Hardware-Assisted Malware Detection using Explainable Machine Learning

Zhixin Pan, Jennifer Sheldon and Prabhat Mishra
Department of Computer & Information Science & Engineering
University of Florida, Gainesville, Florida, USA

***Abstract*—Malicious software, popularly known as malware, is widely acknowledged as a serious threat to modern computing systems. Software-based solutions, such as anti-virus software, are not effective since they rely on matching patterns that can be easily fooled by carefully crafted malware with obfuscation or other deviation capabilities. While recent malware detection methods provide promising results through effective utilization of hardware features, the detection results cannot be interpreted in a meaningful way. In this paper, we propose a hardware-assisted malware detection framework using explainable machine learning. This paper makes three important contributions. First, we theoretically establish that our proposed method can provide interpretable explanation of classification results to address the challenge of transparency. Next, we show that the explainable outcome can lead to accurate localization of malicious behaviors. Finally, experimental evaluation using a wide variety of real-world malware benchmarks demonstrates that our framework can produce accurate and human-understandable malware detection results with provable guarantees.**

***Index Terms*—Malware Detection, Explainable Learning**

I. Introduction

Malicious software (malware) is any software designed to harm a computer, server, or computer network and cause severe damage to the target system. Malware detection is a "cat and mouse" game where researchers design novel methods for malware detection, and attackers develop devious ways to circumvent detection. Signature-based detectors compare the signature of a program executable with previously stored malware signatures. However, signature-based anti-virus software (AVS) is not effective even for known malware with polymorphic or metamorphic features. Also, they are computation intensive, and as a result, they are not suitable for resource-constrained systems such as IoT edge devices that operate under real-time, power and energy constraints.

Recent research efforts explored designing hardware-assisted malware detection with the hardware as a root of trust. The underlying assumption is that although AVS can be fooled by variations in malware code, it is difficult to subvert a hardware-based detector since the malware functionality will remain the same. There are some promising directions for hardware-assisted malware detection using embedded trace buffer (ETB) and hardware performance counters (HPC). ETB based malware detection [1] shows advantages over HPC based methods [2] in terms of classification accuracy. Despite all these advantages, exploiting hardware components for malware detection is still in its infancy - there is no strong theoretical basis. Machine learning has been successfully used for malware detection [3]. However, none of the previous works on machine learning based malware detection are explainable. Therefore, the detection results cannot be interpreted in a meaningful way. A major objective of our proposed approach is to provide transparency in malware detection.

In this paper, we propose a hardware-assisted malware detection that takes advantage of explainable machine learning. This paper makes the following major contributions:

1) To the best of our knowledge, our approach is the first attempt in developing hardware-assisted malware detection using explainable machine learning, which leads to interpretable detection results.
2) The interpretation sheds light on why the classifier makes incorrect decisions, which leads to malware localization.
3) Our experimental evaluation demonstrates the effectiveness of our proposed method in accurate classification of benign and malicious programs, as well as interpretation of detection results.

The rest of this paper is organized as follows. We survey related efforts in Section II. Section III describes our proposed method on malware detection. Section IV presents experimental results. Finally, Section V concludes the paper.

II. Related Work

In the early days, the focus of detection was on static analysis [4] by utilizing software filters for malware detection. Unfortunately, this naive approach can be circumvented by obfuscation [5]. Dynamic detection techniques try to defend against obfuscation [6] by keeping track of the runtime behavior of software and reporting any malicious behavior such as illegal access. However, both static and dynamic detection methods run on the software level, which is unable to detect malware with deviation capabilities. Recent research efforts [1] turned interest into hardware-based detection approaches due to their robust resistance. While existing approaches are promising, they inherits two fundamental limitations: (i) expensive preprocessing to eliminate useless benign cycles, and (ii) user gets only the final decision without understanding how the decision was made or where to locate the infected area.

The demand for explainable machine learning has been steadily increasing ever since machine learning algorithms were adopted in many fields, especially in security domains. Explanation schemes in machine learning aim at tackling

This work was partially supported by the NSF grant CCF-1908131.

