For creating such adversarial examples, we developed two strategies. The first one is called Chunker, the second is called Disguiser. These represent two different approaches. In case of Chunker, we add to the malware chunks of itself and the goal is to increase the TLSH difference between the malware and the crafted adversarial example. In case of Disguiser, we want to hide our malware in a benign file, so the goal is to decrease the TLSH difference between the benign file and the adversarial example. Moreover, these strategies are simple enough to be easily implemented by an attacker in a real-world situation.

3.1 Strategy 1: Chunker

As stated in [19], an extensive empirical analysis showed that the TLSH difference score 40 is a good malware detection threshold. So, our intuition is that if the TLSH difference between the original malware and our crafted adversarial example is larger than 40, then SIMBIoTA-ML will likely misclassify it. Chunker's main idea is to simply add bytes to the end of the malware binary. The question is, how many and what kind of bytes need to be added to reach our goal. If all attached bytes are constant or random, the byte entropy would change and a static analyzer would easily detect it. It seems a reasonable solution to add some chunks from the original malware to itself. With this solution, if we choose properly the chunks, the byte entropy of the modified file will be almost the same as the original. The pseudocode of Chunker is presented as Algorithm 1.

Algorithm 1 *Chunker*

Input: malware binary MW
Output: adversarial example $ADVEX$
1: $CHUNKS \leftarrow$ split MW to 20 equal parts
2: $CH \leftarrow$ element form $CHUNKS$ w/ the closest entropy to MW
3: $ADVEX \leftarrow MW$
4: **repeat**
5: $\quad ADVEX \leftarrow ADVEX + CH$
6: **until** $TLSH\_difference(MW, ADVEX) > 40$
7: **return** $ADVEX$

We performed an empirical study to find out how many chunks need to be added to reach TLSH difference greater than 40. This showed that, in most cases, 4 chunks are enough, which results in only 20% growth in file size.

3.2 Strategy 2: Disguiser

We could also mislead the detection system, if we hide a malware inside of a benign file. So the Disguiser strategy concatenates a benign file to the end of a malware binary. Hence, when this file is executed, the malware will run, although the TLSH hash of the file may be determined by the added benign content. Here, our intuition is that a small malware in a large benign file can be hidden easier, because the TLSH difference between the benign content and the adversarial example will be small. So our goal is to keep this TLSH difference under the explained threshold of 40. According to our measurements, this requires that the size of the benign file is at least 5 times larger than the size of the malware. The pseudocode of Disguiser is presented as Algorithm 2.

Algorithm 2 *Disguiser*

Input: malware binary MW, pool of benign files BN_POOL
Output: adversarial examples $ADVEX\_POOL$
1: **for all** $BN \in BN\_POOL$ **do**
2: $\quad$ **if** $sizeof(MW)/sizeof(BN) < 0.2$ **then**
3: $\quad\quad ADVEX \leftarrow MW + BN$
4: $\quad\quad$ **if** $TLSH\_difference(BN, ADVEX) < 40$ **then**
5: $\quad\quad\quad$ add $ADVEX$ to $ADVEX\_POOL$
6: $\quad\quad$ **end if**
7: $\quad$ **end if**
8: **end for**
9: **return** $ADVEX\_POOL$

3.3 Measurement

In this work, we perform all experiments using the same dataset as used for the evaluation of SIMBIoTA-ML in [13]. This dataset is called CrySyS-Ukatemi benchmark dataset of IoT malware 2021 (or CUBE-MALIoT-2021 for short). The dataset consists of 29,209 malicious ARM samples and 18,715 malicious MIPS samples, extended with 4,727 benign ARM samples and 9,392 benign MIPS samples. For malicious samples, metadata is also available, which details, among others, the date the sample was first seen in the wild (i.e., submitted to VirusTotal). CUBE-MALIoT-2021 is publicly available[1] for use by the IoT malware research community.

According to our extensive measurements, adversarial examples created from this data set with strategy Chunker do not mislead SIMBIoTA-ML significantly (see Figure 3). On the other hand, the robustness of SIMBIoTA-ML against adversarial examples produced by strategy Disguiser is rather poor (see Figure 4). Basically, these adversarial examples are constructed by concatenating a malware and benign file in such a way that the size of the benign part is at least five times larger than the size of malware part. Thus, the TLSH values of these adversarial examples are more similar to the TLSH values of benign files than to the TLSH values of malware samples. Therefore, SIMBIoTA-ML misclassifies these adversarial examples as a benign file. Moreover, there does not seem to be any significant difference in the results in the ARM and the MIPS cases.

