PREEMPT: PReempting Malware by Examining Embedded Processor Traces

Kanad Basu
Electrical and Computer Engineering, New York University

Rana Elnaggar
Electrical and Computer Engineering, Duke University

Krishnendu Chakrabarty
Electrical and Computer Engineering, Duke University

Ramesh Karri
Electrical and Computer Engineering, New York University

ABSTRACT

Anti-virus software (AVS) tools are used to detect Malware in a system. However, software-based AVS are vulnerable to attacks. A malicious entity can exploit these vulnerabilities to subvert the AVS. Recently, hardware components such as Hardware Performance Counters (HPC) have been used for Malware detection. In this paper, we propose PREEMPT, a zero overhead, high-accuracy and low-latency technique to detect Malware by re-purposing the embedded trace buffer (ETB), a debug hardware component available in most modern processors. The ETB is used for post-silicon validation and debug and allows us to control and monitor the internal activities of a chip, beyond what is provided by the Input/Output pins. PREEMPT combines these hardware-level observations with machine learning-based classifiers to preempt Malware before it can cause damage. There are many benefits of re-using the ETB for Malware detection. It is difficult to hack into hardware compared to software, and hence, PREEMPT is more robust against attacks than AVS. PREEMPT does not incur performance penalties. Finally, PREEMPT has a high True Positive value of 94% and maintains a low False Positive value of 2%.

1 INTRODUCTION

Malware has proliferated across a variety of computing platforms e.g., PCs, servers, and smartphones. Malware can be of diverse types, including Trojans, Viruses, Worms and Rootkits, and they can be classified based on their function [1]. Generally, anti-virus software (AVS) are used to detect Malware. However, AVS have some shortcomings [2]. For example, new Malware can subvert AVS by abusing software vulnerabilities [3]. This starts a cat-and-mouse between the Malware and the AVS.

Recently, researchers have started co-opting trusted hardware components to detect Malware. A hardware-based security system cannot be thwarted by software, whether running in an user or a hypervisor mode. Design houses, e.g., Qualcomm, are deploying hardware-based security systems in mobile devices [4]. Malware detection using Hardware Performance Counters (HPC) have been proposed in [2, 4, 5]. However, HPCs are not suited for real-time monitoring and they incur performance penalty.

DAC '19, June 2–6, 2019, Las Vegas, NV, USA
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3317883

We show that embedded trace buffers (ETBs), available in modern processors, can be repurposed for Malware detection and classification. An ETB is an on-chip circular memory buffer, used for post-silicon validation. Post-silicon validation detects functional errors after the chip is manufactured. Scan-based manufacturing test detects manufacture time structural failures, and hence are not at-speed. ETBs operate at the functional clock speed and monitor select internal signals in real-time. ETBs collect signal values every cycle and dump them to an offline debugger through the JTAG debug interface [6–8].

We propose PREEMPT, a real-time, low-latency Malware detection procedure with zero hardware overhead. PREEMPT analyses ETB traces to detect Malware and makes three key contributions:

(1) Develops an ETB-based Malware detector that reuses the signals used for post-silicon debug on an OpenSPARC T1 processor.
(2) Analyzes ETB traces in real-time using Machine learning classifiers.
(3) Detects real Malware (botnets of Gafgyt and Mirai families).

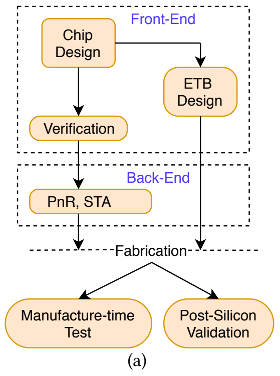
The paper is organized as follows. Section 2 provides background on Malware detection and ETB. Section 3 describes PREEMPT, the proposed ETB-based Malware detection procedure. Section 4 reports the results. Finally, Section 5 concludes the paper.

2 BACKGROUND AND RELATED WORK

2.1 Malware Detection: Prior Work

Malware are pernicious programs, which spread malevolence ranging from denial-of-service to security breaches to invasion of privacy. AVS have been developed to detect different kinds of Malware [9, 10]. The problems with these approaches are threefold. First, AVS have software vulnerabilities. According to a recent study, however robust an AVS is, a stealthier Malware can always be created to circumvent it [11]. Moreover, most AVS use static signatures to detect Malware [2]. A Malware can have multiple executable formats to circumvent these signatures. Finally, since AVS is a software, it is slow, which results in high latency for Malware detection.

