This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2021.3068873, IEEE Transactions on Computers

Efficient Hardware Malware Detectors that are Resilient to Adversarial Evasion

Md Shohidul Islam, *Student Member, IEEE,* Khaled N. Khasawneh, *Member, IEEE,* Nael Abu-Ghazaleh, *Senior Member, IEEE,* Dmitry Ponomarev, *Senior Member, IEEE,* and Lei Yu, *Senior Member, IEEE*

Abstract—Hardware Malware Detectors (HMDs) have recently been proposed to make systems more malware-resistant. HMDs use hardware features to detect malware as a computational anomaly. Several aspects of the detector construction have been explored, leading to detectors with high accuracy. In this paper, we explore whether malware developers can modify malware to avoid HMDs detection. We show that existing HMDs can be effectively reverse-engineered and subsequently evaded. Next, we explore whether retraining using evasive malware would help and show that retraining is limited. To address these limitations, we propose a new type of Resilient HMDs (RHMDs) that stochastically switch between different detectors. These detectors can be shown to be provably more difficult to reverse engineer based on recent results in probably approximately correct (PAC) learnability theory. We show that indeed such detectors are resilient to both reverse engineering and evasion, and that the resilience increases with the number and diversity of the individual detectors. Furthermore, we show that an optimal switching strategy between the RHMDs base detectors not only reduces misclassification on evasive malware but also maintains high classification accuracy on non-evasive malware. Our results demonstrate that these HMDs offer effective defense against evasive malware at low additional complexity.

Index Terms—HMDs, malware detection, evasive malware, adversarial machine learning

1 Introduction

Malicious attackers compromise systems to install malware. Preventing the compromise of systems is practically impossible: attackers obtain privileged access to systems in a variety of ways [1], [2]. While preventing compromise is difficult, detecting malware is also becoming increasingly more complicated. Indeed, modern malware is increasing in sophistication, challenging the abilities of software detectors. Typical techniques for malware detection such as dynamic binary instrumentation [3], information flow tracking [4], and software anomaly detection [5] have both coverage limitations and introduce substantial overhead (e.g., 10x slowdown for information flow tracking is typical in software [4]). These difficulties typically limit malware detection to static signature-based virus scanning tools [6] which have known limitations [7], allowing the attackers to bypass them and remain undetected.

In response to these trends, Hardware Malware Detectors (HMDs) have recently been proposed to make systems more malware-resistant. Several studies have shown that malware can be classified based on low-level hardware features such as instruction mixes, memory reference patterns, and architectural state information such as cache miss rates and branch prediction rates [8], [9], [10], [11]. In addition, the SnapDragon processor from Qualcomm appears to be using HMDs for online malware detection, although the technical details are not published [12]. At a time when malware developers appear to have the upper hand over defenders, HMDs can offer a substantial advantage to defenders because the detector is always on, has little impact on performance or power [13].

In this paper, we first explore whether attackers can adapt malware to create evasive malware; continue to operate while avoiding detection by HMDs. Should HMDs become widely deployed, it is natural to expect that attackers will attempt to evade detection. Intuitively, it should be possible for the attacker to evade detection if there are no restrictions placed on them, by running a mostly normal program and advancing the attack very slowly. However, we assume that the attacker would like to minimize the impact on the malware execution time since some attacks are time sensitive (e.g., covert/side-channels [14]) or computationally intensive [15]. We describe the threat model and limitations in Section 2.

We approach the question of whether malware can evade HMD, and whether HMDs can be made resilient to evasion, in the following steps:

1) Can HMDs be reverse-engineered? Recent results in adversarial classification [16] imply that arbitrarily complex but deterministic classifiers can be reverse-engineered. We confirm that this is the case for HMDs by reverse-engineering a number of detectors under realistic assumptions. We describe our dataset and methodology in Section 3, and present and analyze the reverse-engineered detectors in Section 4.
2) Having a model of the detector, can malware developers modify malware to avoid detection? Evading the detection by changing the behavior of the malware is known as mimicry attacks [17]. We show (in Section 5) that existing HMDs can be rendered ineffective using simple modifications to the malware binary.
3) Can the malware evade detection even if the detector is

- *M. S. Islam and K. N. Khasawneh are with the ECE Department, George Mason University. M.S. Islam is also with the CSE Department, Dhaka University of Engineering & Technology, Gazipur. E-mail:{mislam20, kkhasawn}@gmu.edu*
- *N. Abu-Ghazaleh is with the CSE and ECE Departments, University of California, Riverside. E-mail: naelag@ucr.edu*
- *D. Ponomarev and L. Yu are with the CS Department, Binghamton University. E-mail:{dima, lyu}@cs.binghamton.edu*

0018-9340 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: George Mason University. Downloaded on April 29,2021 at 07:49:57 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2021.3068873, IEEE Transactions on Computers

retrained with some samples of the evasive malware? We show in Section 6 that for simple evasion strategies that can fool a given detector, retraining a logistic regression (LR) detector does not result in effective classification of evasive malware, unless the detection performance on normal malware is sacrificed. In contrast, more sophisticated detectors such as Neural Networks (NN) can be successfully retrained, but the attacker is still able to reverse-engineer the retrained detector and evade it again.

4) After showing that the current generation of HMDs is evadable, we explore whether new HMDs can be constructed that are robust to evasion. In particular, we propose, in Section 7, a new resilient HMD (RHMD) organization that uses multiple diverse detectors and switches between them unpredictably. We show that RHMDs that use even simple base detectors are resilient to both reverse-engineering and evasion. Furthermore, this resilience increases with the number and diversity of the base detectors.
5) Having shown that the RHMD principle makes detection evasion-resilient, the next question we consider is how to control the base detectors' switching behavior to optimize detection and resilience jointly? To this end, in Section 8, we show that by formulating an optimization problem as a Bayesian Stackelberg game, we can improve the detection accuracy of RHMDs as well as their robustness against adversarial attacks.
6) Finally, we explore whether RHMDs fundamentally increase the difficulty of evasion or simply present another hurdle that can be bypassed by attackers. To this end, in Section 9, we overview recent results in Probably Approximately Correct (PAC) learnability theory that proves that RHMDs provide a measurable advantage in increasing the difficulty of reverse-engineering and complicate evasion. By making HMDs resilient to evasion, we bring them closer to practical deployment.

Evasive malware detection has been considered in the context of software malware detectors [17], [18]. Moreover, some existing HMD proposals discuss the possibility of malware evasion [8], [11]. However, ours is the first to explores this important question regarding HMDs in detail and develops solutions to it [19]. We note that while our experiments target HMDs, the underlying evasion problem exists in the context of any adversarial classification problem [16]. Our work advances the state of the art in general, not just for HMDs: we show systematically that reverse-engineering is possible, we develop techniques that use the result of reverse-engineered detectors to efficiently evade detection, and we introduce evade-retrain games and study their resilience to evasion.

In summary, the contributions of the paper are as follows:

- We show that it is possible to reverse engineer HMDs, regardless of their complexity accurately.
- We show that once an HMD has been reverse engineered, malware can effectively evade it using low overhead evasion strategies. This result brings into question the effectiveness of existing HMDs.
- We develop a new attack strategy to create evasive malware (called timer interrupt injection). This new strategy allows the attacker to create powerful evasive malware while drastically reducing the overhead of the attack; more than 80% and 60% less dynamic overhead compared to basic block, i.e., before every control flow altering instruction, and function level, i.e., before every return instruction, injections strategies respectively.
- We show that simple linear HMDs such as LR cannot be retrained to adapt to evasive malware. More complex classifiers such as NN can adapt better, but may break down after several generations of evasion and retrain. Moreover, new malware can still reverse-engineer and evade even such classifiers.
- We develop a new class of resilient HMDs (RHMDs) that operates by randomizing detection responsibility across different diverse detectors. RHMDs cannot effectively be reverse-engineered to enable evasion, which we verify both experimentally and using recent results from PAC learnability theory. The number and diversity of the base RHMD detectors increase the resilience to reverse-engineering and evasion.
- We formulate the RHMD detection and evasion problem as a Bayesian Stackelberg game to generate an optimal switching strategy between the base detectors of the RHMD that maximize the detection accuracy of both evasive and non-evasive malware. Our results indicate that optimized RHMD switching can detect more than 45% and 30% of the malware and evasive malware missed by the RHMD detection respectively.
- Lastly, we study the implementation complexity of such classifiers in hardware.

2 Threat Model and Limitations

In general, our threat model consists of an attacker that tries to re-write malware programs to bypass HMD's detection, given a black-box access to the targeted HMD. In particular, we assume an adversarial attack model that starts with the adversary attempting to reverse engineer HMD's detection model since the black-box access only allows the attacker to observe the classifier's behavior (output) for given programs (whether malware or normal programs). With reliable reverse-engineering information, the attacker can attempt to utilize the information to generate evasive malware that hides by changing some of their characteristics (feature values). Such an evasion mechanism is known as *mimicry* attacks [17], which can be in the form of no-op insertion, code obfuscation by the attackers, or calling benign functions in the middle of the malicious payload [20].

Furthermore, we assume that the attacker is interested in maintaining the malware's original performance after re-writing. If this assumption is not true, an attacker can simply run a regular program with embedded malware that advances the malware program arbitrarily slow (e.g., 1 malware instruction every N normal instructions where N is arbitrarily large), making detection impossible. Note that this is a limitation of all anomaly detectors, and not only HMDs. This assumption is also reasonable for important segments of malware such as: (a) malware that is time-sensitive (e.g., that performs covert or side-channel attacks [14]) and (b) computationally intensive malware such as that executing on botnets being monetized under a pay-per-install model [21] (e.g., Spam bots or Click fraud). Such malware has a utility to the malware writer proportional to their performance.

3 Data and Methodology

We collected malware samples from MalwareDB malware set [22]). The downloaded malware data set consisted of 3000 malware

0018-9340 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: George Mason University. Downloaded on April 29,2021 at 07:49:57 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2021.3068873, IEEE Transactions on Computers

programs. For regular program samples, we used Windows programs since the malware programs that we use are Windows-based. The regular program set contains a variety of applications, including browsers, text editing tools, system programs, SPEC 2006 benchmarks [23], and other popular applications such as Acrobat Reader, Notepad++, and Winrar. In total, the non-malware data set contains 554 programs. Both malware and regular programs data sets were divided into 60% *victim training*, 20% *attacker training*, and 20% *attacker testing* of the detector. The *Victim training* refers to the data set used to train the baseline HMDs, i.e., victim HMDs which the attacker is trying to adapt malware to bypass them; *attacker training* refers to the data set used by the attacker to reverse engineering the victim HMD; and *attacker testing* refers to the data set used by the attacker to create evasive malware programs as well as measuring the reverse-engineering effectiveness. To ensure that there is no bias in the distribution of malware programs across the sets, each set includes a randomly selected subset of malware samples from each type of malware in the overall data set.