4 ADVERSARIAL TRAINING

In this section, we present a possible solution that antivirus companies could use to increase the robustness of existing malware detection systems against adversarial examples. In the previous section, we saw two possible methods that attackers could use to create adversarial examples that evade detection of SIMBIoTA-ML. SIMBIoTA-ML is somewhat robust against the Chunker strategy, but it can be misled by the Disguiser strategy. To overcome this problem, we propose to use SIMBIoTA-ML with adversarial training.

Adversarial training has been used in the image recognition domain to increase the robustness of ML-based models against adversarial examples. We adopt this approach in the domain of malware detection and demonstrate its effectiveness. Adversarial

[1] https://github.com/CrySyS/cube-maliot-2021 (accessed: April 22, 2023)

training in our case means that that the training set of the malware detector algorithm is extended with samples that are crafted by using the adversarial evasion strategies that we proposed.

4.1 Setup

To use adversarial training on SIMBIoTA-ML we have to extend the original training sample set with adversarial examples. Originally, SIMBIoTA-ML is trained on 10% of the malware dataset introduced in Subsection 3.3 (see [13]). The samples in the training set represent malware samples known to the antivirus company. Therefore, we construct adversarial examples for adversarial training from malware samples only from the training set, because the antivirus company has knowledge only about these files. After training SIMBIoTA-ML on this extended set of training samples, we test its performance on the original test set and adversarial examples generated from the test set. In the following, SIMBIoTA-ML is referred to as the upgraded SIMBIoTA-ML after adversarial training and the original SIMBIoTA-ML before adversarial training. Furthermore, we apply adversarial training separately in case of the Chunker and Disguiser strategies.

For adversarial training, we have to determine how many adversarial examples should be included in the training set. In case of Chunker this is somewhat simpler than in case of Disguiser, because the Chunker strategy creates one adversarial example from one malware. Therefore, in case of Chunker we use for training all adversarial examples created from malware files from the training set. In case of Disguiser, the number of adversarial examples created from a single malware can be larger, because the Disguiser strategy pairs one malware with all available benign files and selects the pairs that meet the constraints described in Subsection 3.2. To overcome this problem, we created the LooseDisguiser strategy that is similar to the Disguiser strategy: it pairs benign files with malicious files and creates an adversarial example from a pair if the ratio of the size of the malicious file to the size of the benign file is below 0.2. However, unlike the Disguiser strategy, the LooseDisguiser strategy does not consider the TLSH distance between the hosting benign file and the constructed adversarial example. The LooseDisguiser strategy has a so-called multiply factor parameter (instead of a TLSH threshold) that defines the maximum number of adversarial examples created from a malware. With LooseDisguiser, we can create a constant number of adversarial examples per malware.

Depending on how many adversarial examples are added to the training set, the accuracy of SIMBIoTA-ML changes on the test adversarial example set and the original test set. In case of Chunker, we use for training all adversarial examples created from malware files from the training set. In case of LooseDisguiser, we measure the accuracy of the upgraded SIMBIoTA-ML on the test adversarial example set and on the original test set with different multiply factors. In Figure 2 we see the results of this measurement. In this figure, the ideal point is (1,1) which means 100% accuracy on adversarial example test set and 100% on the original test set. In case of ARM samples, the point corresponding to multiply factor 4 is the closest to this ideal point, however, in case of MIPS samples this multiply factor is 2. Seemingly, SIMBIoTA-ML is more sensitive to noise (i.e., adversarial examples in the training set) in case of MIPS samples. This phenomenon requires further investigation and may

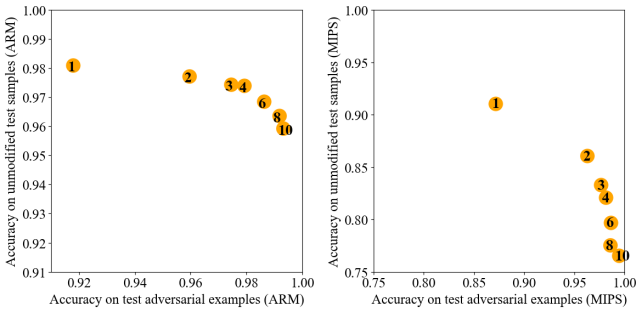

Figure 2: The effect of the multiply factor of the LooseDisguiser strategy on the accuracy of SIMBIoTA-ML w/ adversarial training, in the ARM and MIPS cases. The multiply factors are shown in the small circles, where each circle corresponds to a given combination of accuracy values.