2576-6996/20/$31.00 ©2020 IEEE
DOI 10.1109/ICCD50377.2020.00113

this issue by reasonably demonstrating the reason for making predictions [7]. This task is performed by sorting the input features ordered by their contribution towards the final decision. By analyzing values of top features with highest weights, an human-understandable illustration can be obtained. However, existing approaches can not be naturally extended to malware detection since almost all of them focus on static pixel images while we need to handle input data that are time-sequential records. *To the best of our knowledge, our proposed approach is the first attempt in applying explainable machine learning for hardware-assisted malware detection.*

III. Malware Detection using Trace Analysis

Our proposed approach enables a synergistic integration of hardware trace analysis and explainable machine learning for efficient malware detection. We utilize existing design-for-debug architecture, such as embedded trace buffer, for trace collection. Such traces can be viewed as a $w \times d$ table $\mathbf{X}$, where w is the *width*, d is the *depth*. It represents the recorded values of w traced signals over d clock cycles. We split each column as a single feature component, i.e values of all traced signals within one single cycle. Next, we apply explainable machine learning for malware detection.

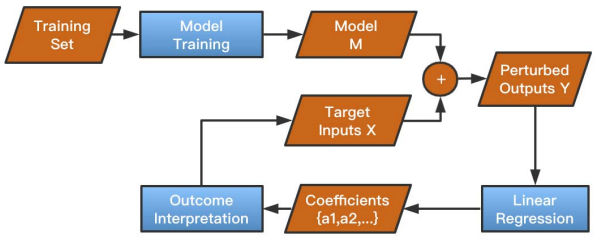


Figure 1: Proposed malware detection framework consisting of three major activities: training, regression, and interpretation.

Figure 1 shows an overview of our proposed method that consists of three major tasks. The first task is *model training*, where we train a machine learning classifier M using collected traces. The second task performs *linear regression*. For an input instance $\mathbf{x}$ we want to explain, perturb $\mathbf{x}$ randomly to generate artificial input dataset $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ...\}$ and feed them to M to obtain corresponding model output $\mathbf{Y} = \{M(\mathbf{x}_1), M(\mathbf{x}_2), ...\}$, based on which we further compute a linear regression function. The goal of the last task is to perform *outcome interpretation*. Specifically, the top features ranked by the magnitude of coefficients will provide users the crucial timing information of malware. The remainder of this section describes these three tasks in detail.

A. Model Training

As discussed in Section II, hardware-assisted malware detection techniques should monitor the behavior of software at run-time. Therefore, relying on single-cycle data is not effective since malicious behavior usually consumes several sequential cycles. Moreover, single-cycle based strategies are likely to mispredict a benign software as malicious. This is due to the fact that malware also contains normal operations, and considering these benign operations as important features of malware can lead to misclassification. A well-designed preprocessing strategy can mitigate this by filtering overlapped common behaviors shared by both malicious and benign programs. However, the difficulty of designing such a strategy is very high, and there is no guarantee that it can be performed in all cases. Therefore, an ideal machine learning model for our task should satisfy the following two properties:

(1) Ability to accept time series type data as input.

(2) Ability to make decisions utilizing potential information concealed in consecutive adjacent inputs.

Therefore, we propose to utilize *Recurrent Neural Network (RNN)* to tackle this problem. As shown in Figure 2 , instead of finishing the input-output mapping in one forward pass, the RNN accepts sequential inputs. For each single input x_i, RNN not only provides immediate response h_i, but also stores the information of the current input by updating the architecture itself. And information corresponding to the previous step will also be fed into the architecture to supply extra information.

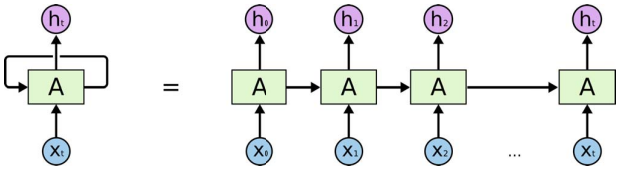


Figure 2: Recurrent Neural Network

B. Regression

Once we have the well-trained model, we can start to perturb the target input to generate corresponding perturbed output dataset. Applying linear regression will lead to a linear prediction model to fit our artificial dataset. Note that a linear prediction model can always be expressed by a polynomial, then we can utilize the expression to extract weight information. Formally, given a data set $\{\mathbf{y}, \mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$, where n is the number of samples, linear regression takes the following form by appending error variable ϵ:

$$\mathbf{y} = \sum_{i=1}^{n} a_i \mathbf{x}_i + \epsilon$$

where a_is are model parameters, and the goal is to minimize ϵ as much as possible. In our case, the input is the $w * d$ trace table $\mathbf{X}$ as mentioned before. Since we treat each column of this table as an individual input feature, we have $\mathbf{X} = [\mathbf{x}_1\ \mathbf{x}_2\ ...\ \mathbf{x}_d]$, where each $\mathbf{x}_i$ is a vector in the size of $w*1$. We choose $\mathbf{y}$ as the output of last hidden state, this leads to an optimization problem:

$$\arg\min_{\mathbf{a}} ||\mathbf{X}\mathbf{a} - \mathbf{y}||_2$$

where $\mathbf{a} \in \mathbb{R}^d$ is $[a_1\ a_2\ ...\ a_d]^T$, i.e, coefficients to be solved. This is a common convex optimization problem and its solution can be obtained by $\mathbf{a} = (\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t\mathbf{y}$. Unfortunately, this method cannot be directly applied to solve our task. First, this theoretical solution exists only when $\mathbf{X}^t\mathbf{X}$ is invertible (full rank), which may not be true most of the time. Second, even when $\mathbf{X}^t\mathbf{X}$ is full rank, linear regression assumes input vectors are *independent*, otherwise it will produce unreliable results when any two of $\mathbf{x}_i$ (columns) are highly correlated. Specifically, assume that the regression function is computed

to be $\hat{\mathbf{y}} = a\mathbf{x}_1 + b\mathbf{x}_2 + c\mathbf{x}_3 + d$, where $\mathbf{x}_1$ and $\mathbf{x}_2$ are highly related features and they are very close to each other. Then there is a canceling effect between a and b. Increasing a by certain amount while decreasing b by the same amount at the same time will not lead to drastic change in $\hat{\mathbf{y}}$. The problem becomes ill-posed since absolute value of a and b can vary significantly under different computing procedure or initial conditions. Then the comparison between $|a|$, $|b|$ and $|c|$ is not useful, therefore, the interpretability of the model is greatly reduced. Since adjacent columns in trace table are sequential records of signal values within a short duration, violation of this independence assumption is likely to happen.

In our study, we applied *ridge regression*, which is an improved least squares estimation method, and the fitness of correlated data is stronger than general regression. Ridge regression is achieved by appending one extra penalty term to the optimization problem:

$$\arg\min_{\mathbf{a}} ||\mathbf{Xa} - \mathbf{y}||_2 + \lambda||\mathbf{a}||_2$$

Intuitively, a size constraint is imposed to restrict the absolute value of all coefficients, which alleviate the problem of high variance of coefficients. Moreover, notice

$$\arg\min_{\mathbf{a}} ||\mathbf{Xa} - \mathbf{y}||_2 + \lambda||\mathbf{a}||_2 \rightarrow \arg\min_{\mathbf{a}} ||(\mathbf{X} - \lambda\mathbf{I})\mathbf{a} - \mathbf{y}||_2$$

Replacing $\mathbf{X}$ with $\mathbf{X} - \lambda\mathbf{I}$ is a general way to avoid the problem for $\mathbf{X}$ being singular matrix. Also, data was centralized and the problem of high variance is alleviated. Therefore, with ridge regression, coefficients obtained is more reliable and fit better for our dataset, which has high correlation.

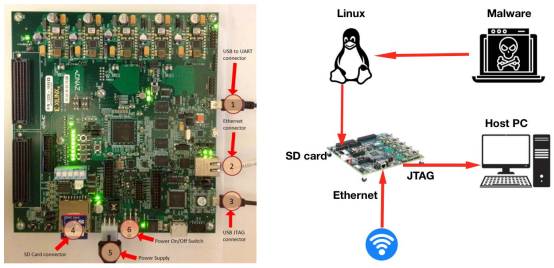
C. Outcome Interpretation

Once coefficients of regression are obtained, we can derive the importance ranking, then interpret it into meaningful information in the context of malware detection. The top features come with large coefficients that are likely to be related to the malicious behavior. Next, we can check the clock cycle distribution of these top features. It is expected to provide us with extra information about the malware. For example, if we observe adjacent cluster of top features, then the time slot within which they reside shall provide a general indication of time information about when malicious behavior happened. Similarly, if clock cycle numbers are periodically separated, the detected malware is likely to repeat its malicious activity periodically. Typical malware acting like this usually works in a client-server mode, where client program steals private data and sends message to the hacker's server in a periodic interval. For a closer look, we can split the table into rows and go through the same process. This will lead to identification of trace signal values that are most likely relevant to the malicious behavior, which in turn will lead to malware localization.