The data was collected by running both malware and regular programs on a virtual machine with a Windows 7 operating system. To allow malware programs to carry out their intended functionality, the Windows security services and firewall were disabled. Furthermore, the dynamic traces of executed programs were collected using Pin instrumentation tool [24]. Unlike mobile malware where many malware samples require user interaction and necessitate special efforts to ensure correct behavior [25], we observed that the vast majority of our windows/desktop malware operated correctly (through manual inspection and examination malware behavior during run-time); several malware samples tripped the intrusion detection monitoring systems on our network as they attempted to discover and attack other machines, until we separated the environment into an independent subnet.

The collected trace duration for each executed program was 5000 system calls or 15 million committed instructions, starting after a warm-up period, whichever is reached first. While ideally, we would have liked to run each program longer, we are limited by the computational overhead; since we are collecting run-time behavior of the programs using dynamic profiling information through Pin within a virtual machine, the trace collection requires several weeks of execution on a small cluster and produces several terabytes of compressed profiling traces. We believe that this data set is sufficiently large to establish the feasibility and provide trustworthy experimental results. Our experimental evaluation is extremely computationally intensive; for each reverse-engineering experiment, we need to create (train) multiple reverse-engineered detectors and test all of them. Furthermore, we use the whole data for testing, i.e, we are doing multiple inferences (one per data sample). Each detection (inference) requires $0.3\mu s$. Finally, in all of our experiments, cross-validation was performed to avoid biased results, thus, we need to repeat our experiments multiple times across different subsets of the data.

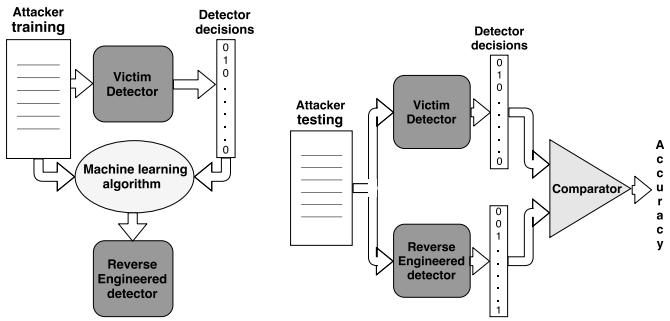
We collected different feature vectors, specifically:

- Executed instruction mixes (called *Instructions* in the rest of the paper): this feature tracks the frequency of instructions, i.e., number of times each opcode executed. Since the number of op-codes is large, we have used correlation analysis, as a feature selection method, on the training set to select the most correlated op-codes (instructions) to the training data labels.
- Memory address patterns (called *Memory* in the rest of the paper): this feature tracks the distribution of memory accesses. Specifically, we capture the memory access (read/write) pattern by calculating the distance between the memory address of the current load/store instruction and the memory address of the first load/store instruction in the features collection window. Subsequently, we created a histogram of read distances and write distances separately quantized into bins. For every features collection window, we store the frequency of each bin to create the feature vector.
- Architectural events (called *Architectural* in the rest of the paper): tracks the numbers of different architectural events occurring in an execution period such as unaligned memory accesses and taken branches.

These features are modeled after those used in prior HMD studies [8], [13].

4 Reverse-Engineering HMDs

This section demonstrates that we can successfully reverse-engineer HMDs based on supervised learning (e.g., similar to those presented in [8], [13]). Reverse-engineering the malware detector allows the adversary to construct a model of the HMD. Please note that this is different from IC reverse-engineering [26], as the adversary's goal is to reverse-engineer the detection model, e.g., weights and hyperparameters, rather than the logic of the HMDs' hardware implementation. The detection model is necessary to be able to methodically develop evasive malware. We assume that the adversary can query the targeted detector and observe it's detection output; if they do not, the problem becomes NP-Hard [16].



(a) Reverse-engineering of victim

(b) Evaluating reverse-engineered detector

Fig. 1: Overview of the reverse-engineering process

Figure 1a shows the steps in reverse-engineering a detector. First, the adversary prepares a training data set that is composed of both regular and malware programs: this is a separate data set from the one used to train the victim detector, which is unknown to the attacker. Next, the adversary uses this data set to query the victim detector and record its detection decisions. The decisions are used as the label for the data as we construct the reverse-engineered detector. Finally, the adversary may use different machine learning classification algorithms trained with the labeled data to build the new reverse-engineered detector.

Figure 1b shows the evaluation of the reverse-engineered detector. The adversary first prepares an attacker testing data set, as described above. Next, both the original detector and reverse-engineered detector are queried using the attacker testing data set.

0018-9340 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: George Mason University. Downloaded on April 29,2021 at 07:49:57 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2021.3068873, IEEE Transactions on Computers

Finally, the percentage of equivalent decisions made by the two detectors is calculated. Note that from the adversary point of view, it does not matter if the detector is classifying malware and regular programs correctly; instead, the attacker desires to mimic the victim detector's classification and evaluates success on that basis.

For most of our studies, we evaluate baseline detectors that use logistic regression (LR) and neural networks (NN); the methodology naturally generalizes to other classification algorithms. We implemented the NN classifier as a multi-layer perceptron (MLP) with a single hidden layer that has a number of neurons equal to the number of features in the feature vector. We use the *tanh* function as the activation function. The rationale for selecting these two algorithms is that prior studies showed that LR performs well and has low complexity, facilitating hardware implementations [13]. NN features more complex classification ability, capable of producing a non-linear classification boundary. These detectors allow us to contrast the detector complexity's impact on both the reverse-engineering process and mimicry attacks. For some studies, we use other classifiers to illustrate some generalizations of our conclusions.

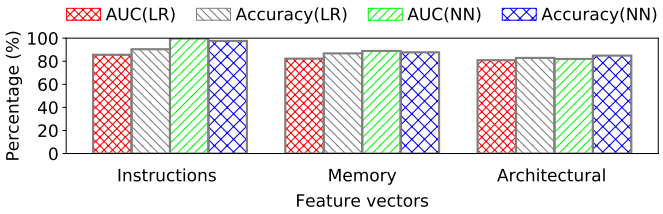


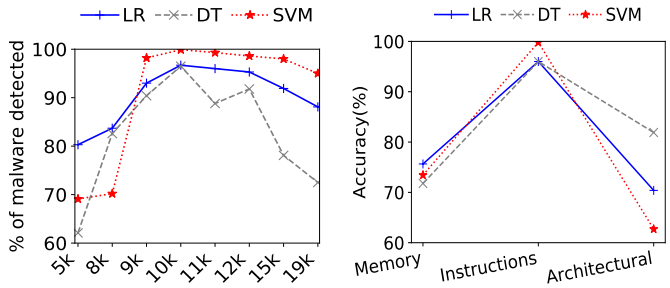
Fig. 2: Performance of individual detectors

The victim data set is used to train different detector instances using each of the two algorithms for each of the three different features, resulting in six detectors. Figure 2 shows the performance of the detectors in classifying malware and regular programs using area under the curve (AUC) and the accuracy of the classification metrics for each of the detectors. Note that this figure shows the performance of the baseline HMD, which we will be attempting to reverse-engineer. AUC is the area under the Receiver Operating Characteristics (ROC) curve which plots the sensitivity of the detector against the percentage of false positives; the larger the AUC, the better the classification. Accuracy refers to the ROC point, which maximizes the accuracy (percentage of decisions that are made correctly). It is a more direct measure of performance since the HMD classification threshold will be typically set to perform at or near this optimal point.

We assume that the attackers do not know the details of how the target victim detector was trained. Thus, they do not know important configuration parameters of the detector including: (1) the size of the instruction window that is used to collect the features; the detector collects the feature over a collection window, typically measured in thousands of instructions; (2) the specific feature used for the classification. However, we assume that the attacker has a set of candidate features that includes the feature used by the target detector; (3) the classification algorithm used by the target detector. Importantly, the attacker has access to a machine with a similar detector so they can test hypotheses and evaluate the success of the mimicry attacks. Next, we show how the attacker can reverse-engineer the detection period and the features used in training the target detector.

4.1 Target Detector Classification Period

The classification period refers to the size of the instruction window used to collect the classification features. Prior work [8] has shown that a classification period of about 10K instructions works well for supervised learning classifiers, but a detector may be trained with a different classification period. For this experiment, we used a classifier built using the Instruction mix feature, which we assume the attacker knows (later we relax this assumption). The target detector collection period is 10K. We prepare multiple pairs of attacker testing and training datasets, using different collection periods. Next, we train a reverse-engineered detector using different data sets and evaluate its accuracy. We construct three reverse engineered detectors using three machine learning algorithms for each of the attacker data sets. The machine learning algorithms used are LR, decision tree (DT), and support vector machine (SVM). The results of these experiments are shown in Figure 3a. The results show that the highest accuracy for reverse-engineering for each of the machine learning algorithms used is when the collection period is the same as the victim's collection period (10K). Thus, by using an experiment such as this one, the attacker can infer the victim's collection period.



(a) Collection periods (b) Features

Fig. 3: Reverse-engineer configurations

4.2 Target Detector Feature

Malware detectors can be built using different features. In this subsection, we explore reverse-engineering the victim detector's feature vector by only querying the victim detector. We use a detector based on the Instruction mix feature with a classification window of 10K instructions. We prepared multiple pairs of attacker testing and training data sets using the same collection period (10K), but using different feature vectors. Next, we construct reverse-engineered detectors using the attacker training data sets labeled with the victim detector's output as malware or regular program. For each of the attacker data sets, we constructed three detectors using different machine learning algorithms, which are: LR, DT, SVM. The results of this experiment are shown in Figure 3b. The results show that the highest accuracy is achieved when the feature vector is the same as the victim's feature vector (Instructions). We conclude that the victim HMD features can be successfully reverse-engineered.

Note that at the correct value of the feature and period, it is possible to obtain 0-error reverse-engineering in our experiments. This is consistent with results from PAC learning theory, which we overview in Section 9. Although we showed how to separately reverse-engineer the classification period (assuming that the classification feature is known) and the classification feature (assuming the classification period is known), we can also jointly reverse-engineer them both. The process involves constructing detectors

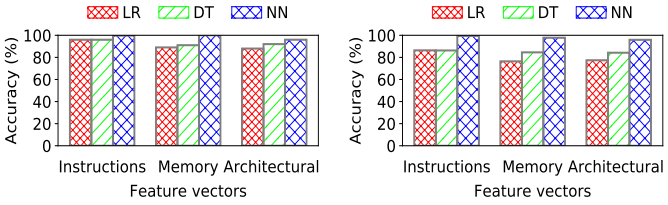
0018-9340 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: George Mason University. Downloaded on April 29,2021 at 07:49:57 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2021.3068873, IEEE Transactions on Computers

with different classification features and periods, and finding the detector that maximizes the reverse-engineering accuracy.