Table 1: Number of elements in the train and test adversarial samples constructed w/ the Chunker, LooseDisguiser, and Disguiser strategies, in the ARM and MIPS cases.

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

lead to further research directions. To keep it simple, we choose multiply factor 4 for LooseDisguiser in case of both architectures.

The exact numbers of samples obtained in this way are shown in Table 1. In case of Disguiser, the train adversarial sample set is empty, because we use the adversarial examples of this strategy only for testing the original and the upgraded SIMBIoTA-ML. Similar to the experiment in [13], we repeated the adversarial training 12 times to eliminate the effects of randomly splitting the dataset into a 10% size training and 90% size testing part. Some cells of Table 1 contain intervals, rather than specific values, because the number of elements in the train and test sets may differ slightly in the 12 measurements.

4.2 Results

In this subsection, we present the results of adversarial training on SIMBIoTA-ML. We measure the detection accuracy of SIMBIoTA-ML trained on the extended training set and show that it remains high both for the original malware samples and for the adversarial samples[2].

In case of Chunker, on the left side of Figure 3, we see that both the original and the upgraded SIMBIoTA-ML have ca. 99% accuracy on the original test set. On the right side, we notice that the test adversarial examples of Chunker somewhat mislead the original SIMBIoTA-ML: its accuracy decreases to ca. 93%. At the same time, accuracy of the upgraded SIMBIoTA-ML remains high at

[2]Note that when testing only with adversarial examples, the accuracy of the model coincides with its recall

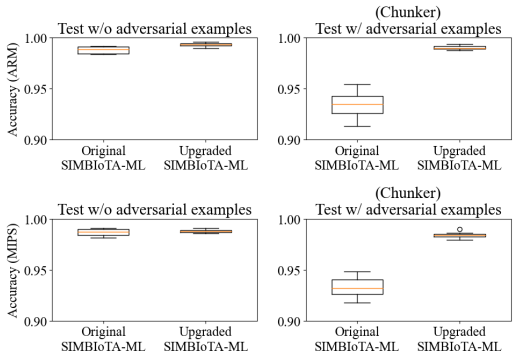

Figure 3: Comparison of the accuracy of the original and the upgraded SIMBIoTA-ML on the original test samples, and on the adversarial test samples constructed w/ Chunker strategy, in the ARM and MIPS cases.

ca. 99%. In Figure 4, we can see that SIMBIoTA-ML can be completely mislead by both the Disguiser and the LooseDisguiser strategies. After adversarial training with the samples of LooseDisguiser, the upgraded SIMBIoTA-ML has a significantly increased accuracy on the adversarial test set constructed with LooseDisguiser (ca. 97%). Moreover, the upgraded SIMBIoTA-ML, which was trained with the adversarial examples of LooseDisguiser, remains surprisingly robust against adversarial examples of Disguiser too. While the accuracy of SIMBIoTA-ML remarkably increases on adversarial examples after adversarial training, the accuracy of the upgraded SIMBIoTA-ML on the original test sample set is only slightly lower than the original SIMBIoTA-ML's accuracy.

4.3 Discussion

The antivirus company does not necessarily know the exact algorithm of the attacker; in fact, most of the time this is the case. Our setting models this scenario, because the antivirus company uses LooseDisguiser for training, but the attacker uses the (more powerful) Disguiser strategy. Nonetheless, the upgraded SIMBIoTA-ML, which was trained with the adversarial examples of LooseDisguiser, is surprisingly robust against adversarial examples of Disguiser too. The price that we have to pay for this robustness is the increased training time and the increased size of the detection model; however, we argue that both are bearable in practice. In Subsection 4.1, we saw that adversarial training requires an extended training set, and usually, an increased training set comes with increased training time and increased model size. This time, the increased training time is not critical, because the adversarial training would be performed on the backend (see Section 2) and we can assume that the backend has practically unlimited resources compared to the IoT devices. Regarding the model size, we indeed observe an increase of 10%, in case of Chunker, and 20%, in case of Disguiser, due to the 1.5 and 3 times, respectively, increase of the training set size. This translates to a few kilobytes of extra memory needed, which we believe to be still acceptable even on the resource constrained IoT devices.