Researchers have started using on-chip hardware components like HPCs to detect Malware. The underlying assumption is that although an AVS can be circumvented by variations in Malware code, it is difficult to subvert a hardware-based detector, since the Malware function will remain the same. Over the years, researchers have proposed various HPC-based Malware detection schemes [2, 4, 5]. HPC-based Malware detection was analyzed in [12]. HPC-based Malware detection has an unacceptably high false positive rate (FPR) of 15% [2]. HPCs measured inside a VM differ from those that are measured on the hardware [12]. Moreover, even HPC-based



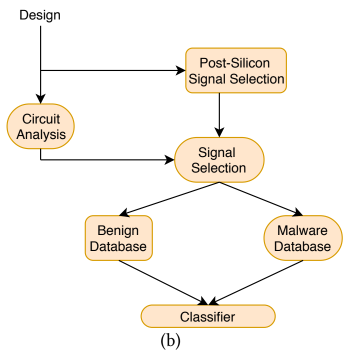
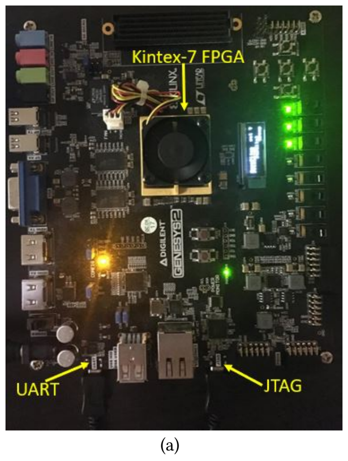


Figure 1: (a) Post-silicon validation as part of the IC design flow. (b) Overview of PREEMPT.



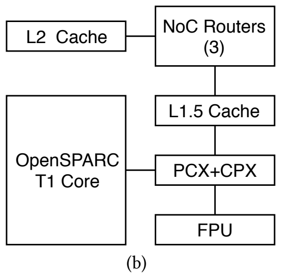


Figure 2: Experiment Platform: (a) FPGA platform that implements the OpenPiton/OpenSparc design. (b) Architecture of an OpenPiton tile.

Malware detection is not real-time; the system is interrupted to collect HPC readings, yielding about ~5% performance penalty. There is a trade-off between detection latency and performance penalty in HPC-based schemes.

2.2 Post-Silicon Validation

The increase in circuit complexity, a reduction in the transistor feature size and the time-to-market window result in design bugs that escape pre-silicon verification. Post-silicon validation is used to detect these escaped bugs. The primary challenge in post-silicon validation is the limited observability. Scan chains are not suitable, since functional errors can only be debugged in real-time with the chip running at the functional clock frequency [8].

In order to improve observability in a manufactured chip, design-for-debug techniques such as ETBs have been developed [8]. The overview of trace buffer-based post-silicon debug methodology is shown in Figure 1(a). Designing an ETB involves identifying and exposing useful internal signal states for debugging. To keep the size of the ETB small, the signals to be traced for post-silicon validation are selected during the design phase [6]. Most signal-selection algorithms use restoration of untraced signal states during debug as a measure [6–8].

Since the ETB is an example of dedicated debug hardware, researchers have been attempting to re-purpose it for other functionalities when the debug mode is not on. For example, the ETB can be used as a victim cache [13].

The research question that this study answers is the following: Can we re-purpose the ETB and the associated debug infrastructure for security monitoring? Three motivations behind re-using ETBs to detect Malware are: **(1) ETBs have the benefits of hardware-based Malware detection as described in Section 2.1. (2) ETBs do not incur performance penalty on other applications. (3) ETBs support low-latency Malware detection.**

3 PREEMPT: ETB-BASED MALWARE DETECTION

Figure 1(b) outlines the ETB-based Malware detection methodology, PREEMPT. The first step selects the trace signals. The second step analyzes the circuit to remove redundant signals, like resets and scan enables. We use SigSET signal-selection tool that provides good restoration performance[1] [14]. A set of signals with a high restoration ratio indicates a high-connectivity among all signals in the circuit. This translates into a high probability of detecting a change in state of any signal in the circuit (traced or untraced). Our hypothesis is these signals can distinguish between Malware and benign programs. Next, we create a database of traces for the benign programs and Malware. These traces are used as inputs to an ML classifier.

3.1 Experiment Platform

We run malicious and benign applications on the OpenPiton platform [15], shown in Figure 2(a), which comprises of an OpenSparc T1 core[2] [16], implemented on a Xilinx Kintex-7 FPGA integrated on a Digilent Genesys2 board. We connect the FPGA to a host machine via Universal asynchronous receiver-transmitter (UART). We use peripherals on the host to control the OpenSparc T1 cores.