4.3 Performance of Reverse-engineered HMD

In the next set of experiments, we evaluate the performance of the reverse-engineered detectors. We reverse-engineer LR and NN detectors, but the reverse-engineered detector is constructed using three machine learning algorithms: LR, DT, and NN. The results are shown in Figure 4a and Figure 4b. The results show that NN can reverse-engineer both types of detectors with high accuracy (e.g., less than 1% error for all LR detectors). The performance is somewhat lower for NN since the separation criteria used in the classification is more complex and therefore more difficult to reverse-engineer accurately. As can be expected, the simpler linear detector (LR) cannot effectively capture the non-linear behavior of NN, consistent with PAC learning theory as we discuss in Section 9.



(a) Logistic regression (b) Neural networks

Fig. 4: Reverse-engineering efficiency

5 Developing Evasive Malware

After reverse-engineering the victim detectors, the next step that attackers are likely to take is to develop systematic transformations of their malware that can evade HMDs detection. The malware developers may modify their malware in any way, to attempt to produce behavior in the feature space of the detector that causes them to be classified as normal. Possible strategies to accomplish this goal include using polymorphism to produce different binaries [27]. However, since we are working with actual malware binaries, we do not have the source available to apply general transformations. Moreover, most of the malware is obfuscated, making decompilation difficult and challenge binary rewriting tools. To address these challenges, we developed a methodology to dynamically insert instructions into the malware execution in a controllable way (Figure 5). In particular, we either simulate the invocation of a timer interrupt or construct the Dynamic Control Flow Graph (DCFG) of the malware during execution by instrumenting it through the PIN tool [24]. Next, we add instructions into the control flow graph in a way that does not affect the program's execution state.

The injected instructions must change the feature vector in a controlled way based on the reversed-engineered classifier to attempt to move the malware across the classification decision boundary to be classified as normal. For the Instruction feature, the injection of opcodes increases the weight of the corresponding feature directly. For the memory feature, insertion of load and store instructions with controlled distances changes the histogram of memory reference frequencies. For architectural features, the effects may not be directly controllable. For example, increasing the cache hit rate or the branch predictor success rate requires inserting code segments that will generate cache misses or predictable branches, respectively. Without loss of generality, all of our experiments

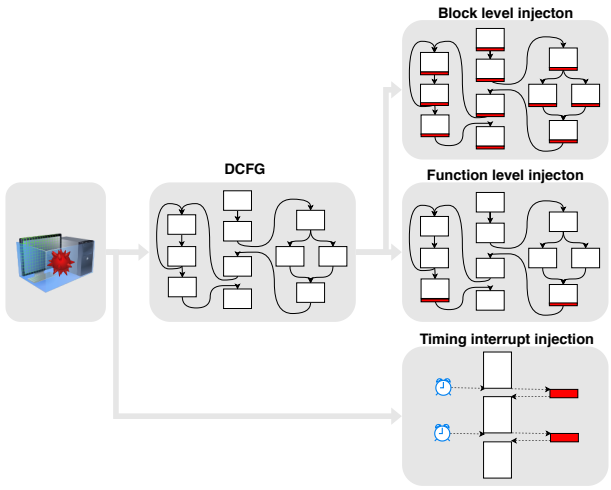


Fig. 5: Methodology for generating evasive malware

use the instruction feature. When attempting to change multiple features, the evasion code must combine these strategies (for example, alternating their use at different injection points).

We explore three approaches for injecting instructions: (1) Block level: insert instructions before every control flow altering instruction. Note that the instructions inserted at that point in the program are executed every time that control flow instruction is reached (i.e., we do not change the instructions that are injected at a particular point in the program, once they are injected); (2) Function level: insert instructions before every return instruction; (3) Periodic injection: in this strategy, we set a timer interrupt that injects instructions at a controllable frequency. Figure 7, demonstrate an example of injecting instruction at both block-level (e.g., before control flow altering instructions such as `je`) and functional-level (e.g., before `ret` instructions). Referring to Figure 5, block level and function level instruction injection requires construction of DCFG, while timing interrupt injection does not. We describe these two general approaches separately (in Subsections 5.1 and 5.2).

Random instruction injection: We first check if injecting randomly chosen instructions in the malware programs is sufficient to evade detection to establish that the injection must be specific to the detector. Each malware program data set is divided into two sets based on whether the victim detector successfully detected them without modification. Each of the data sets is modified using our framework to inject the additional instructions and retested; the results are shown in Figure 6. Clearly, injecting random instructions at the basic block level or the function call level does not help in evading detection.

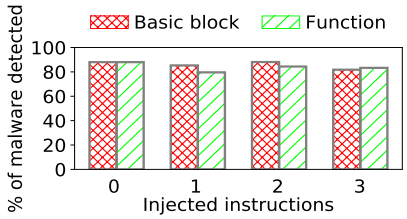


Fig. 6: Detection with random instruction injection

Reverse-engineering driven instruction injection: The next set of strategies we use is to exploit the information in the structure of the reverse-engineered detector to attempt to avoid

0018-9340 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: George Mason University. Downloaded on April 29,2021 at 07:49:57 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2021.3068873, IEEE Transactions on Computers

detection. Ideally, we would like to make the malware appear close to regular programs in terms of behavior so that the detector cannot successfully identify it. Therefore, we designed a strategy that injects instructions based on the parameters of the reverse-engineered detector (from Section 4) to make the detection difficult.

5.1 Block and Function Level Injection

We evaluate the success of the evasive strategies in avoiding detection. To understand the rationale behind the instruction selection, we must explain some details about the operation of LR. LR is defined by a vector, θ, that identifies the linear separation between the points being classified. The weights of the vector elements determine the relative importance of these elements. Since we are only adding instructions, we pick the instructions whose weights are negative to move the malware towards the other side of the separation line. In this first strategy, we inject only those instructions that have the least weights in the vector.

Original code	Block-level injection	Function-level injection

main: mov $3, %eax
mov $4, %ebx
call add1
sub $7, %eax
jz L1
imul %eax, %eax, $10
j L2
L1: idiv %eax, %eax, $10
L2: ret
...
add1: add %eax, %ebx
ret
	main: mov $3, %eax
	mov $4, %ebx
	or $0, %ebx
	call add1
	sub $7, %eax
	or $0, %ebx
	jz L1
	imul %eax, %eax, $10
	or $0, %ebx
	j L2
	L1: idiv %eax, %eax, $10
	or $0, %ebx
	L2: ret
	...
	add1: add %eax, %ebx
	or $0, %ebx
	ret
		main: mov $3, %eax
		mov $4, %ebx
		call add1
		sub $7, %eax
		jz L1
		imul %eax, %eax, $10
		j L2
		L1: idiv %eax, %eax, $10
		or $0, %ebx
		L2: ret
		...
		add1: add %eax, %ebx
		or $0, %ebx
		ret

Fig. 7: Examples of block and function levels instruction injection; the red colored instructions represent injected instructions.

Figure 9a, shows the percentage of malware detected by both the original and reverse-engineered detectors, after injecting the malware using the information for the reverse engineered detector at the basic block and the function levels. We observe that the modified malware evades detection by both detectors.

We conduct a similar experiment for the NN detectors, where the classifier is not clearly defined by a single vector and the separation plane is not linear. We develop a heuristic approach to identify candidate instructions for insertion. Figure 8 shows a NN with one hidden layer. Each circle in the figure represents a neuron (input, hidden, and output neurons) in the network. To compute the overall weight contributed by a single input, we multiply out its contributions to the network's eventual output and sum out these products. For the example in Figure 8, the weight of input I1 can be estimated as:

$$w_1 = w_{11}^1 \times w_1^{out} + w_{12}^1 \times w_2^{out} + w_{13}^1 \times w_3^{out}$$

More generally, for input j, the weight is:

$$w_j = \Sigma_{i=1}^{n} w_{ji} \times w_i^{out}$$

With multiple hidden layers, we must add all the factors on all the paths to which a given input contributes.

The procedure above allows us to collapse the NN description into a single vector that summarizes the contribution of each feature. This allows us to use the same strategies for instruction selection

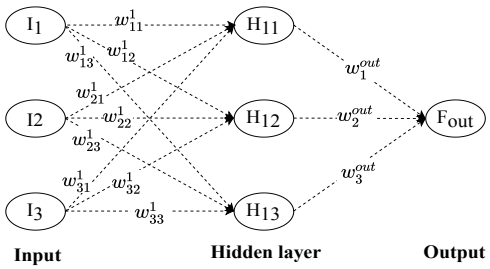
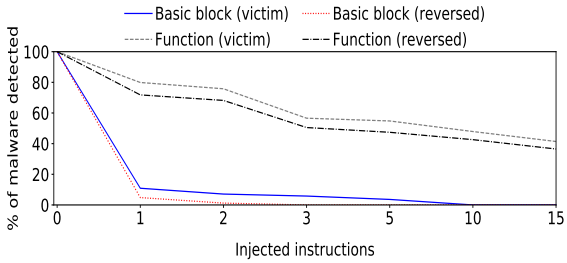


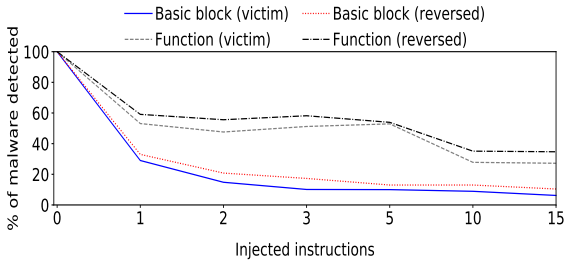
Fig. 8: Neural network with one hidden layer overview

that we used in LR; for example, we can select the instruction with the most negative weight for insertion. However, for NN, this is an approximate strategy; for LR, if we inject more of the negative weight instructions, we are guaranteed to monotonically decrease as the dot product of θ and the collected feature from the malware execution becomes increasingly more negative. However, the same cannot be guaranteed for NN because of its non-linear separation plane. For example, consider a feature vector element that contributes negatively to one neuron a and positively to another b. Without evasion, the malware is detected with a firing and b not firing. Initially, increasing the element's weight may turn a below the threshold leading the malware to evade detection. However, if we keep increasing the weight, b eventually fires, causing the malware to be detected again.

Figure 9b shows the percentage of modified malware detected by the NN victim and reverse-engineered detectors. While the evasive strategy also works in this case, it is slightly less effective; with 2 injected instructions per basic block, we can evade detection 80% of the time.



(a) Logistic regression



(b) Neural networks

Fig. 9: Detection with least weight injection

We assume that the attacker is interested in maintaining the malware's performance and does not want to arbitrarily slow it down to evade detection. Figure 10, shows the static and dynamic overhead of injecting instructions both at the basic block level and the function call level. Inserting a single instruction at the basic block level was effective in evading detection for most malware for

0018-9340 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: George Mason University. Downloaded on April 29,2021 at 07:49:57 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2021.3068873, IEEE Transactions on Computers

LR; both the static overhead (increase in the text segment of the executable) and the dynamic overhead (increase in execution time) is about 10%.