5 RELATED WORK

Unlike traditional solutions, ML-based malware detection can be highly automated [20]. Furthermore, they use static and dynamic program analysis for extracting the required feature vectors [14]. Hence, their detection capabilities are better than that of traditional malware detection approaches. Feature vectors can be extracted from different sources, including the samples' instructions [18], their control-flow, invoked API functions and system calls [1], grey scale images of binaries, strings, and messages sent over network [11]. In addition, solutions that combine machine learning with cloud-based approach scale well and can be applied also in the IoT field [17]. They can use different ML models, including convolutional neural networks [15], recurrent neural networks, random forest classifiers [18], fuzzy and fast fuzzy pattern trees [7].

There are many different approaches for adversarial attacks also in the context of malware detection [4]. From these approaches we can highlight *append* and *slack* attacks [16] for their simplicity. Append attacks generate bytes and add them to the end of malware binary. Slack attacks add or modify bytes in slack regions of a binary, which are gaps between neighboring sections of an executable file. Our presented strategies (Chunker & Disguiser) resemble the previously mentioned append attack. There are other solutions for generating and appending bytes to the end of a binary, including gradient-based approach [9]. Another more advanced technique is program obfuscation, which can change the binary representation of a program while preserving its functionality. In order to do so, ML solutions can be used, including reinforcement learning-based approaches [2], Generative Adversarial Networks (GAN) and Recurrent Neural Networks (RNN) [8]. Obfuscating existing malware samples may be a successful strategy, but we do not use it, because from the perspective of 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-ML, and its detection performance on them has already been measured in [13]. Adversarial training is an effective way to increase the robustness of a ML-based systems against adversarial examples and it can be applied also in the malware detection domain. Indeed, there are several existing solutions that use this technique to improve their malware detection system [10], however, we applied first in the domain of ML-based IoT malware detection.

6 CONCLUSION

In this paper, we studied the robustness of SIMBIoTA-ML, an ML-based IoT malware detection method, against adversarial example attacks. We constructed two different strategies for creating adversarial examples from existing malware files: Chunker and Disguiser. Both strategies append bytes to the end of the malware, so they are relatively simple methods. Our measurement study shows that in case of Chunker, SIMBIoTA-ML has high detection rate, while in case of Disguiser, it has poor performance. To overcome this problem, we used the adversarial training concept to increase the robustness of SIMBIoTA-ML against the presented adversarial evasion techniques. For adversarial training, we extended the training sample set of SIMBIoTA-ML with adversarial examples constructed

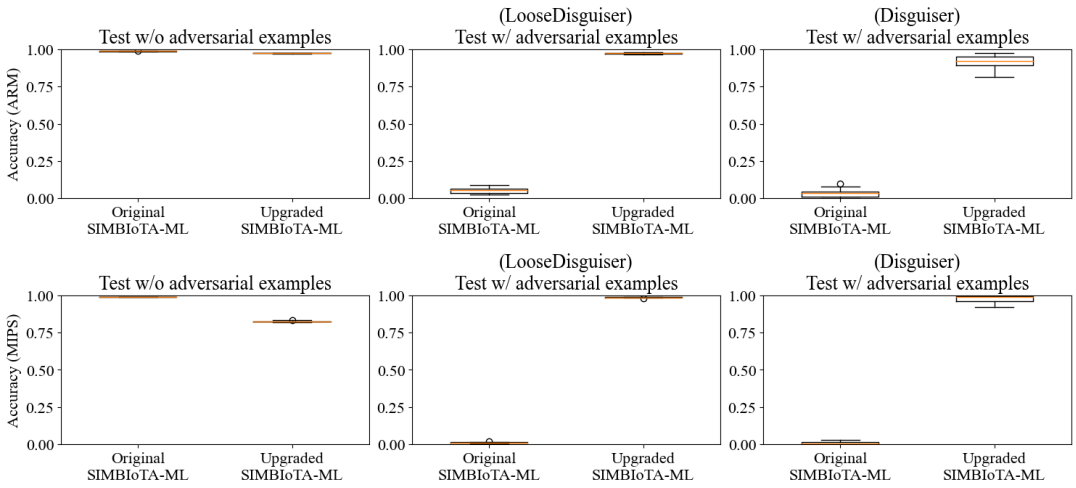

Figure 4: Comparison of the accuracy of the original and the upgraded SIMBIoTA-ML on the original test samples, and on the adversarial test samples constructed w/ LooseDisguiser strategy, and on the adversarial samples constructed w/ Disguiser strategy, in the ARM and MIPS cases.

