Adversarial example generation architecture for a Random forest based malware detector

Tatiana Barbova

December 9, 2022

Supervisor: Dr. Levente Buttyan

Laboratory of Cryptography and System Security (CrySyS Lab)
Department of Networked Systems and Services (BME HIT)
Budapest University of Technology and Economics
Budapest, Hungary

Abstract

In the ICT dominated world, the threat of malware is constantly rising, and with the technology advancement the attacks become more sophisticated and harder to notice. As a result, automated malware detection has become an important direction of cybersecurity research, especially for IoT devices, having more vulnerabilities and performance constraints. For this purpose, machine Learning models are constructed to detect malware based on the feature set of a binary. However, the ML models are not perfect, and they are susceptible to adversarial attacks. In this study we investigate the generality of an AE generation framework that allows us to explore erroneous model states and fool the detector, specifically for IoT malware. Our results serve as a great baseline and overview of challenges for further research in this field.

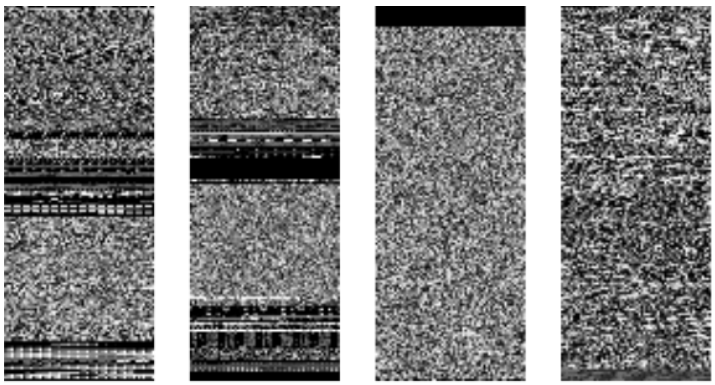
1 Introduction

The threat of malware is spreading rapidly with the increasing integration of ICT devices in the everyday life and software and hardware diversification. Especially this threat concerns the IoT devices with their high interconnection and low security requirements due to computational limits. Nowadays IoT devices get more and more integration in critical applications, for example in healthcare for remote patient monitoring, so malware detection for IoT devices specifically is of a particular importance. One of the most famous IoT malware attacks, Mirai[1], infected over 600,000 IoT devices coordinated by no less than 484 unique Command-and-Control servers resulting in Denial of Service (DoS).

Currently various techniques for malware detection exist with different performance bottlenecks. In recent years with evolution of Machine Learning, and especially Deep Learning methods, these approaches found its application in malware detection as well[2]. Compared to the traditional detection methods, Deep Learning is simpler in implementation due to automatic feature recognition, but vulnerable to Adversarial Examples [4](AEs): samples which were slightly perturbed in order for the ML model to produce desirable output (misclassification in this particular case). Adversarial attacks are an active research area currently, e.g. AE generation was studied for visualization-based malware detection methods[3]. Using AEs during the training process of the model can increase its robustness towards such attacks.

It is important to emphasize why existing studies on adversarial attacks for general malware cannot be generalized for IoT malware. IoT devices have a different ISA and different structure for the binary files consequently, so a study specifically for IoT malware should be conducted separately. In figure 1, the binaries for Windows and IoT devices are transformed to grayscale images and the structure difference is obvious even to a human eye.

On the other hand, coverage-guided fuzzing [5] is an effective way to find bugs in software. More model states discovered signals that the model testing is more objective and extensive. Test suits covering more model states are better able to find model errors. In this study, these 2 main ideas will be implemented together to generate AE for IoT malware detectors, the short summary of relevant studies will be provided below to support the motivation.



(a) Win. ben. (b) Win. mal. (c) IoT ben. (d) IoT mal.