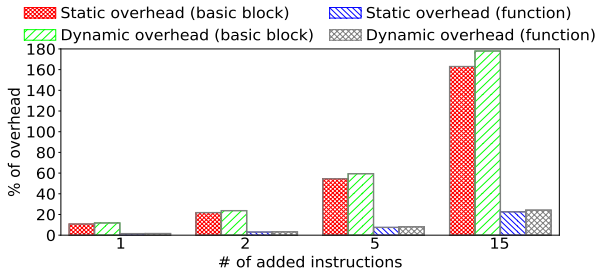


Fig. 10: Static and dynamic injection overhead

We also consider selecting the instruction for injection among all the instructions with negative weight, with a probability proportional to the weight; we call this strategy the *weighted injection strategy*. Figure 11, shows the percentage of malware detected by the victim, after weighted injection of the malware using the information for both the reverse-engineered detector and the victim detector at the basic block and at the function call levels. The evasion success using the reverse-engineered detector is almost equal to the success when using the actual victim detector. The advantage of this strategy is that it makes it more difficult to detect the evasion if the detector is retrained as explained in the next section.

5.2 Timing Interrupt Injection

In both the block-level and function-level injection strategies, we have little control over the frequency of appearance of the injected instructions in the execution stream of the modified malware. For example, if the basic blocks are long in the program execution period, the injection may become infrequent and the malware may get exposed. Moreover, if the basic blocks are short, the injection's overhead may increase, and it may be easier to detect the injection due to the high frequency of the injected instructions. For these reasons, we investigate an injection policy that uses a timer interrupt to control the timing of the injection and the number of injected instructions. Figure 12 shows the percentage of malware detected by the victim when we use this policy as a function of the injected instructions per 10K regular instructions. At around 500 injected instructions, the most negative instruction policy allows almost all malware programs to evade detection, which translates to more than 80% less dynamic overhead compared to basic block injection. In addition, for the weighted injection policy, this occurs at around 1000 injected instructions, which translates to more than 60% less dynamic overhead compared to basic block injection.

6 Retraining Victim Detectors

The previous section results demonstrate that existing HMDs that use supervised learning [8], [13] can be fairly easily evaded. The next question we consider is: if a detector is retrained with the addition of evasive malware samples in the training data set, would it be able to classify them correctly? If the answer is yes, then perhaps the weights can be updated regularly to allow the detector to adapt to emerging malware. However, there is still a possibility that the retrained detector could itself be reverse-engineered and evaded again. Moreover, as attackers continue to evolve, it is not

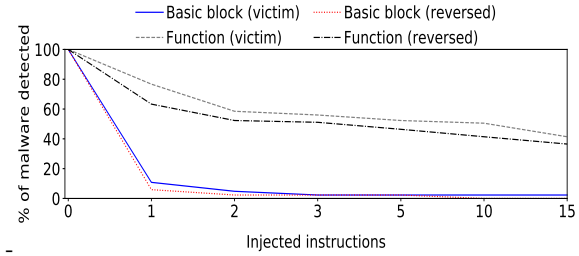


Fig. 11: Detection with weighted injection (LR)

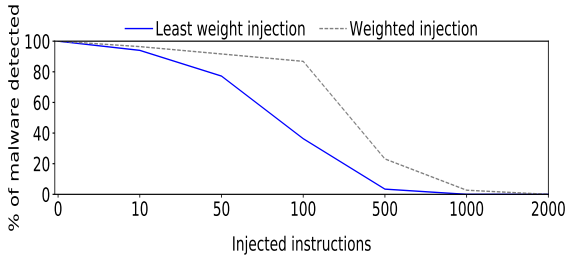
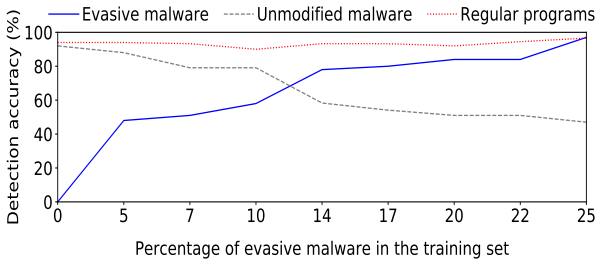


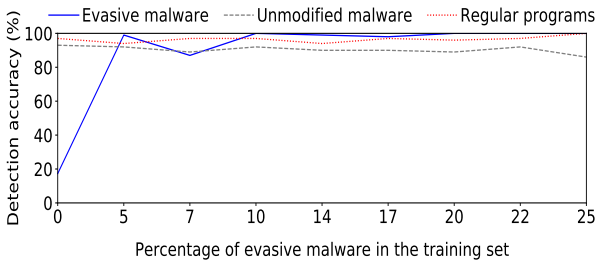
Fig. 12: Timing interrupt attack

clear if the detector will eventually become ineffective due to the number of classes it is attempting to separate or converge to a classification setting that is impossible to evade.

Figure 13a shows the effect of increasing the percentage of evasive malware programs in the training data that we use to retrain the simple LR detector. For example, the point with 10% indicates that 10% of the malware part of the training set consists of evasive malware, modified with one of our evasion strategies. We see that, in general, increasing the percentage of evasive malware leads to more accurate detection. Unfortunately, this comes at the cost of losing accuracy for non-evasive malware, making simple retraining an ineffective strategy. Furthermore, it is interesting to see that the accuracy in classifying regular programs does not degrade.



(a) Logistic regression



(b) Neural networks

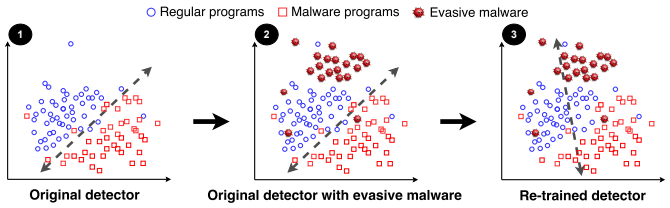
Fig. 13: Effectiveness of retraining

Figure 14a illustrates why linear detectors, such as LR, sacrifice

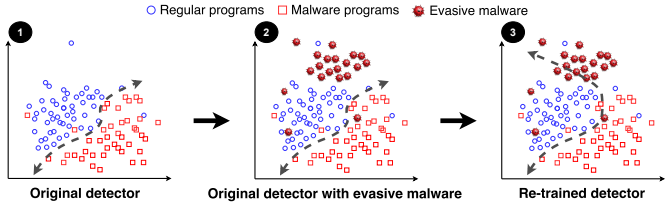
0018-9340 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: George Mason University. Downloaded on April 29,2021 at 07:49:57 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2021.3068873, IEEE Transactions on Computers

accuracy when retrained. Figure 14a(1) shows that there is a linear separation between the malware and regular programs. Figure 14a(2) demonstrates that the evasive malware have to cross the separation boundary in order to evade detection. Figure 14a(3) shows that with the retrained detector, it may be impossible to find a linear separation between malware (including evasive malware) and regular programs. In contrast, non-linear classifiers such as NN (Figure 13b) are able to detect this new form of malware with high accuracy, even with a low percentage of evasive malware in the retraining set. This can be achieved without affecting the detection accuracy of non-evasive malware or regular programs. Figure 14b illustrates why non-linear detectors are more effective when retrained. Even when evasive malware crosses the original classification's separation boundary, a new non-linear boundary can be found that separates the two malware classes from normal programs. Thus, HMDs must be non-linear if we want to retrain them in response to evasive malware detection.



(a) Linear model



(b) Non-Linear model

Fig. 14: Illustration of effect of retraining a linear and non-linear classifiers with data that includes evasive malware

Figure 15 shows the detection of several generations of NN detectors. n each generation, we repeat the detector's retraining by adding malware from the previous generations to the training set. The original detector in generation 1 is first evaded successfully as we see low detection for evasive malware. After we retrain, we see that evasive malware developed to evade detector 1 is now detected successfully (rightmost bar for detector 2). However, if we reverse-engineer the detector and evade it again, we can do so successfully as evidenced by the low detection of the evasive malware in the third bar for detector 2. As the retrain-evade process is continued, we expected one of two outcomes: (1) the detector will no longer be able to classify; or (2) the decision boundary will tighten and malware will no longer be able to evade. After 7 generations, the detector can no longer be trained successfully as malware and normal programs became inseparable using our NN. There are two possible explanations: (1) the feature is not sufficiently discriminative, and it is possible to turn malware to be similar to normal programs with respect to this feature. Note that in each successive generation, the overhead is increased, and this level of overhead may not be acceptable to the attacker; or (2) NN could no longer represent the complex decision boundary between the different classes of evasive malware and normal programs, similar to how LR failed after one generation.

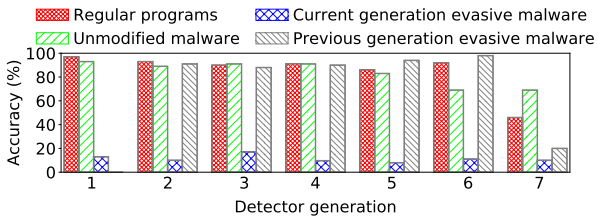


Fig. 15: Detection after retraining over several generations

7 Evasion-Resilient HMDs

Although retraining the detectors detect evasive malware that has representatives in the training set, we showed that retraining might eventually fail as attackers continue to evade. Moreover, retraining cannot detect novel evasive malware: they can be reverse-engineered and evaded even after retraining.

In this section, we introduce a new class of evasion-resilient HMDs (RHMDs). RHMDs leverage randomization to make detectors resistant to reverse-engineering and, thus, evasion. In particular, randomization introducing an error to the attacker's reverse engineering process. The introduced error increases while increasing the number and diversity of the base RHMD detectors; we show a proof for this claim in Section 9 based on PAC learnability theory.

We randomize two detectors' settings: i) The feature vectors used for detection; and ii) The collection periods used in the detection. In particular, we construct detectors with these heterogeneous features and switch between them stochastically in a way that cannot be predicted by the attacker.

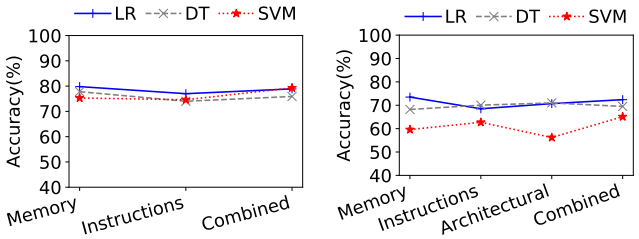
Our first study examines the effect of randomizing the feature vectors used for detection. We start with two detectors using the same detection period. The results of this experiment are shown in Figure 16a. We reverse-engineer the detector using two of the original feature vectors as well as a combination of them. In particular, the point in the figure marked "combined" represents reverse-engineering with a combined detector using the union of the two feature vectors. Using an RHMD with two detectors, reverse-engineering the detector becomes substantially more difficult because the model now includes two diverse detectors which are selected randomly. The diversity can be further expanded by using a pool of three detectors — the results of this approach are shown in Figure 16b. Again, the combined point on the figure refers to a reverse-engineering attempt using the union of the three feature vectors of the three individual detectors. As seen from the results, reverse-engineering becomes harder with increased diversity.