IV. Experimental Evaluation

A. Experimental Platform

We ran malicious and benign programs on the Xilinx Zynq-7000 SoC ZC702 evaluation board as shown in Figure 3. We consider a wide variety of malware families [8] including the following three popular ones: *BASHLITE Botnet*, *PNScan Trojan* and *Mirai Botnet*. Our benign programs include system



(a) ZYNQ SoC Board (b) Platform Layout

Figure 3: Experimental Platform

binaries such as $ls, mkdir, ping, netstat$[1]. The traced values gathered by running both malware and benign programs on the hardware board are utilized as inputs to our classifier.

B. Evaluation: Accuracy

We compare the accuracy of our approach with the state-of-the-art hardware-assisted malware detector, PREEMPT [1]. PREEMPT utilizes two types of implementation, random forest (PREEMPT_RF) and decision tree (PREEMPT_DT). We run both malicious and benign software on our hardware platform. We executed a total of 367 programs (including both malicious and benign ones) and all the traced data were mixed up and further split into training (80%) and test (20%) sets after labeling. Total training epochs are 200 for every model and we plot test accuracy every 10 epochs. The performance of all methods are depicted in Figure 4.

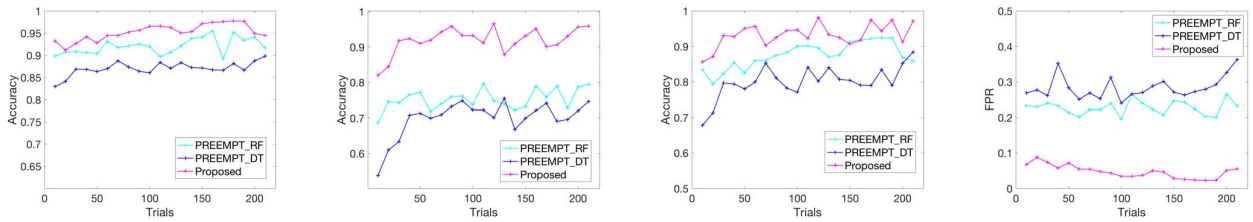
Figure 4 compares the prediction accuracy of our approach with PREEMPT_RF and PREEMPT_DT. As we can see, our proposed method provides the best malware detection accuracy. The PREEMPT appeared fragile in the face of PNScan, with an average of 62.7% accuracy for DT and 76.9% for RF, while the proposed method provided an average accuracy as high as 91.4%. For BASHLITE, both proposed method and RF performed well and the best accuracy of our method is 98.9%. For Mirai, our proposed method achieved 97.5% accuracy while PREEMPT attained a maximum accuracy of 92.5% with RF. Note the inferior performance of PREEMPT_DT in Mirai.

If we omit malware and test models on traced data gathered from benign software only, Figure 4(d) shows the false positive rate (FPR) of all three methods. The diagram illustrates the major drawback of PREEMPT, it possesses an average FPR as high as 25.9% with RF, and 31.6% with DT. In other words, it is very likely to mispredict a benign software as malware. Tested benign software samples also execute Linux system binaries like *netstat* and *ping*, which are also frequently executed by botnet malware. Since the PREEMPT cannot analyze time sequential data, it failed to recognize benign execution of these binaries with the help of context and produced wrong predictions. In contrast, our framework obtained FPR as low as 3.4%.

C. Evaluation: Outcome Interpretation

We also evaluated the performance of our method by interpreting the contribution factor for the classification results.

[1]ping and netstat are important since our malware are botnets.



(a) Performance on BASHLITE (b) Performance on PNScan (c) Performance on Mirai (d) False Positive Rate (FPR)

Figure 4: Performance of machine learning models: (a) - (c) for various malware, and (d) for benign benchmarks

For clock-cycle related analysis, an example of executing BASHLITE's client on host machine is shown in Figure 5.