Figure 1: Sample visualization of benign and malicious binaries. Here, 1(a) and 1(b) show benign and malicious Windows binaries, respectively. While, benign and malicious IoT binaries are shown in 1(c) and 1(d), respectively[3]

2 Reference papers

2.1 MalFuzz: Coverage-guided fuzzing on deep learning-based malware classification model[6]

Authors affirm that there is no related work to test deep learning-based malware detection models specifically. Therefore, they propose MalFuzz. It uses the idea of coverage-guided fuzzing to test deep learning-based malware detection models. The framework works in a way that based on the seed selection algorithm a sample is mutated. The sample is fed to the model and the coverage information is calculated. If this is a new coverage, then this seed is considered to be with high mutation potential and it is added back to the seed space. Additionally, if it produced a misclassification, it is a valid test case. For the model state representation, MalFuzz uses the first and last layer neuron values as an approximation. To calculate the new coverage, MalFuzz uses the fast approximate nearest neighbor algorithm. Besides that, the seed selection strategy is based on the size of a sample and its mutation potential. Based on MalConv, Convnet, and CNN 2d detection models, they compared the modified Tensor-Fuzz and MAB-malware with MalFuzz, and in all 3 cases MalFuzz was generating more classification errors in the same timeframe. Regarding the executability of the samples, the mutation operation of MalFuzz can retain the original functionality of malware with probability of 92.9%.

2.2 SIMBIoTA-ML: Light-weight, Machine Learning-based Malware Detection for Embedded IoT Devices[7]

This paper proposes a light-weight antivirus solution that enables embedded IoT devices to take advantage of machine learning-based malware detection. The model used is a Random Forest classifier. For the feature generation TLSH is used. TLSH[8] is a similarity hash algorithm, but it is different from cryptographic hashes, since its goal is to maximize collisions. It implies that "unimportant" changes in the input do not alter the TLSH output significantly. This model has a true positive malware detection rate of around 95%, while having a low false positive detection rate at the same time. Besides, the detection process of SIMBIoTA-ML has a near-constant running time, which allows IoT developers to

have better approximations of the delay introduced by scanning a file for malware, which is crucial for time-critical applications.

3 Goals of the research

This study proposes to examine the generality of the MalFuzz AE generation framework to other detection models, as the deep neural networks are already known to be susceptible to this kind of attacks, but these are not the only models used for malware detection. The model chosen for this purpose is Random Forest classifier as in SIMBIoTA-ML by Papp et al. Due to its simplicity and hash-based feature selection approach. The specific objectives are as follows:

1. High misclassification rate
2. Preserving executability
3. Approaches and possible issues for future generalization

4 The architectural overview

The adversarial example generation framework is based on the one of MalFuzz[6], and it is illustrated on Figure 2. In the beginning all the binaries (both malware and benign) are in the normal seed pool and the priority seed pool is empty. This will store mutated seeds that discovered a new model state, and thus they have high mutation potential and the probability to be misclassified. The seed selection strategy: if the priority seed is not empty, a sample is chosen from there, otherwise from the normal pool randomly. Then, a mutation operation is applied to the seed and this is later fed to the detection model. If the sample is misclassified, it is recorded as a valid test case. Regardless of the model output, for every seed the coverage information is extracted, and the distance calculation algorithm is used to compare this model state to the previously recorded model states, and if the algorithm decides that a new state was discovered, the corresponding mutated seed is added to the priority pool and the new coverage is recorded.

4.1 Seed manager

Compared to MalFuzz, the working framework was simplified with the goal of advancing it gradually based on the performance. More precisely, the sample candidates for AEs are chosen independently from the original file size, rather than having seed pools based on the size of the malware and a probability function assigned to that. Though if a perturbed sample did discover a new model state, it is added to the new seed pool considered having a high mutation potential, same as in the MalFuzz framework. These seeds will have a priority for mutation over the other seeds. From the mutation operations only Overlay Append (OA) was implemented – appending a class' contents to the end of a binary of an opposite class. This mutation operation is chosen since it is easy to implement and also it preserves the executability of the sample with a 100% probability (appending bytes to the end of a binary is equivalent with appending to an unreachable area). The number of bytes appended is proportional to the size of the original binary file – 0.1%, and is taken from the end of a file.