Openpiton is a tiled manycore architecture, with each chip comprising of multiple OpenSparc cores. Intra-chip communication between tiles is via a high-speed Network-on-chip (NoC). The tiles (cores+caches+FPU+etc.) are arranged in a 2-D mesh. Inter-chip communication is possible via the chip bridge interface. The chip bridge connects the chip to an off-chip logic known as chipset, that comprises of I/O, traffic splitter, DRAM and network routers. The structure of an OpenPiton tile is shown in Figure 2(b). It comprises of a OpenSparc T1 core (modified, e.g., without the cryptographic unit and with a reduced number of threads from 4 to 2), two levels of cache – L1.5 and L2, three NoC routers, Floating point unit (FPU) and the cache core crossbar (CCX). The CCX has two parts:

(1) **CPX:** The cache processor crossbar to transfer messages from the cache to the core.
(2) **PCX:** The processor cache crossbar to transfer messages from the core to the cache.

The caches connect directly to the 3 NoC routers. The core is connected to the caches, the FPU as well as the routers (for I/O operations) using the CCX interface. The original OpenSPARC T1

[1]Any trace-signal-selection tool, providing a high restoration, can be used.
[2]The OpenSparc T1 core is a industry-hardened multi-threaded design that implements a stable instruction set architecture and comes with compiler and OS support.

design had 2 caches - L1 and L2. The L1 is a private cache for each core, while the L2 cache is distributed across all tiles in a chip. Both L1 data and instruction caches are 4-way set associative. However, the data cache is of size 8KB with line size 16-bytes, while the instruction cache is of size 16KB with line size 32-bytes. The L2 cache is also 4-way set associative with a size of 64KB per tile.

3.2 Malware Families

In this study we consider two Malware families [17]:

(1) **Gafgyt** or Bashlite family of Linux Malware are used to launch Distributed denial-of-service (DDoS) attacks. In 2016, one million IoT devices were affected by this Malware [18]. Since Gafgyt is written in C, it can be cross-compiled and applied across processor architectures.
(2) **Mirai** is a Linux Malware family that transforms a remotely controlled equipment into a "bot", and uses it to launch large scale network attacks. Mirai is a sophisticated upgrade of Gafgyt, that includes a built-in functionality to scan for vulnerable devices. While the command and control server IP is hardcoded in Gafgyt, Mirai resolves the address using DNS.

3.3 Trace Database

We collect Malware traces by running 100 Linux/SPARC Malware samples downloaded from VirusTotal [17]. A total of 300 more Malware samples, also collected from VirusTotal, are used for cross-validation (Section 4.3). Our benign programs are SPEC benchmarks compiled to run on SPARC architectures and system binaries, e.g., *ls*, *mkdir*, *chmod*, *ping*, *netstat*[3]. The traces obtained by running these programs on the OpenPiton platform are inputs to the classifiers, as explained in Section 3.6.

3.4 Classifiers

We use four classifiers from the Python scikit-learn library: KNeighborsClassifier for K-nearest neighbor (KNN), RandomForestClassifier for Random forest (RF), DecisionTreeClassifier for Decision tree (DT), and MLPClassifier for Neural Networks (NN). We choose these four classifiers, since they provide satisfactory performance in existing hardware-based Malware detection schemes [2, 4]. For each classifier, the ratio of the size of the training set to the size of the test data set is 4:1. We report True Positive (TP), False Positive (FP), and Precision for each classifier.

3.5 Trace Signal Selection

The trace signals selected by SigSET offer high restoration, since they are connected to large number of untraced signals in the circuit. These ETB signals are used for debug and hence, re-using them for detecting Malware will not incur any overhead. However, if the designer has the bandwidth to trace more signals exclusively for Malware detection, it would be beneficial. In this study we only use the signals selected for debug. We applied SigSET on the top-level design netlist to obtain a list of 128 trace signals.

SigSET selects the lower 128 bits of the *cpx_spc_data_cx*3. The *CPX_SPC* is a 145-bit data packet. Bits 128-144 provide various information regarding the CPX packet, e.g., valid CPX packet, transaction type, error in the transaction. The lower 128 bits are reserved for the CPX data that is transferred to the CPU.