To further increase detector diversity, we construct detectors with two different collection periods (10K cycles period and 5K cycles period), resulting in a pool of six detectors, which are randomly chosen by the detection logic. The results are presented in Figure 17. Consistent with the previous trend, additional diversity makes reverse-engineering even more difficult. Note that having detectors operating on the same features with different periods does not substantially increase the hardware complexity; the different weights for the two detectors must be kept separately, but the collection logic and the detector evaluation logic are shared.

Having reverse-engineered the detector, we use our evasion framework to inject instructions to evade it. Given that the reverse-engineering becomes inaccurate in RHMDs and given the random switch between the individual detectors, the constructed evasive malware can no longer hide from detection (Figure 18). It is

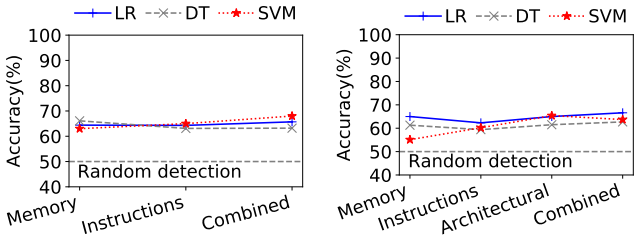
0018-9340 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: George Mason University. Downloaded on April 29,2021 at 07:49:57 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2021.3068873, IEEE Transactions on Computers



(a) Two feature vectors (b) Three feature vectors

Fig. 16: RHMD reverse engineering (features)



(a) Two feature vectors (b) Three feature vectors

Fig. 17: RHMD reverse engineering (features and periods)

interesting to note that the higher the detector's diversity, the more resilient it is to evasion, consistent with PAC learnability theory discussed in Section 9. These results demonstrate that this approach to constructing HMDs provides resilience to evasive malware. The average detection accuracy of the RHMD without evasion (Figure 18 with 0 injected instructions) is equal to the average accuracy of its base detectors since the randomization selects between the detectors with equal probability. Thus, the average loss of detection due to randomization is the difference of accuracy between the most accurate detector and the average of all base detectors.

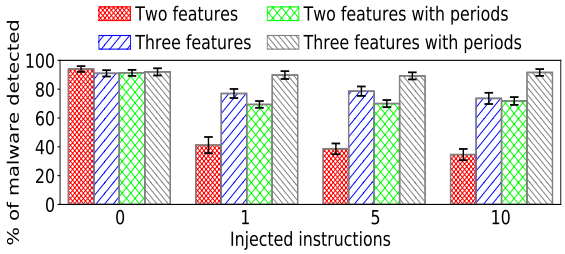


Fig. 18: RHMD evasion resilience

For online detection, we evaluate hardware implementation overhead by implementing RHMDs using Verilog, as an extension of an open source x86-compatible core (AO486) [28]. The detectors collect information from the CPU pipeline commit stage and apply the detection logic at the detection period. We found that the core frequency was not effected by adding the RHMD implementation, but this was not surprising because RHMDs are implemented off the critical path. Furthermore, we tried to run malware executables on the AO486 core but, unfortunately, we were not successful since the software ecosystem is different, and we were not able to boot Windows on the AO486 core. After synthesizing the new core implementation on an FPGA board for a configuration with three detectors corresponding to the three features with the same period, we observed that the area and power increase is modest: 1.72% and 0.78%, respectively. Note that the resilient detectors can also be used to make offline detection [8] resilient to evasion.

8 Opt-RHMD: Optimal Switching Strategy

After demonstrating that the RHMD framework is resilient to reverse-engineering and thus evasion attacks (Section 7), in this section, we explore whether we can optimally configure RHMDs for performance while retaining or improving resilience. Specifically, since the RHMD uses a uniform random switching between the based detectors, in this section, we explore finding an optimized switching strategy that would guarantee high detection accuracy and retain robustness of RHMDs.

In principle, the RHMD framework is similar to the Moving Target Defense (MTD) framework that is used in software security. In general, MTD systems' goal is to reduce the attack surface (success rate) by proactively switching between multiple software configurations [29]. However, for RHMDs (a classification task), a uniform random switching strategy might reduce overall detection accuracy because the base detector with the highest detection accuracy is not always selected [30]. Thus, in order to design an effective switching strategy, that retains a good detection accuracy and guarantees robustness against evasion attacks, we have to reason about attacks in a multi-agent game theoretic fashion, which facilitate providing formal guarantees about the security of such systems.

In a malware detection framework, the RHMD (defender/leader) selects a base detector up-front for the next detection. The inputs (user) to RHMDs can be legitimate (not evasive/modified) or adversarial (evasive/modified) programs. In addition, an attacker can observe (or follows) the RHMD over time, i.e., have black-box access to the RHMD, before choosing an evasion strategy, e.g., the instruction injection method and where to inject the instructions. These characteristics motivated us to formulate the RHMD system as a repeated Bayesian Stackelberg Game (BSG) [31]. In a BSG, the leader/defender (RHMD) starts the game by selecting a move (a base detector). The user can observe a finite number of moves and probabilistically learn the defender's switching strategy. Therefore, the defender has to choose a move (a base detector) that maximizes the defender reward (detection accuracy and robustness) in this game, given that the user (attacker) knows the defender strategy, which can be formulated as finding the Stackelberg Equilibrium in a Bayesian Game. We describe the formulation and show our experimental results for the optimal-RHMD switching in the following subsections.

8.1 Stackelberg Formulation of OPT-RHMD

Defender: The defender (RHMD) configuration space consist of six base detectors, N={*Instructions*-5K, *Instructions*-10K, *Memory*-5K, *Memory*-10K, *Architectural*-5K, *Architectural*-10K}, which correspond to the best performing RHMDs in Section 7. Note that each base detector name represents the feature vector and the detection period used to build that detector. For the purpose of modeling, let $\mathscr{D}$ represent the RHMD. The RHMD follows a uniform random probability distribution across the configuration space to select a base detector n ($\in N$) for detecting each input. Consequently, this model selection policy creates an equal chance of choosing the models that have low accuracy or high vulnerability to evasion, resulting in potentially a sub-optimal switching strategy.

Users: Users of the RHMDs are divided into two groups: Legitimate users ($\mathscr{L}$) and Adversary ($\mathscr{A}$), where $\mathscr{L}$ provide un-modified

0018-9340 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: George Mason University. Downloaded on April 29,2021 at 07:49:57 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2021.3068873, IEEE Transactions on Computers

TABLE 1: Reward values of the defender, legitimate user, and adversarial user for various attacks; rewards in column $\mathscr{L}$ follows $\left(R^{\mathscr{D}'}, R^{\mathscr{L}}\right)$ format and in column $\mathscr{A}$ follows $(\mathrm{R}^{\mathscr{D}'}, R^{\mathscr{A}})format$.

RHMD base detectors	Legitimate user, $\mathscr{L}$	Adversarial, $\mathscr{A}$, attack on					
		Instructions – 5K	*Instructions* – 10K	*Memory* – 5K	*Memory*10K	*Architectural* – 5K	*Architectural* – 10K
Instructions – 5K	(94, 94)	(2, 98)	(4, 96)	(81, 19)	(80, 20)	(77, 23)	(80, 20)
Instructions – 10K	(96, 96)	(7, 93)	(2, 98)	(83, 17)	(84, 16)	(78, 22)	(82, 18)
Memory – 5K	(90, 90)	(82, 18)	(82, 18)	(15, 85)	(19, 81)	(76, 24)	(77, 23)
Memory – 10K	(90, 90)	(84, 16)	(85, 15)	(17, 83)	(14, 86)	(72, 28)	(71, 29)
Architectural – 5K	(84, 84)	(87, 13)	(87, 13)	(80, 20)	(83, 17)	(5, 95)	(5, 95)
Architectural – 10K	(85, 85)	(87, 13)	(88, 12)	(86, 14)	(83, 17)	(7, 93)	(4, 96)

input, i.e., not trying to evade the detection, while $\mathscr{A}$ tries to evade detection by providing evasive malware. We use the notation u_n to represent the evasive malware set that is created by the evasion strategy u $(\in U)$ using the model of base detector n $(\in N)$.

For every usage of a base detector, we calculate a reward value for each player. (1) *Interaction between $\mathscr{D}'$ and $\mathscr{L}$*: when $\mathscr{D}'$ and $\mathscr{L}$ uses a base detector, they both get a reward, which is equal to the accuracy of the detector, e.g., using a detector with 92% accuracy gives a reward of 92 to both $\mathscr{D}'$ and $\mathscr{L}$. (2) *Interaction between $\mathscr{D}'$ and $\mathscr{A}$*: For attacking a base detector n with attack u, an adversary $\mathscr{A}$ gets a reward, which we model simply as the expected success rate. Conversely, we assume the defender's reward is equal to its detection accuracy for that evasive malware input. For instance, considering an attack success rate of 80%, the attacker's reward is 80 and the defender's reward is $100-80=20$, although different weighting of reward is possible, for example, to emphasize either detection or resilience. In our formulation, we use $R^{\mathscr{D}'}_{n,u}$, $R^{\mathscr{L}}_{n,u}$, and $R^{\mathscr{A}}_{n,u}$ to denote the reward value of $\mathscr{D}'$, $\mathscr{L}$, and $\mathscr{A}$ respectively. To generate the reward for each of the players, we created an evasive malware set for each of the six RHMD base detectors (N), using the *attacker testing* set. Next, we evaluated the detection performance for each of the RHMD base detectors to each of the evasive malware sets. The results are shown in Table 1.

Game formulation and optimization: The interaction between defender ($\mathscr{D}'$) and its users ($\mathscr{L}$, $\mathscr{A}$) are modeled as Bayesian Stackelberg game, where the defender plays first, i.e., choose a base detector. Next comes the user (attacker) as the second player, where $\mathscr{A}$ tries to maximize the defender's loss function by inputting evasive malware. In contrast, the defender's goal is to reduce the success rate of evasion, i.e., minimizing loss function against $\mathscr{A}$, and maintain high detection accuracy on non-evasive programs, i.e., minimizing loss function against $\mathscr{L}$, which is a multi-objective optimization problem. The importance of each objective can be modeled as the defender's prediction of the user ($\mathscr{L}$ or $\mathscr{A}$). Satisfying this multi-objective constraint is equivalent to finding the Stackelberg equilibrium of the game. Authors in [32] formulated the Mixed Integer Quadratic Program (MIQP), briefly listed below, which yields the equilibrium point of the game.