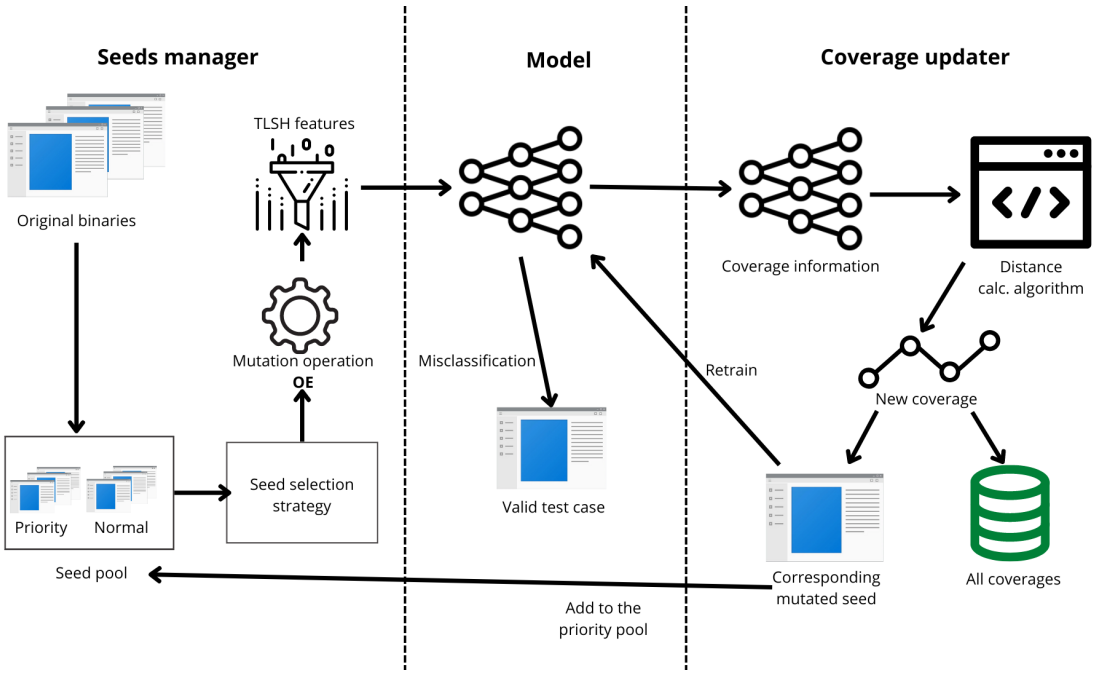


Figure 2: The data flow in the developed architecture

4.2 Model

In this study the used detection model is the Random Forest classifier, having 10 estimators with maximum tree depth of 6. This architecture proved to be efficient for malware detection in the study of Papp et al. This model is of a particular interest for this research, since information gain based classifiers are believed to be less susceptible to adversarial attacks, compared to the deep neural networks for example, and this is to be verified. On the other hand, the feature selection process is more difficult, since they must be chosen carefully for a decision tree, whereas in case of neural networks the feature extraction is done automatically.

4.3 Feature selection

To keep the consistency with the above mentioned paper, the same method of feature selection was performed. First the TLSH hashing is applied to the binary, and then this value is transformed into features by splitting. From the header: L value, Q1 ratio and Q2 ratio (3 features). Then the bytes that represented the buckets are split into bit pairs (128 more features). In this study checksum was included in the feature set as well since this is one of the fields easiest to manipulate without changing the semantics of the other features. Moreover, using hashing as a feature selection method is of an additional value for an adversarial attack study, since hashing is trying to compress data to its most essential part, however the adversarial methods usually alter non-significant parts of a sample, to make the perturbation less noticeable. The main question is whether the hash will keep the information delivered by the mutated bytes.

4.4 Coverage updater

MalFuzz uses the values from the first and the last layers of the detector neural network as coverage information. As having a different type of model now, a new method is needed to decide whether

a sample discovered a new model state. This is one of the general challenges of this architecture: sometimes coverage information of a model is not obvious or cannot be extracted directly and some approximations must be done. In this case however it is easy: after passing a sample to the random forest classifier it is possible to get the decision path – a compressed sparse row matrix representing whether a particular tree node was gone through with a 1 and 0 otherwise, together with an array storing the indices for every estimator. This is quite a precise way to represent a state of the model, and, after unfolding the sparse matrix, different methods were used to calculate the distance between the coverages: the euclidean distance, the manhattan distance and the sum-difference (as the distance vectors are binary, we can subtract them and sum up the absolute values). Regarding the distance calculation algorithm, MalFuzz paper did not describe the way how a distance thresholds were chosen, so in this research it is done empirically as well for all of the 3 distance metrics, though it is easier to give semantics to this threshold. For example, if to calculate the distance the sum-difference method is used, then the meaning of the threshold would be the number of different nodes, and it can even be set dynamically, for example 5% of the whole number of nodes. More on that will be elaborated in the Experiments section.