Table 1 delineates the lower 128 bits of the CPX bus, for each CPX bus transaction type. Load transactions (*LOAD_RET*) transfer data from an L2 cache or I/O to a core. Since the data cache has a

Bits	LOAD_RET byte cacheline offset	IFILL_RET Instrn. cacheline	EVICT_REQ	ST_ACK	INT_REQ Interrupt
0-7	15	{0, 16}	[2-5] Invalidate way	[2-5] Store/ ACK way	Yes Yes
8-15	14	{0, 16}	-	-	Yes
16-23	13	{0, 16}	-	-	Yes
24-31	12	{0, 16}	-	-	Yes
32-39	11	{4, 20}	-	-	Yes
40-47	10	{4, 20}	-	-	Yes
48-55	9	{4, 20}	-	-	Yes
56-63	8	{4, 20}	-	-	Yes
64-71	7	{8, 24}	-	-	-
72-79	6	{8, 24}	-	-	-
80-87	5	{8, 24}	-	-	-
88-95	4	{8, 24}	-	-	-
96-103	3	{12, 28}	-	-	-
104-111	2	{12, 28}	-	-	-
112-119	1	{12, 28}	Invalidate vector	Store/ACK vector	-
120-127	0	{12, 28}	Invalidate vector	Store/ACK vector	-

Table 1: Details of the lower 128 bits of *cpx_spc_data_cx*3. The table explains the bits for the Load, Instruction fill, Invalidation, Store acknowledgment, and Interrupt type transactions. For Load, the table shows the byte at each cache-line offset corresponding to a set of bit positions. For Instruction fill, the table shows the 32-bit instructions at each cache-line, corresponding to a set of bit positions. For Invalidate and Store acknowledgements, the table shows the bit positions that create the corresponding vectors and choose the associativity. For atomic instructions and interrupts, the table shows the relevant bit positions.

16-byte line size, the 128-bits represent the cache-line that is being filled in the D-cache. The 128-bits can be divided into 16 fragments, each 8-bit long corresponding to the cache-line offset, as shown in Table 1. For example, bits 0-7 correspond to the byte that fills the cache-line at offset 15, bits 8-15 correspond to the byte that fills the cache-line at offset 14, etc. The big-endian format of the Load data dictates this ordering.

In case of Instruction fills (*IFILL_RET*), four 32-bit instructions are transferred as shown in Table 1. For an Instruction-fill, 32-bytes of data are returned in contrast to 16-bytes of data for LOAD instructions. Since the data field of CPX packet is 128 bits, two consecutive data packets are sent. Hence, two cache-lines are mentioned for each bit-position in Table 1. Four entries in Table 1 are needed for information about a single instruction.

For Invalidation and Store Acknowledgment transactions, represented by (*EVICT_REQ*) and (*ST_ACK*) in Table 1, bits 112 to 127 are used to generate a vector, while bits 2 to 5 are used to determine associativity (the caches are 4-way set associative). Atomic instructions (*ATOMIC_RES*) take all the 128-bits, while interrupts (*INT_REQ*) require only the lower 64 bits.

Analyzing the benign program and Malware traces, we found that 98% of CPX transactions correspond to either a Load return or an Instruction fill. We divided the 128-bit trace into four parts of 32-bits each. For Instruction fill, each part corresponds to one of the four instructions being transferred by a 128-bit packet. For Load operations, each part corresponds to four consecutive bytes or one 4-byte word of the cache-line that fills the D-cache, i.e., four entries in Table 1. This aids to analyze the change in classification accuracy with trace buffer width, performed in Section 4.2. We create feature vectors out of these 128-bits for classification, with each vector comprising of four features. The classifiers distinguish

[3]ping and netstat are important since our Malware are botnets.

Malware traces from the benign by analyzing the cache-lines and the instructions that are filled in the D- and the I- caches.

3.6 Pre-process Traces to Create Trace Vectors

Once the trace data are collected, they need to be pre-processed before being used as inputs to the ML classifier. Our trace pre-processing procedure consists of two stages.

3.6.1 Cleanup of reset values in Traces. Malware and benign programs sometimes initialize all registers to their reset states, i.e., the reset signals are set to 1 and the remaining signals are reset to 0. The associated traces are not useful; a reset value of a register offers no clue regarding the program and hence, these traces should be removed before classification.

3.6.2 Cleanup of Benign Traces. Parts of the Malware include benign functions [2]. Hence, these functions will have traces similar to benign programs. These traces are labeled twice – one with label 0 in the benign applications and another with label 1 for Malware. This creates a multi-class label for the same data and undermines classification performance. We label them once, as benign, during training. These traces are purged from the Malware database. Once the ETB traces are cleaned up, we convert them into feature vectors for training and testing. We created four feature vectors of 32-bit each for both benign programs and Malware.