Suppose, $\mathbf{x}$, $\mathbf{q}^{\mathscr{L}}_u$, and $\mathbf{q}^{\mathscr{A}}_u$ denotes the strategy vector of $\mathscr{D}'$, $\mathscr{L}$, and $\mathscr{A}$, respectively. In addition, let us denote the rewards of the defender selecting strategy detector n and the user selecting strategy u as $R^{\mathscr{D}'}_{n,u}$, $R^{\mathscr{L}}_{n,u}$, and $R^{\mathscr{A}}_{n,u}$, for $\mathscr{D}'$, $\mathscr{L}$, and $\mathscr{A}$, respectively. Basically, the strategy vector denotes the set of choices each player can make and the goal of each player is to maximize the received rewords. Our goal is to choose a strategy that maximizes the defender's reward while allowing the attacker to choose the most effective attack strategy. In particular, we need to solve the following optimization problem to calculate the defender's optimal switching strategy:

$$\max_{x,\,q} \sum_{n\in N}(\alpha\cdot\sum_{u\in U}R^{\mathscr{D}'}_{n,u}\,x_n q^{\mathscr{A}}_u+(1-\alpha)\cdot R^{\mathscr{D}'}_{n,u}\,x_n q^{\mathscr{L}}_u) \tag{1}$$

$$\text{s.t.}\quad \sum_{n\in N}x_n=1 \tag{2}$$

$$0\le x_n\le 1\quad \forall n\in N \tag{3}$$

$$\sum_{u\in U}q^y_u=1;\quad q^y_u\in\{0,1\},\quad \forall y\in\{\mathscr{A},\mathscr{L}\} \tag{4}$$

$$0\ \le\ v^y-\sum_{n\in N}R^y_{n,u}x_n\ \le(1-q^y_a)M;\quad \forall u\in U^y,\ \forall y\in\{\mathscr{A},\mathscr{L}\} \tag{5}$$

where α is the probability of the system being used by user $\mathscr{A}$, i.e., the input is evasive malware, and M is large positive constant. Furthermore, the constraints in (2), (3), and (4) are for guaranteeing that the probabilities assigned for a strategy vector selection sum up to 1. The final constraint in (5) is for an attacker's optimization purpose, who wants to maximize its reward (v^y) against the defender's selected strategy. To solve the MIQP optimization problem, we utilized the calculated rewards values in Table 1 and the Gurobi environment [33]. The resulting selection strategy for the Opt-RHMD is shown in Figure 19, which shows the best probability of selecting each of the 6 based detectors compared to a uniform random selection. Note that the resulting optimal selecting strategy uses only 4 of the 6 base detectors to achieve high detection accuracy while maintaining robustness.

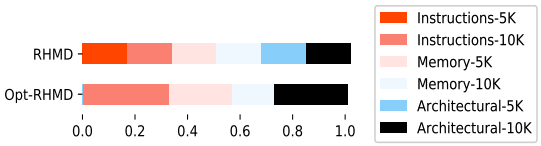


Fig. 19: OPT-RHMD switching probability

8.2 Opt-RHMD Experimental Evaluation

Our first study examines the reverse engineering (Section 4.3) effectiveness of Opt-RHMD. The results of this experiment are presented in Figure 20. We reverse-engineer the detector using all feature vectors as well as a combination of them (marked as *"combined"* in the Figure). Consistent with the previous results (Section 7), reverse-engineering the Opt-RHMD is substantially more difficult, compared to reverse engineering both HMDs as well as the most diverse RHMD in our experiments (Figure 17b).

Furthermore, based on the obtained reverse-engineered detector, we use our evasion framework to inject instructions to evade it. Given that the reverse-engineering becomes inaccurate in Opt-RHMDs, the constructed evasive malware can no longer hide from detection (Figure 21). In addition, given the optimized switching between individual base detectors, Opt-RHMD can detect more than 45% of the malware missed by the RHMD. Moreover, in the case of evasive malware, opt-RHMD can detect more than 30% of the evasive malware missed by RHMD.

0018-9340 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: George Mason University. Downloaded on April 29,2021 at 07:49:57 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2021.3068873, IEEE Transactions on Computers

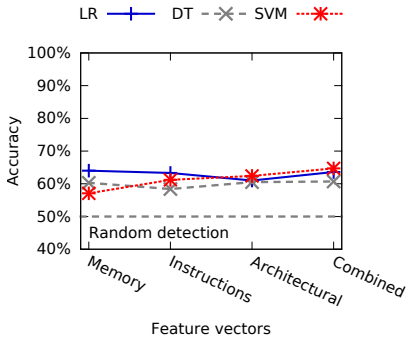


Fig. 20: OPT-RHMD reverse-engineering

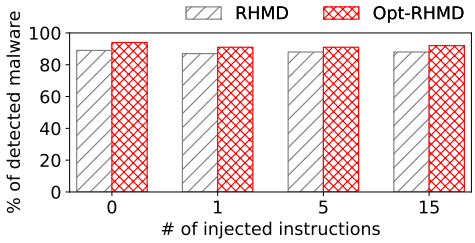


Fig. 21: OPT-RHMD evasion resilience

9 Theoretical Basis for RHMD

This section provides theoretical support for the resilience of RHMD for evasion based on probably approximately correct (PAC) learnability theory [34]. In particular, we show that randomized classification is inherently more difficult to reverse-engineer than a deterministic classifier, even one with arbitrarily high complexity.

9.1 Learnability of Deterministic Classification

Consider a learning system (a defender) that uses a single classifier to classify malware from normal programs. Consider another *reverse engineering* learning system (an attacker) that uses past classification data, e.g., by repeatedly querying the defender classifier, to determine with high accuracy the nature of the defender classifier.

Formally, let H be the class of possible classifiers (also called hypothesis class) a learning system considers. Let P be the probability distribution over instances (x,y), where x is an input feature vector, and y is a label in $\{0,1\}$. We assume below that $y=\bar{h}(x)$, i.e., a deterministic function that gives the true label of x. For any $h\in H$, let $e(h)=\Pr_{x\in P}[h(x)\neq\bar{h}(x)]$ be the expected error of h w.r.t. P. We define $e_H=\inf_{h\in H}\ e(h)$ as the optimal (smallest) error achievable by any function $h\in H$. Let $D=\{(x_1,y_1),...,(x_m,y_m)\}$ be a training data set of size m generated according to P and $\mathscr{D}$ be the set of all possible D. A learning algorithm is a function $L:\mathscr{D}\rightarrow H$ which produces a classifier $\hat{h}\in H$ given a training set D.

Definition 1. *A hypothesis class H is learnable if there is a learning algorithm L for H with the property that for any $\varepsilon,\delta\in(0,1/2)$ and distribution P there exists a training sample size $m_0(\varepsilon,\delta)$, such that for all $m\geq m_0(\varepsilon,\delta)$, $\Pr_{D\in\mathscr{D}}[e(L(D))\leq e_H+\varepsilon]\geq 1-\delta$, i.e., L will with probability at least $(1-\delta)$ output a hypothesis $\hat{h}\in H$, whose error on P is almost $(e_H+\varepsilon)$. H is efficiently learnable if $m_0(\varepsilon,\delta)$ is polynomial in $1/\varepsilon$ and $1/\delta$, and L runs in time polynomial in m, $1/\varepsilon$ and $1/\delta$.* [1]

1. This definition is taken, in a slightly extended form, from [34].

The definition above says that a hypothesis class H is efficiently learnable (i.e., learning is easy) if we can compute with high probability an approximately optimal candidate from this class given a polynomial number of samples. The error bound $(e_H+\varepsilon)$ for an approximately correct classifier $\hat{h}\in H$ consists of two components. ε becomes arbitrarily small and hence $e(\hat{h})$ approaches e_H when the number of training samples increases polynomially w.r.t. $1/\varepsilon$. The other component e_H depends on the learning bias [34] about H. That is the set of assumptions that the learner makes (e.g., the type of classifiers and the underlying features). With a good choice of H, e_H can be arbitrary small; e_H is zero if H contains the true classifier $\bar{h}$. We observed the implications of this result in Section 4 when the correct feature and detection period lead to the highest accuracy reverse engineered classifier.

The concept of PAC learnability applies to the learning tasks by both the defender and the attacker, with one caveat: for the defender, the goal is to correctly predict the *true* label of an instance (i.e., $y=\bar{h}(x)$); while for the attacker, the goal is to correctly predict the label of an instance *assigned* by the defender's classifier (i.e., $y=\hat{h}(x)$). As shown in [16], efficient learning for $\bar{h}$ by the defender implies efficient learning for $\hat{h}$ (called efficient reverse-engineering) by the attacker. Suppose that the defender has learned $\hat{h}$ from an efficiently learnable H. Provided the attacker identifies the type and features of the classifiers in H, then $\hat{h}$ is contained in the hypothesis class used by the attacker, and $e_H=0$, i.e., the distribution P over $(x,\bar{h}(x)$ can be efficiently reverse-engineered with arbitrary precision [16]. Without prior knowledge of H, the attacker can tune its hypothesis class based on the error rate on the training data collected over repeated queries.

These results support the reverse-engineering experiments in Section 4. In particular, the analysis shows that *reverse-engineering a deterministic classifier is "easy" in practice regardless of the complexity of the defender's classifier*. Increasing the complexity of the defender's classifier can make it more costly to reverse-engineer it, but will not change the arms race's outcome between the defender and attacker with respect to the difficulty of evasion.

9.2 Learnability of Randomized Classification

We now consider a defender that uses randomized classification such as the model used in RHMD. As before, consider a distribution P over instances (x,y), with $y=\bar{h}(x)$ as the ground truth. Suppose that we have n hypothesis classes H_i, all efficiently learnable. Let $\hat{h}_i\in H_i$ be the classifier learned from these classes, respectively, with the corresponding error rate $e(\hat{h}_i)$. Additionally, let $\triangle_{i,j}=\Pr_{x\in P}[\hat{h}_i(x)\neq\hat{h}_j(x)]$ for all i,j: that is, $\triangle_{i,j}$ measures the difference between two classifiers $\hat{h}_i(x)$ and $\hat{h}_j(x)$ over the data distribution. Consider a space of policies parametrized by $p_i\in[0,1]$ with $\sum_i p_i=1$, where we choose $\hat{h}_i\in H_i$ with probability p_i. Let $\vec{p}$ denote the corresponding probability vector. Then, a policy $\vec{p}$ induces a distribution $Q_{\vec{p}}$ over (x,z), where $z=\hat{h}_i(x)$ with probability p_i. The defender will incur a baseline error rate of $e_{\vec{p}}(h)=\Pr_{x\in Q_{\vec{p}}}[\hat{h}_i(x)\neq\bar{h}(x)]=\sum_i p_i e(\hat{h}_i)$ if there is no reverse-engineering effort.