Another key difference from the MalFuzz framework is that the model is not static. Whenever a new model state is discovered, the model is retrained with the corresponding sample. This is done in order to check the hypothesis whether the model becomes more robust if the AEs are used in the training process.

5 Experiments

5.1 Technical data and setup

The technical data of the machine: Ubuntu 22.04, processor AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx. Python 3.10.6 was used for coding.

The dataset contains 4000 IoT binaries – 2000 benign and 2000 malware, all of ARM architecture, provided by the CrySyS lab. For the purpose of the adversarial attacks the MIPS binaries were used too but not during the training, since it was intended to study the behavior of the model when it encounters binary sequences unseen previously as well.

Every test was running for 500 epochs, for each of every distance calculation algorithms, and in every epoch a single AE was generated and fed to the model. The first stage only included malware samples. In the second stage of the experiment both benign and malware were mutated, this was done with the goal to see the comparative performance of the model. There was an equal probability for both benign and malware binaries to be chosen for a mutation operation. The results of the experiments are summarized in the tables 2, 3 and 4.

5.2 Threshold considerations

It can be observed that for all the mutation operations, the lower the threshold, the higher malware misclassification rate. It is true for both cases when the mutation operations were applied to malware only, and to both malware and benign samples as well. This can be explained in the following way: a lower threshold means that more results would be considered as new state coverages, and, even if with smaller increments, overall the model state exploration is done quicker and the AE generation is more successful.

	Euclidean		
	>3	>6	>9
Malware only	99.20%	84.40%	73.00%
Benign & Malware	6.44%	24.56%	28.17%
	100.00%	82.09%	71.37%

	Manhattan		
	>12	>35	>60
Malware only	86.40%	84.40%	73.00%
Benign & Malware	5.39%	22.81%	27.67%
	100.00%	82.27%	72.06%

	Sum-difference		
	>0.02%	>0.05%	>0.015%
Malware only	84.00%	76.00%	73.00%
Benign & Malware	19.91%	28.07%	28.17%
	96.21%	90.84%	71.37%

Table 1: Misclassification rates for the different distance calculation algorithms and thresholds

At the same time, with the increasing threshold, the benign misclassifcation rate increases, which is the opposite trend. A possible explanation for this could be that the chosen mutation operation – OA, is more efficient for malware samples, in a way that the random forest classifier uses the last bucket pairs (corresponding to the end of the file) at the deeper level of the trees, whereas mostly benign files are classified long before reaching those tree levels. To verify this hypothesis it would be necessary to compare average or most frequent decision paths of a benign and of a malware sample, which is left for the future of the research.

Another important observation is the sensitivity of the results to the threshold values. It makes it hard to compare the algorithms to each other since if a larger threshold was chosen for any of them, the performance would worse. This emphasizes the importance of introducing a distance metric that actually has semantics and for which threshold value would not be purely empirical, for example the length of the longest common prefix. However, one observation can be made surely: for the sum-difference algorithm, an overall higher benign misclassification rate is associated, which makes it underperforming, despite having a good semantics of threshold value.

It is worth mentioning that with a lower threshold the model is retrained with a larger number of the corresponding AEs, but the misclassification rate still increases. Further experiments were running for 1500 epochs, and only malware binaries were exposed to mutation.

5.3 Detector performance considerations

Initially the model was retrained every time a new coverage was discovered with the corresponding sample. In the figure 3 a chart is presented, it contains an average of misclassificated samples per every 10 generated AEs for different detector retraining options. The obvious trend is that with time the model did not improve the performance, which means that the AE generator is indeed good at fooling the detector. The most stable strategy is to retrain the model every time after a new coverage is discovered. In the region where every curve peaks, probably an overfitting happens.