4 EXPERIMENTAL RESULTS

4.1 Classification of Benign vs Malware

An ML-based classifier is trained on data from two classes – benign and Malware. Once trained, it predicts the probability that the new data belongs to a particular class. The trace data (Malware+benign) is mixed and split into training and test sets. The results are reported in Figure 3(a). All the classifiers yield high TP and Precision, which shows that they can detect most of the malicious traces accurately. For KNN, RF and DT classifiers the FP is $< 1.5\%$. This is $10\times$ lower than the best FP of 15% obtained by HPC-based Malware detectors [2, 12]. For the Neural Network classifier, the FP is 6.7%, which is $2.2\times$ less than the HPC-based schemes.

4.2 ETB Width vs Classification Accuracy

Let us study how classification accuracy changes with ETB width. Until now, ETB monitored 128 signals. We observe the classification accuracy when the ETB width is reduced to 64 and 32 signals. This helps us understand how the performance of PREEMPT is impacted when design constraints limit the ETB width. The TP and FP variations for all 4 classifiers are shown in Figure 4(a) and Figure 4(b), respectively. As ETB width increases ($32 \rightarrow 64 \rightarrow 128$), TP increases and FP decreases. This is because as more signals are traced every cycle, the ETB gets a better view of the internal events in the circuit and in turn a superior ability to distinguish benign programs from Malware. However, even when ETB monitors only 32 signals, 3-of-the-4 classifiers (KNN, RF and DT) have a TP $> 80\%$.

4.3 Cross Validation

We cross-validate our model using traces from unknown Malware samples. For cross-validation, we downloaded 300 additional Malware samples from VirusTotal [17], and arranged them into 3 sets of 100 each. Figure 3(b) reports cross-validation accuracy by measuring the TP value. The cross-validation accuracy of the model is

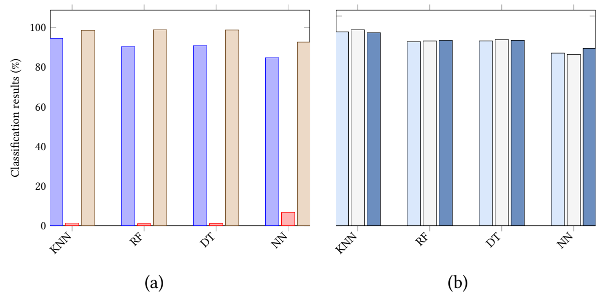


Figure 3: (a) Classification performance for the four classifiers. TP, FP and Precision are shown as blue, red and yellow bars respectively. For the KNN, RF and DT classifiers, TP > 90% and FP < 1.5%. KNN has the highest TP (94.6%) and RF has the best FP (1%). (b) Cross-Validation accuracy. KNN has the best TP followed by RF and DT. The three colored bars represent each cross-validation set.

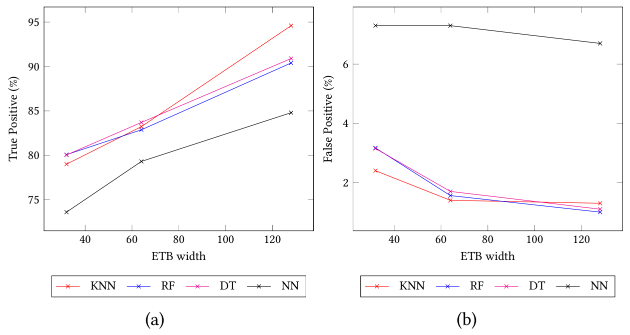


Figure 4: Variation of (a) TPR (b) FPR with ETB width.

high (up to 93%) confirming that it can handle unknown Malware samples. Among the classifiers, KNN has the best classification performance, followed by RF and DT.

4.4 Malware Family Classification using ETB

We applied our ML-based classifiers to distinguish between Malware of Gafgyt and Mirai families. This is essential to understand whether ETB traces can distinguish one type of Malware from another. In this case, the benign software traces are not used for training the classifiers. Gafgyt Malware is labeled 0 and Mirai is labeled as 1 during training. The results are shown in Figure 5(a). We report the percentage of correctly classified Malware of each family. All the classifiers perform well in identifying Gafgyt traces. The average accuracy is 90%. However, the classifiers don't perform well when identifying Malware samples of Mirai family. This is especially true for NN-classifier, when the accuracy is only 46%. This is because, as explained in Section 3.2 Mirai is an upgrade over Gafgyt [19] and hence, there are a number of identical instructions between Malware of these two families.