Suppose the attacker observes a sequence of data points from $Q_{\vec{p}}$, and tries to efficiently learn the hypothesis class $H=\cup_i H_i$. For any $h\in H$, let $e_{\vec{p}}(h)=\Pr_{x\in Q_{\vec{p}}}[h(x)\neq\hat{h}_i(x)]=\sum_i p_i e(h)$, the expected error of h w.r.t. $Q_{\vec{p}}$, and we define $e_{\vec{p},H}=\inf_{h\in H}\ e_{\vec{p}}(h)$ as the optimal (smallest) error achievable by any function $h\in H$ under a policy $\vec{p}$. Definition 1 naturally extends to the randomized setting: in particular, the distribution P becomes $Q_{\vec{p}}$ and the error bound $(e_H+\varepsilon)$ becomes $(e_{\vec{p},H}+\varepsilon)$.

0018-9340 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: George Mason University. Downloaded on April 29,2021 at 07:49:57 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2021.3068873, IEEE Transactions on Computers

Theorem 1. *Suppose that each H_i is efficiently learnable, and $\hat{h}_i \in H_i$ be the classifier learned from these classes by a defender, respectively, with the corresponding error rate $e(\hat{h}_i)$. Then, any distribution $Q_{\vec{p}}$ over (x,z) can be efficiently reverse engineered, with $e_{\vec{p},H}$ bounded by* $\min_i \sum_{j\neq i} p_i \triangle_{i,j} \leq e_{\vec{p},H} \leq 2(\max_i e(\hat{h}_i))$. [2]

This theorem shows that on the one hand, even with randomization, reverse-engineering is easy as long as all classifiers among which the defender randomizes accurately predict the target - that is $\max_i e(\hat{h}_i)$, the maximal error among the n classifiers, is arbitrarily small. On the other hand, the attacker's error depends directly on the difference among the classifiers, which can be significant if at least some of the classifiers are not very accurate, allowing them to disagree more often. According to the error bound $(e_{\vec{p},H} + \varepsilon)$, even though ε becomes arbitrarily small as the number of queried samples increases, the defender will inevitably suffer from an error caused by $e_{\vec{p},H}$. This error can be high; for example, when randomizing two classifiers of error 0.2 and 0.1 with $p_1 = p_2 = 0.5$, $e_{\vec{p},H}$ is in [0.15, 0.4]. In contrast, in the deterministic setting, e_H can be 0. For our experiments with the pool of six detectors we measured the error to be around 25% on our testing dataset.

The above theorem also suggests a trade-off between the accuracy of the defender under no reverse-engineering vs. the susceptibility to being reverse-engineered: using low-accuracy but high-diversity classifiers allow the defender to induce a higher error rate on the attacker, but will also degrade the baseline performance against the target. To combat reverse engineering effort, we propose to randomize among a set of low-complexity, low-accuracy classifiers (e.g., logistic regression), rather than deploying a single high-complexity, high-accuracy classifier (e.g., deep neural network or random forest). The former is also more suitable for a hardware implementation than the latter. Although this low accuracy applies to the classification of each individual period, we raise the overall accuracy of the detector by averaging the decisions across multiple intervals.

9.3 Evasion Without Reverse Engineering

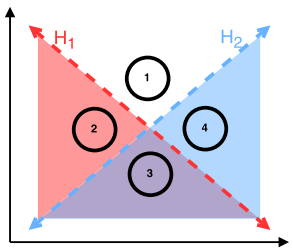


Fig. 22: Impact of Randomization on Evasion

Our threat model assumes that an attacker needs to reverse engineer a detector before evading it. The theoretical resilience claims on RHMDs rely on the difficulty of this reverse engineering. In this section, we consider whether it is possible to evade the detector without reverse engineering. To provide intuition, we start with Figure 22, which shows the decision boundaries of two diverse base detectors learned from hypothesis classes H_1 and H_2. The two decision boundaries are not mutually exclusive (H_1 malware regions are 2, 3 and H_2 malware regions are 3, 4). To fool both detectors, the malware has to move to region 1 which both detectors treat as normal. Note that these decision boundaries represent hyperplanes in an n-dimensional feature space for LR, and complex surfaces in the same space for NN. Therefore, as we increase diversity the target area for evasion gets smaller. Thus, provided that detectors are diverse, making random insertion guesses is unlikely to succeed and expensive to validate. Note that evasion must succeed continuously across consecutive detection windows, which complicates attempts to incrementally evade the detector.

This example also provides intuition on why randomization complicates reverse engineering (as shown by Theorem 1). The attacker has to suffer a significantly increased error ($e_{\vec{p},H}$) if she tries to learn a decision boundary from the same hypothesis classes adopted by the defender. Otherwise, she has to learn a decision boundary of a higher complexity class which requires exponentially larger number of samples.

If the attacker knows precisely the configuration of the base detectors of an RHMD, we verified that it is possible to evade it, for example, by iteratively evading each. This approach incurs a high overhead since instructions need to be injected to evade each of the detectors. We do not consider this case as part of our threat model. Resilience in this case may be achieved if we make the decision boundary of the RHMD non-stationary. This can be accomplished by having a large set of candidate features and periods, of which a random subset is used for the RHMD at any given time. This is an interesting area for future research.

10 Related Work

In this section, we discuss related work organized into two main parts. First, we review related work in malware anomaly detection. In the second part, we discuss adversarial classification and some important recent results in that domain. We show that these results are consistent with our results both in terms of reverse engineering and the use of randomization as a defense.

10.1 Malware Detection

In general, malware detection uses either static (focusing on the structure of a program or system) or dynamic (analyzing the behavior during execution) approaches. Static approaches can be evaded using program obfuscation or simple code transformations [7]. On the other hand, the advantage of dynamic malware detection is that it is resilient to metamorphic and polymorphic malware [7] and can even detect previously unknown malware. However, disadvantages include false positives, and a high monitoring cost during run-time.

A number of works have looked at using low-level architecture features for malware detection, such as frequency of opcodes use in malware [35], evaluation of opcode sequence signatures [4], [36] and opcode sequence similarity graphs [37], which consider offline analysis. Further, Demme et al. [8] proposed collecting performance counter statistics for programs and malware under execution and used them to show that offline detection of malware is effective. Then, a real-time hardware malware detector was built by Ozsoy et al. [13]. Tang et al. [9] used unsupervised learning to detect malware exploits, which will make the regular program deviate from the baseline execution model. Kazdagli et al [11] identified some pitfalls in training and evaluating HMDs for mobile malware, and proposed several improvements to them.

Khasawneh et al. used ensemble learning to improve the accuracy of HMDs [10]. Superficially, ensemble learning is similar

2. This theorem is formed by combining Theorem 2.2 and Corollary 2.3 (with detailed proofs) from [16].

0018-9340 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: George Mason University. Downloaded on April 29,2021 at 07:49:57 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2021.3068873, IEEE Transactions on Computers

to RHMD since it combines the output of multiple diverse detectors to improve the detection performance. However, since ensemble classifiers are deterministic, they can be reverse engineered and evaded. In contrast, the *stochastic switching* between individual detectors in RHMD makes both reverse-engineering and evasion difficult with a difficulty that increases with the number and diversity of the individual detectors. Smutz et al. also studied the use of an ensemble for PDF malware detection [18]; when the baseline detectors disagree, they consider this a possible indicator of evasive malware.

10.2 Adversarial Classification

Several other studies have looked at attacking machine learning models. Attacks can be classified into two types: poisoning and evasion attacks [38]. In poisoning attacks, the adversary focuses on injecting malicious samples in the training data as an attempt to influence the accuracy of the model. For evasion attacks, similar to our own, the adversary crafts input samples that aim to be misclassified by the model. Several evasion attacks were studied in the image classification field. An adversary can make changes to an image's pixels to cause the miss-classification of the image but will not change the image's visibility to the human eye [38]. Since images have high entropy, they can be easily manipulated without changing the appearance of the image. On the other hand, in the malware detection domain, manipulating malware programs has different challenges since the malware functionality needs to be preserved. Evasion attacks in contexts outside image classification have also been considered. Such attacks are called *mimicry* attacks [17]. Although recent studies [16], [39] have provided theoretical grounds for randomization as a possible solution in adversarial classification, practical algorithms have yet to be developed for this problem.

Prior work proposed evasive attacks on PDF malware detectors [40], [41]. These works consider static classifiers using structural features present in the PDF image. In contrast, our contribution targets detectors for a wide range of malware, and we consider run-time anomaly detection using microarchitectural features. Besides the different nature of the application, our work makes a number of contributions relative to these papers, including showing how to reverse engineer the classifiers, reverse-engineering driven instruction injection to evade detection (they use random modifications), exploring the impact of retraining, and providing theoretical insights based on PAC theory into the structure of the problem. Moreover, these studies do not explore resilient classification. Furthermore, Dinakarrao et al. [42] proposed evasion attacks on HMDs by utilizing a wrapper (running in another thread) that runs along with the malware and executes a benign workload to hide the malicious malware activities. This attack works on Hardware Performance Counter (HPC) based HMDs since HPCs are shared between threads. However, HMDs that collect features from the hardware directly (such as the ones we develop in this work) are thread specific; the features collected are not impacted by other running threads and cannot be evaded using this approach. Our developed attack strategies are general (can evade any HMD detector) and do not require the presence of a wrapper. The transferability of attacks between HMDs that are built using different machine learning models was studied in [43]. In contrast, in this paper, we study the transferability of attacks between HMDs that are built using different features.

Similar to the reverse engineering component of our work, Tramèr et al. [44] were able to reverse-engineer machine learning models against production Machine Learning-as-a-service (MLaaS) providers. However, they assumed that they know the features used by the target classifier. In addition, Shokri et al. [45] were able to use reverse-engineered models to perform a membership inference attack (given a data record and black-box access to a model, determine if the record was in the model's training dataset) against MLaaS providers. In both works, they attempt reverse engineering using random noise. We believe that this approach does not work in our threat model since we do not have access to the classifier confidence, and the classification is a continuous process, which makes it difficult to assess incremental changes.

Most relevant to our work, Kuruvila et al. proposed a follow up to our earlier RHMD work [19], which basically evaluated RHMDs that uses a huge number of base detectors. In contrast, in this paper, we show that the detection performance of RHMD can be improved by formulating an optimization problem as a BSG.

11 Concluding Remarks

Recently proposed HMDs have demonstrated remarkable accuracy in classifying malware using low-level features. If HMDs are widely deployed, we must expect that attackers will attempt to evade detection, as is the case in any adversarial setting. This paper considers HMD detection in the presence of evasive malware in detail. In particular, we showed that the attacker can accurately reverse-engineer earlier proposed detectors. Moreover, once the detector is reverse-engineered, we demonstrated simple evasive techniques that can successfully hide malware from detection. We also explored whether having detectors be retrained to capture evasive malware once samples become known (similar to the current practice of updating virus signatures) can be used to harden detectors against evasive malware. For LR, such retraining was not effective. In contrast, NN detectors could easily be retrained to detect the evasive malware. However, after several rounds of an evade-retrain game between the attacker and defender, the NN classifier could no longer be effectively retrained.