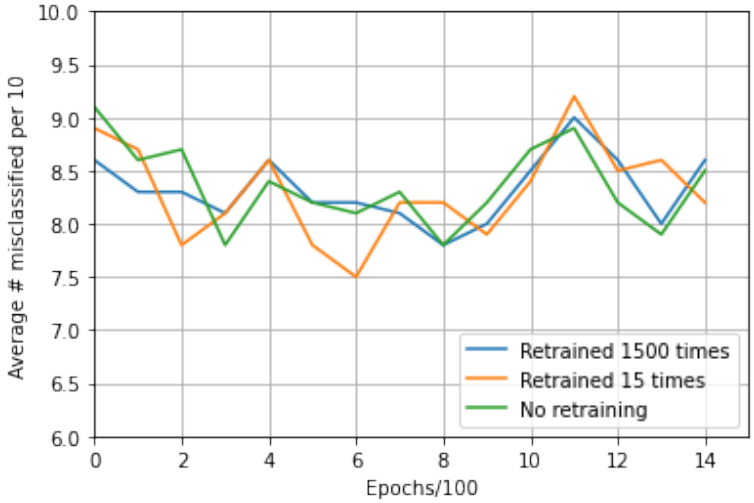


Figure 3: Average number of misclassified samples per 10 samples

5.4 Mutation operation considerations

Decreasing the size of the appended byte sequence from 0.1% of the file length to 0.02% did not deteriorate the performance (approximately 83.3% misclassification rate). However, when removing the MIPS byte sequences from the available mutation pool, the misclassification rate significantly dropped (from 83.3% with MIPS to 60.2% without MIPS). Having the results of the previous section, we can conclude that the model hardly learns on significantly different byte sequences unless they were initially present in the dataset.

5.5 Running time

As this study is a novel approach in IoT malware detection, a comparison with other coverage-based fuzzers for this problem cannot be done yet. The closest architecture, MalFuzz, is generating AEs for neural networks and it was tested for Convnet, Malconv and CNN 2-d detectors. On average, MalFuzz generates 694 AEs in 20 minutes, i.e. 34.7 samples/min. The proposed method, depending on the threshold value and the mutation operation considerations above, can generate 1250 AEs during a 1500 epoch run in approximately 16 minutes, u.e 78.125 samples/min.

6 Directions for further research

1. All the other mutation operations described in MalFuzz[6] are to be added to the framework and compared to the original performance – Section Append, Section Add, Remove Debug, Remove Certificate and Break Checksum. Overlay Append only provides a limited opportunity to manipulate an original sample and in a similar pattern, which is far from the real world.

2. Distance calculation algorithms with better semantics and a threshold estimation methodology should be introduced, based on the tree structure rather than lists of nodes

3. Other detection models based on this feature set would be constructed, for example a logistic regression, so that a comparative performance analysis can be done.

7 Conclusion

The described architecture proved to be efficient in AE generation for decision-tree based malware detection models. The misclassification rate can reach 95% regardless of the distance calculation algorithm, and the executability of all the samples is preserved as well. This leads to the idea that possibly there is no malware detector perfectly safe from the adversarial attacks. The best thing that can be done is using an extensive training dataset (in terms of size, ISA and malware families) as the misclassification rate can indeed be reduced this way. The architecture is easy to construct for the interested to conduct their own experiments. As an attempt to generalize an already existing AE generating framework, this work discovered the main challenges in this process: for some models extracting coverage information is not obvious; calculating the distance between coverages and choosing the threshold is mostly empirical; and so far few other studies exist in this area for an extensive comparative analysis. However, this work is just a baseline, and it has many directions to improve mentioned in the previous section.

References

[1] 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.

[2] Aslan Ö. A. and Samet R. (2020) *A comprehensive review on malware detection approaches* IEEE Access, vol. 8, pp. 6249–6271

[3] Aminollah Khormali, Ahmed Abusnaina, Songqing Chen, DaeHun Nyang, and Aziz Mohaisen (2019) *COPYCAT: Practical Adversarial Attacks on Visualization-Based Malware Detection*

[4] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, *Deepfool: A simple and accuratemethod to fool deep neural networks," in Proceedings of the 2016 IEEE Conferenceon Computer Vision andPattern Recognition* CVPR, 2016, pp. 2574–2582.

[5] Li J., Zhao B., and Zhang C. (2018) *Fuzzing: a survey* Cybersecurity, vol. 1, no. 1, p. 6

[6] Yuying Liu, Pin Yang, Peng Jia ID, Ziheng He, Hairu Luo (2022) *MalFuzz: Coverage-guided fuzzing on deep learning-based malware classification model*

[7] Dorottya Papp, Gergely Acs, Roland Nagy, Levente Buttyan (2022) *SIMBIoTA-ML: Light-weight, Machine Learning-based Malware Detection for Embedded IoT Devices*

[8] 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.