4.5 Malware Cross-family Validation

In this section, we examine if a classifier is trained using Malware from one family, can it detect Malware from other families correctly? The experiments in Section 3.2 are different from those in this section. In Section 3.2, the classifiers were expected to predict a malicious trace as belonging to Gafgyt or Mirai family. On the

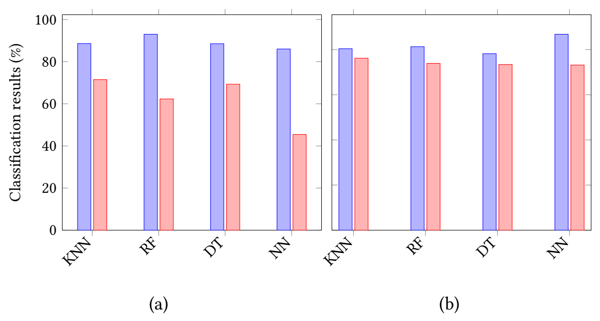


Figure 5: (a) Classification of Malware families. (b) Classification accuracy when two different families of Malware are used for training and test. The blue and red bars are Gafgyt and Mirai family-detection accuracy, respectively.

other hand, in this experiment the classifier has to predict the malicious trace as Malware, and not benign; with the constraint that the classifier is not trained with traces from the same family of Malware that it is being tested.

The training database for the first experiment consists of two sets of traces – benign programs (labeled 0) and Gafgyt Malware (labeled 1). After training, traces from Mirai family are used to test the classifier performance. Ideally, the classifier should correctly identify all Mirai traces as Malware. The results are shown in Figure 5(b) as blue bars. For all classifiers, we get high classification accuracy (> 78%). NN-based classifier provides the highest accuracy. In the second experiment, benign traces (labeled 0) and Mirai traces (labeled 1) are used for training the classifier and Gafgyt traces are used for testing. The results are presented in Figure 5(b) as red bars. The accuracy in this case is lower than Figure 5(b), the average being 73%. The number of Mirai samples we obtained are less than Gafgyt traces, and hence, the classifier is biased toward benign data.

4.6 How Quickly Can Malware be Spotted?

We examine how many clock cycles does it take for PREEMPT to detect a malicious trace. The time required by HPC-based Malware detectors depend on how frequently the HPC readings are taken. Increasing the frequency of HPC readings improves Malware detection latency. However, it incurs a performance penalty, since the system needs to be halted to read out the HPCs.

In order to estimate the detection latency of PREEMPT, we randomly choose 15 Malware samples from the Malware pool. We run each Malware sample and use the trace obtained every cycle as an input to the classifier. The classifier labels the trace as being either benign or malicious. The first cycle when a classifier tags a malicious trace as Malware is reported in Table 2. In order to maintain timing, we do not pre-process the traces as discussed in Section 3.6, so that even the reset and benign traces are retained. KNN classifiers detect malicious traces with the lowest latency, while NN classifiers have the maximum detection latency.

The malicious traces are mostly detected in < 100 cycles. or the KNN-based classifier, the maximum number of cycles to obtain a malicious trace is 463 cycles for sample 15. The maximum number of cycles for DT-based classifier is 528 cycles for sample 4. The malware detection latency depends on how fast the traces are transferred to the ML-based classifier and the speed of the classifier.

Malware	Malicious trace (cycles)			
Sample	KNN	RF	DT	NN
1	118	118	118	118
2	50	50	50	50
3	42	42	42	42
4	211	528	528	528
5	266	266	266	266
6	138	138	138	248
7	0	0	0	0
8	98	98	98	98
9	4	4	4	4
10	56	56	56	56
11	272	272	272	439
12	97	97	97	97
13	93	94	94	93
14	29	29	29	29
15	463	463	463	463

Table 2: Number of cycles required to detect a malicious trace. For each classifier the worst-case detection latency is colored in red. For most cases, malicious traces are detected within 100 cycles from their activation. For all samples, they are detected within 600 cycles.