In response to these limitations, we proposed resilient HMDs (RHMD) that switch unpredictably between a diversity of detectors. We showed that RHMDs can resist reverse-engineering and complicate evasion. We further improve RHMD detection accuracy and robustness by formulating the problem as a Bayesian Stackelberg Game and solve it as a multi-objective optimization problem to get an optimized switching strategy. In addition, We showed both empirically, and from PAC learnability theory, that RHMDs resilience increases with the number and diversity of the individual detectors available to select from. With this class of resilient HMDs, hardware malware detection becomes a promising direction as a defense against the continued proliferation of malware.

Our threat model assumed that the attacker does not have whitebox access to the RHMD: they do not apriori know the detailed implementation of the RHMD. If they did, we showed that, in theory the attacker can evade detection although at a cost proportional to the number of detectors. Our future work will explore how to defend RHMDs against these whitebox attacks.

12 Acknowledgement

The work in this paper is partially supported by National Science Foundation grants CNS-1422401, CNS-1619322 and CNS-1617915.

0018-9340 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: George Mason University. Downloaded on April 29,2021 at 07:49:57 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2021.3068873, IEEE Transactions on Computers

REFERENCES

[1] L. Davi, A. Dmitrienko, A.-R. Sadeghi, and M. Winandy, "Privilege escalation attacks on android," in *international conference on Information security*. Springer, 2010, pp. 346–360.

[2] S. Abraham and I. Chengalur-Smith, "An overview of social engineering malware: Trends, tactics, and implications," *Technology in Society*, vol. 32, no. 3, pp. 183–196, 2010.

[3] A. Dinaburg, P. Royal, M. Sharif, and W. Lee, "Ether: malware analysis via hardware virtualization extensions," in *Proceedings of the 15th ACM conference on Computer and communications security*, 2008, pp. 51–62.

[4] G. Yan, N. Brown, and D. Kong, "Exploring discriminatory features for automated malware classification," in *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2013, pp. 41–61.

[5] G. Gu, P. A. Porras, V. Yegneswaran, M. W. Fong, and W. Lee, "Bothunter: Detecting malware infection through ids-driven dialog correlation." in *USENIX Security*, vol. 7, 2007, pp. 1–16.

[6] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Computing Surveys (CSUR)*, vol. 44, no. 2, 2012.

[7] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*. IEEE, 2007, pp. 421–430.

[8] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. Stolfo, "On the feasibility of online malware detection with performance counters," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 559–570, 2013.

[9] A. Tang, S. Sethumadhavan, and S. J. Stolfo, "Unsupervised anomaly-based malware detection using hardware features," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2014, pp. 109–129.

[10] K. N. Khasawneh, M. Ozsoy, C. Donovick, N. Abu-Ghazaleh, and D. Ponomarev, "Ensemble learning for low-level hardware-supported malware detection," in *International Symposium on Recent Advances in Intrusion Detection*. Springer, 2015, pp. 3–25.

[11] M. Kazdagli, L. Huang, V. Reddi, and M. Tiwari, "EMMA: A new platform to evaluate hardware-based mobile malware analyses," *CoRR*, vol. abs/1603.03086, 2016. [Online]. Available: http://arxiv.org/abs/1603.03086

[12] "Qualcomm smart protect technology," 2016, accessed July 2016: https://www.qualcomm.com/products/snapdragon/security/smart-protect.

[13] M. Ozsoy, K. N. Khasawneh, C. Donovick, I. Gorelik, N. Abu-Ghazaleh, and D. Ponomarev, "Hardware-based malware detection using low-level architectural features," *IEEE Transactions on Computers*, vol. 65, 2016.

[14] J. Chen and G. Venkataramani, "Cc-hunter: Uncovering covert timing channels on shared processor hardware," in *MICRO*.

[15] H. Drucker, D. Wu, and V. N. Vapnik, "Support vector machines for spam categorization," *IEEE Transactions on Neural networks*, vol. 10, no. 5, pp. 1048–1054, 1999.

[16] Y. Vorobeychik and B. Li, "Optimal randomized classification in adversarial settings," in *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2014, pp. 485–492.

[17] D. Bruschi, L. Cavallaro, and A. Lanzi, "An efficient technique for preventing mimicry and impossible paths execution attacks," in *Performance, Computing, and Communications Conference, 2007. IPCCC 2007. IEEE Internationa*. IEEE, 2007, pp. 418–425.

[18] C. Smutz and A. Stavrou, "When a tree falls: Using diversity in ensemble classifiers to identify evasion in malware detectors," in *NDSS*, 2016.

[19] K. N. Khasawneh, N. Abu-Ghazaleh, D. Ponomarev, and L. Yu, "Rhmd: Evasion-resilient hardware malware detectors," in *MICRO*, 2017, pp. 315–327.

[20] M. Kayaalp, T. Schmitt, J. Nomani, D. Ponomarev, and N. Abu-Ghazaleh, "Scrap: Architecture for signature-based protection from code reuse attacks," in *HPCA*. IEEE, 2013, pp. 258–269.

[24] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: building customized program analysis tools with dynamic instrumentation," in *Acm sigplan notices*, vol. 40, no. 6. ACM, 2005, pp. 190–200.

[21] J. Caballero, C. Grier, C. Kreibich, and V. Paxson, "Measuring pay-per-install: The commoditization of malware distribution." in *Usenix security*, 2011, p. 15.

[22] Malwaredb, "Liste Malware," 2010, available online (last accessed, May 2015): www.malwaredb.malekal.com.

[23] J. L. Henning, "Spec cpu2006 benchmark descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.

[25] M. Kazdagli, V. J. Reddi, and M. Tiwari, "Quantifying and improving the efficiency of hardware-based mobile malware detectors," in *MICRO*. IEEE, 2016, pp. 1–13.

[26] Q. Alasad and J. Yuan, "Logic obfuscation against ic reverse engineering attacks using plgs," in *ICCD*. IEEE, 2017, pp. 341–344.

[27] I. You and K. Yim, "Malware obfuscation techniques: A brief survey." in *Proc. International Conference on Broadband, Wireless Computing, Communications and Applications (BWCCA)*, 2010, pp. 297–300.

[28] A. Osman, "The ao486 project," 2014. [Online]. Available: http://opencores.org/project,ao486

[29] R. Zhuang, S. A. DeLoach, and X. Ou, "Towards a theory of moving target defense," in *Proceedings of the First ACM Workshop on Moving Target Defense*, 2014, pp. 31–40.

[30] S. Sengupta, S. G. Vadlamudi, S. Kambhampati, A. Doupé, Z. Zhao, M. Taguinod, and G.-J. Ahn, "A game theoretic approach to strategy generation for moving target defense in web applications." in *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems*, 2017, pp. 178–186.

[31] S. G. Vadlamudi, S. Sengupta, M. Taguinod, Z. Zhao, A. Doupé, G.-J. Ahn, and S. Kambhampati, "Moving target defense for web applications using bayesian stackelberg games," in *Proceedings of the 2016 international Conference on Autonomous Agents & Multiagent systems*.

[32] P. Paruchuri, J. P. Pearce, J. Marecki, M. Tambe, F. Ordonez, and S. Kraus, "Playing games for security: An efficient exact algorithm for solving bayesian stackelberg games," in *Proceedings of the 2016 international Conference on Autonomous Agents Multiagent systems*.

[33] I. Gurobi Optimization, "Gurobi optimizer reference manual," *URL http://www.gurobi.com*, 2018.

[34] T. M. Mitchell, *Machine Learning*. McGraw-Hill, 1997.

[35] D. Bilar, "Opcodes as predictor for malware," *International journal of electronic security and digital forensics*, vol. 1, no. 2, pp. 156–168, 2007.

[36] I. Santos, F. Brezo, J. Nieves, Y. K. Penya, B. Sanz, C. Laorden, and P. G. Bringas, "Idea: Opcode-sequence-based malware detection," in *Engineering Secure Software and Systems*. Springer, 2010, pp. 35–43.

[37] N. Runwal, R. M. Low, and M. Stamp, "Opcode graph similarity and metamorphic detection," *Journal in computer virology*, vol. 8, no. 1-2, pp. 37–52, 2012.

[38] N. Pitropakis, E. Panaousis, T. Giannetsos, E. Anastasiadis, and G. Loukas, "A taxonomy and survey of attacks against machine learning," *Computer Science Review*, vol. 34, p. 100199, 2019.

[39] B. Biggio, G. Fumera, and F. Roli, "Adversarial pattern classification using multiple classifiers and randomisation," *Structural, Syntactic, and Statistical Pattern Recognition*, pp. 500–509, 2008.

[40] W. Xu, Y. Qi, and D. Evans, "Automatically evading classifiers," in *Proceedings of the 2016 network and distributed systems symposium*, vol. 10, 2016.

[41] P. Laskov *et al.*, "Practical evasion of a learning-based classifier: A case study," in *S&P*. IEEE, 2014, pp. 197–211.

[42] S. M. P. Dinakarrao, S. Amberkar, S. Bhat, A. Dhavlle, H. Sayadi, A. Sasan, H. Homayoun, and S. Rafatirad, "Adversarial attack on microarchitectural events based malware detectors," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[43] K. N. Khasawneh, N. B. Abu-Ghazaleh, D. Ponomarev, and L. Yu, "Adversarial evasion-resilient hardware malware detectors," in *ICCAD*. IEEE, 2018, pp. 1–6.

[44] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis," in *USENIX Security*, 2016, pp. 601–618.

[45] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *S&P*. IEEE, 2017, pp. 3–18.

0018-9340 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: George Mason University. Downloaded on April 29,2021 at 07:49:57 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2021.3068873, IEEE Transactions on Computers

Md Shohidul Islam is a Ph.D. student in the ECE Department at George Mason University. He received his M.S. in Computer Engineering from the University of Ulsan. His research interests are in the areas of systems security and adversarial machine learning.

Khaled N. Khasawneh is an Assistant Professor in the ECE Department at George Mason University. His research interests are in architecture support for security and adversarial machine learning. He received his Ph.D. in Computer Science from the University of California at Riverside in 2019.

Nael Abu-Ghazaleh is a Professor in the Computer Science and Engineering department and the Electrical and Computer Engineering department at the University of California at Riverside. His research interests are in the areas of secure system design, parallel discrete event simulation, networking and mobile computing. He received his Ph.D. from the University of Cincinnati in 1997.

Dmitry Ponomarev is a Professor in the Department of Computer Science at SUNY Binghamton. His research interests are in the areas of computer architecture, secure and power-aware systems and high performance computing. He received his Ph.D. from SUNY Binghamton in 2003.

Lei Yu is a Professor in the Department of Computer Science at SUNY Binghamton. His research interests are in the areas of machine learning and data mining. He received his Ph.D. from Arizona State University in 2005.

0018-9340 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: George Mason University. Downloaded on April 29,2021 at 07:49:57 UTC from IEEE Xplore. Restrictions apply.