	C_1	C_2	C_3	C_4	C_5	C_6	C_7	...
R_1	8ca0	bc00	1886	8ca0	a68c	c401	e401	
R_2	a2c2	08b4	a6ab	e6a2	0004	88fd	b422	
R_3	b485	ec00	28d4	2101	506c	02f0	02e2	
R_4	8c21	6002	d201	90c4	02f9	90a2	0048	
...								
	0.003	0.028	0.073	0.431	0.136	0.288	0.001	

Figure 5: Interpretation of BASHLITE client's traced signals

Figure 5 shows a snapshot of the trace table, where each row represents the values in a register in specific clock cycles (each column represents a specific clock cycle). Our proposed method computed the corresponding contribution factor of each clock cycle towards the RNN output using linear regression, which is shown as weights in the last (colored) row. As we can see, the weight of C_4 is significantly larger than the others. This immediately indicates the clock cycle of malicious behavior. By tracing the execution, we find that C_4 points to the timestamp before the start of function "*processCmd*" in BASHLITE, which is the most important function of BASHLITE to perform its malicious functionality. In other words, this is the starting point and exact reason for recognizing this program as malware.

Another example of outcome interpretation is shown in Figure 6, where we measure the contribution of each traced register signal. The given data is the trace table of executing Mirai's bot on host machine. This time we evaluate the contribution row-by-row, and the result is listed on the right side of the trace table.

	C_1	C_2	C_3	C_4	C_5	...	
R_1	c436	6002	21a4	d401	fcb4		0.096
R_2	2244	00cf	a6ab	8ad5	008a		0.068
R_3	0001	0004	00a1	0000	001b		0.631
R_4	00de	b8f1	b0a1	800e	7402		0.001
R_5	028a	a800	0028	0042	06c0		0.165
							...

Figüre 6: Interpretation of Mirai bot's traced signals

As we can see, register R_3 is recognized as the most important factor. Here R_3 is storing the variable "*ATTACK_VECTOR*" in Mirai. This variable records the identity of attack modes, based on which the bot takes relative actions to perform either a UDP attack or DNS attack. This attack-mode flag is the most important feature of a majority of malware bot programs, and our proposed method successfully extracted it from the traces to illustrate the reason for making this prediction.

V. Conclusion

Design of trustworthy systems needs to eliminate malicious software (malware) as well as malicious hardware. Detection of hardware Trojans utilize an effective combination of simulation-based validation [9], [10] and side-channel analysis [11], [12]. State-of-the-art malware detection methods have several limitations including limited prediction accuracy and lack of transparency. Our proposed approach addresses these limitations by developing a regression-based explainable machine learning algorithm. Our approach is able to find the major contributors among all input features by perturbation and linear regression. Experimental results demonstrated that our approach significantly outperforms (with average accuracy of 98.9%) state-of-the-art approaches on three most popular malware families. Unlike existing approaches, our approach provided transparency in prediction results, which is vital for outcome interpretation as well as malware localization.

References

[1] Kanad Basu et al. PREEMPT: preempting malware by examining embedded processor traces. In *DAC*, page 166, 2019.
[2] Xueyang Wang et al. Hardware performance counter-based malware identification and detection with adaptive compressive sensing. *ACM TACO*, 13(1):3, 2016.
[3] Joshua Saxe and Konstantin Berlin. Deep neural network based malware detection using two dimensional binary program features. In *10th MALCON*, pages 11–20, 2015.
[4] A. Moser, C. Kruegel, and E. Kirda. Limits of static analysis for malware detection. pages 421–430, Dec 2007.
[5] *Malware Obfuscation Techniques: A Brief Survey*. IEEE Computer Society, 2010.
[6] Jacob et al. Behavioral detection of malware: from a survey towards an established taxonomy. *JCV*, 4(3):251–266, 2008.
[7] Marco Ribeiro et al. "why should I trust you?": Explaining the predictions of any classifier. In *SIGKDD*, 2016.
[8] Kishore Angrishi. Turning internet of things(iot) into internet of vulnerabilities (iov). *CoRR*, 2017.
[9] Yangdi Lyu and Prabhat Mishra. Automated trigger activation by repeated maximal clique sampling. In *ASP-DAC*, 2020.
[10] A.Ahmed et al. Scalable trojan activation by interleaving concrete simulation and symbolic execution. In *ITC*, 2018.
[11] Zhixin Pan et al. Test generation using reinforcement learning for delay-based side-channel analysis. In *ICCAD*, 2020.
[12] Huang et al. Scalable test generation for trojan detection using side channel analysis. *TIFS*, 13(11):2746–2760, 2018.