4.7 Mix of Malware and Benign Traces

In a real world scenario, Malware do not run in isolation. They run in the background along with a benign software. The challenge in this case is to determine whether PREEMPT can identify malicious traces in presence of a benign program. This experiment was performed for the HPC-based Malware detection scheme by [5] and [4]. Both used a Virtual machine (VM)-based system to run benign applications and Malware together and measure the HPC values. However, as reported by [12], HPC values in a VM environment differ significantly from HPC values on actual hardware; hence, the conclusions of [5] and [4] are debatable. On the other hand, we perform experiments on the Genesys-2 FPGA platform.

We ran SPEC benchmark “hmmer" in the background and 15 Malware samples from Virustotal in the foreground. We did not pre-process the traces using the techniques from Section 3.6 so as to retain the timing information of each trace. We performed timing analysis (Section 4.6) using the KNN-classifier trained in Section 4.1 to detect Malware from these traces. PREEMPT was able to obtain a malicious trace within 112 cycles. Therefore, even if a Malware tries to hide within benign operations, PREEMPT can detect it.

4.8 Combination of Classifiers

We study the implications of combining classifiers by voting on their predictions. This determines how the classification performance improves if one can use more than one classifier. We notice how many of the traces are classified as malware by combining all four classifiers, how many by combining 3-out-of-4, how many by 2-out-of-4 classifiers and how many by at least one classifier. When we consider 3-out-of-4, a trace is a malware if it is classified as such by {KNN, RF, DT} or by {KNN, RF, NN} or by {KNN, DT, NN} or by {DT, RF, NN}. The results are reported in Figure 6.

4.9 Analysis of the Results

We visualize the high-dimensional traces by mapping the traces into two-dimensional embedding traces using t-distributed stochastic neighbor embedding (t-sne) [20]. This identifies patterns in the data that contribute to the high classification accuracy in our experiments. We use the implementation of t-SNE in Python from

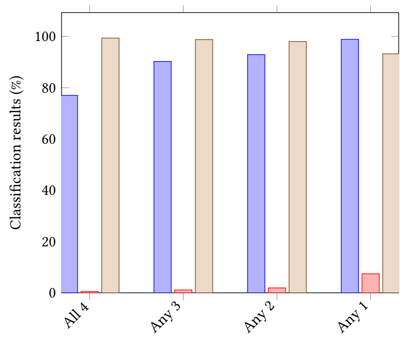


Figure 6: Classification performance of classifier combinations. TP, FP and Precision are represented by blue, red and yellow bars. While TP increases with reduction in number of classifiers needed for a majority vote, FP worsens. Precision is highest when all four classifiers are considered.

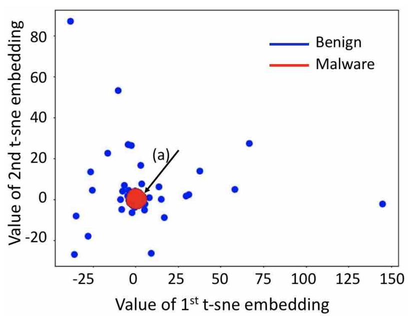


Figure 7: Scatter plots of the first and second t-sne embeddings of benign and Malware traces. The embeddings of the Malware traces form a dense cluster that can be distinguished from the benign traces using non-linear classifiers. (a) Malware t-sne embeddings form a dense cluster.

the scikit learn library. We use matplotlib library in Python to plot scatter plots of the obtained t-sne embeddings[4].

Figure 7 shows the scatter plots of the density embeddings after scaler transformation. It shows that Malware t-sne embeddings form a well-defined cluster while the t-sne embeddings that belong to benign traces span the whole feature space. This distinct cluster observable in the scatter plot implies that the benign and the Malware traces have different Cache Processor Crossbar (CPX) access patterns that can be separated using non-linear classifiers. Some data points of the t-sne embeddings of the benign traces overlap with data points of t-sne embeddings of the Malware. We conclude that these data points account for benign traces that are incorrectly labeled as malicious (false positive points) in our results.

5 CONCLUSION

PREEMPT is a debug hardware trace-based Malware detection technique. PREEMPT re-uses debug trace data and hence, incurs zero overhead. The ML-based classifiers provide high TP value ($> 94\%$), while maintaining a low FP value ($< 2\%$). PREEMPT was validated on Linux-Malware samples from VirusTotal. PREEMPT detects Malware with extremely low latency. PREEMPT can discriminate between two close Malware families.

In summary, PREEMPT offers the following important advantages: Zero silicon area overhead. Zero performance penalty. High Malware classification accuracy. Low Malware detection latency. Difficult to hack as it is hardware-based. This approach can be extended to detect rootkits, backdoors, and ransomware. PREEMPT can be adapted to Malware in Windows, Android and MacOS. Another study can assess the effect of alternate trace signal selection algorithms. For some devices, real-time observability is not available. An aggregate of traces need to be dumped periodically and analyzed. One can examine how this impacts classification accuracy.