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

ACKNOWLEDGMENTS

The research presented in this paper was supported by the European Union project RRF-2.3.1-21-2022-00004 within the framework of the Artificial Intelligence National Laboratory. The work was also part of 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.

REFERENCES

[1] Muhamed Fauzi Bin Abbas and Thambipillai Srikanthan. 2017. Low-Complexity Signature-Based Malware Detection for IoT Devices. In *Applications and Techniques in Information Security*, Lynn Batten, Dong Seong Kim, Xuyun Zhang, and Gang Li (Eds.). Springer Singapore, Singapore, 181–189.

[2] H. Anderson, Anant Kharkar, Bobby Filar, David Evans, and Phil Roth. 2018. Learning to Evade Static PE Machine Learning Malware Models via Reinforcement Learning. *ArXiv* abs/1801.08917 (2018).

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

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

[5] Marco Barreno, Blaine Nelson, Anthony D. Joseph, and J. D. Tygar. 2010. The security of machine learning. *Machine Learning* 81, 2 (01 Nov 2010), 121–148.

[6] Emanuele Cozzi, Pierre-Antoine Vervier, Matteo Dell'Amico, Yun Shen, Leyla Bilge, and Davide Balzarotti. 2020. The Tangled Genealogy of IoT Malware. In *Annual Computer Security Applications Conference* (Austin, USA) *(ACSAC '20)*. Association for Computing Machinery, New York, NY, USA, 1–16.

[7] Ensieh Modiri Dovom, Amin Azmoodeh, Ali Dehghantanha, David Ellis Newton, Reza M. Parizi, and Hadis Karimipour. 2019. Fuzzy pattern tree for edge malware detection and categorization in IoT. *Journal of Systems Architecture* 97 (2019), 1–7.

[8] Weiwei Hu and Ying Tan. 2017. Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN. *CoRR* abs/1702.05983 (2017). arXiv:1702.05983

[9] Bojan Kolosnjaji, Ambra Demontis, Battista Biggio, Davide Maiorca, Giorgio Giacinto, Claudia Eckert, and Fabio Roli. 2018. Adversarial Malware Binaries: Evading Deep Learning for Malware Detection in Executables. In *2018 26th European Signal Processing Conference (EUSIPCO)*. 533–537.

[10] Deqiang Li, Qianmu Li, Yanfang (Fanny) Ye, and Shouhuai Xu. 2021. Arms Race in Adversarial Malware Detection: A Survey. *ACM Comput. Surv.* 55, 1, Article 15 (nov 2021), 35 pages. https://doi.org/10.1145/3484491

[11] Yair Meidan, Michael Bohadana, Yael Mathov, Yisroel Mirsky, Asaf Shabtai, Dominik Breitenbacher, and Yuval Elovici. 2018. N-BaIoT—Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders. *IEEE Pervasive Computing* 17 (07 2018), 12–22.

[12] Jonathan Oliver, Chun Cheng, and Yanggui Chen. 2013. TLSH – A Locality Sensitive Hash. In *2013 Fourth Cybercrime and Trustworthy Computing Workshop*. 7–13.

[13] Dorottya Papp, Gergely Ács, Roland Nagy, and Levente Buttyán. 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,*. INSTICC, SciTePress, 55–66.

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

[15] Jiawei Su, Danilo Vargas Vasconcellos, Sanjiva Prasad, Daniele Sgandurra, Yaokai Feng, and Kouichi Sakurai. 2018. Lightweight Classification of IoT Malware Based on Image Recognition. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 02. 664–669.

[16] Octavian Suciu, Scott E. Coull, and Jeffrey Johns. 2019. Exploring Adversarial Examples in Malware Detection. *2019 IEEE Security and Privacy Workshops (SPW)* (2019), 8–14.

[17] Hao Sun, Xiaofeng Wang, Rajkumar Buyya, and Jinshu Su. 2017. CloudEyes: Cloud-Based Malware Detection with Reversible Sketch for Resource-Constrained Internet of Things IoT Devices. *Softw. Pract. Exper.* 47, 3 (mar 2017), 421–441.

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

[19] Csongor Tamás, Dorottya Papp, and Levente Buttyán. 2021. SIMBIoTA: Similarity-based Malware Detection on IoT Devices. In *IoTBDS*. SCITEPRESS, 58–69.

[20] Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. 2019. Survey of machine learning techniques for malware analysis. *Computers & Security* 81 (2019), 123–147.