6 ACKNOWLEDGEMENTS

This work supported in part by NYU CCS, NYU-AD CCS, National Science Foundation (A#: 1526405) and (A#: 1513130).

REFERENCES

[1] G. McGraw and G. Morrisett, "Attacking malicious code: A report to the infosec research council," *Proceedings of IEEE software*, vol. 17, no. 5, pp. 33–41, 2000.
[2] 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 Comp. Arch. News*, vol. 41, no. 3, pp. 559–570, 2013.
[3] B. Min, V. Varadharajan, U. Tupakula, and M. Hitchens, "Antivirus security: naked during updates," *Proceedings of Software: Practice and Experience*, vol. 44, no. 10, pp. 1201–1222, 2014.
[4] M. Kazdagli, V. J. Reddi, and M. Tiwari, "Quantifying and improving the efficiency of hardware-based mobile malware detectors," in *IEEE/ACM International Symposium on Microarchitecture*, p. 37, 2016.
[5] X. Wang and R. Karri, "Numchecker: Detecting kernel control-flow modifying rootkits by using hardware performance counters," in *ACM/IEEE Design Automation Conference*, pp. 1–7, 2013.
[6] H. F. Ko and N. Nicolici, "Algorithms for state restoration and trace-signal selection for data acquisition in silicon debug," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 2, pp. 285–297, 2009.
[7] X. Liu and Q. Xu, "Trace signal selection for visibility enhancement in post-silicon validation," in *Design, Automation & Test in Europe Conference & Exhibition*, pp. 1338–1343, 2009.
[8] K. Basu and P. Mishra, "Rats: Restoration-aware trace signal selection for post-silicon validation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 4, pp. 605–613, 2013.
[9] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket.," in *Network and Distributed Security Symposium*, vol. 14, pp. 23–26, 2014.
[10] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: behavior-based malware detection system for android," in *ACM workshop on Security and privacy in smartphones and mobile devices*, pp. 15–26, ACM, 2011.
[11] S. Jana and V. Shmatikov, "Abusing file processing in malware detectors for fun and profit," in *IEEE Symposium on Security and Privacy*, pp. 80–94, 2012.
[12] B. Zhou, A. Gupta, R. Jahanshahi, M. Egele, and A. Joshi, "Hardware performance counters can detect malware: Myth or fact?," in *ACM Asia Conference on Computer and Communications Security*, pp. 457–468, 2018.
[13] N. Jindal, P. R. Panda, and S. R. Sarangi, "Reusing trace buffers to enhance cache performance," in *IEEE Design, Automation & Test in Europe Conference & Exhibition*, pp. 572–577, 2017.
[14] K. Rahmani, P. Mishra, and S. Ray, "Efficient trace signal selection using augmentation and ilp techniques," in *IEEE International Symposium on Quality Electronic Design*, pp. 148–155, 2014.
[15] J. Balkind, M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, A. Lavrov, M. Shahrad, A. Fuchs, S. Payne, X. Liang, *et al.*, "Openpiton: An open source manycore research framework," *ACM SIGARCH Comp. Arch. News*, vol. 44, no. 2, pp. 217–232, 2016.
[16] I. Parulkar, A. Wood, J. C. Hoe, B. Falsafi, S. V. Adve, J. Torrellas, and S. Mitra, "Opensparc: An open platform for hardware reliability experimentation," in *Workshop on Silicon Errors in Logic-System Effects*, pp. 1–6, 2008.
[17] V. Total, "Virustotal-free online virus, malware and url scanner," *Online: https://www. virustotal. com/en*, 2012.
[18] T. Spring, K. Carpenter, and M. Mimoso, "Bashlite family of malware infects 1 million iot devices," *Threat Post*, 2016.
[19] A. Marzano, D. Alexander, O. Fonseca, E. Fazzion, C. Hoepers, K. Steding-Jessen, M. H. P. Chaves, Ítalo Cunha, D. Guedes, and W. M. Jr., "The Evolution of Bashlite and Mirai IoT Botnets," in *IEEE International Symposium on Computers and Communication*, pp. 813–818, 2018.
[20] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[4]The t-sne embedding analysis has a quadratic time and space complexity in the analyzed data points. This is not time-efficient and hence is a one-time experiment and not part of the classifier.