Hardware-Assisted Malware Detection using Machine Learning

Zhixin Pan, Jennifer Sheldon, Chamika Sudusinghe, Subodha Charles and Prabhat Mishra
Department of Computer and Information Science and Engineering
University of Florida, Gainesville, Florida, USA

***Abstract*— Malicious software, popularly known as malware, is a serious threat to modern computing systems. A comprehensive cybercrime study by Ponemon Institute highlights that malware is the most expensive attack for organizations, with an average revenue loss of $2.6 million per organization in 2018 (11% increase compared to 2017). Recent high-profile malware attacks coupled with serious economic implications have dramatically changed our perception of threat from malware. Software-based solutions, such as anti-virus programs, are not effective since they rely on matching patterns (signatures) that can be easily fooled by carefully crafted malware with obfuscation or other deviation capabilities. Moreover, software-based solutions are not fast enough for real-time malware detection in safety-critical systems. In this paper, we investigate promising approaches for hardware-assisted malware detection using machine learning. Specifically, we explore how machine learning can be effective for malware detection utilizing hardware performance counters, embedded trace buffer as well as on-chip network traffic analysis.**

***Index Terms*—Malware detection, machine learning, hardware performance counters, trace buffer, on-chip traffic.**

I. Introduction

We are living in the era of Internet-of-Things (IoT), an age in which the volume of connected smart computing devices exceeds the human population. Malware (malicious software) is a critical threat today because of our increasing reliance on such computing devices. There is a wide assortment of malware including Trojans, Viruses, Worms, Adware, Spyware, Crimeware and Rootkits. They are meticulously designed to damage a computer, server, or computer network and lead to severe impairment to the target system [1]. For example, drive-by-download malware seizes control of a computer system and compromises the security and privacy of stored data [2]. Similarly, nation-state hackers exploit malware for intelligence and to attack critical infrastructures [3], [4]. Malware is also exploited to control IoT devices as botnets and launch large scale attacks through the Internet [5]. The portability of malware also empowers it to proliferate across diverse platforms at an alarming rate. With the rapid development of Internet and smart devices in recent years, malware-implanted applications are increasingly exposing systems to a wide variety of attacks.

Figure 1 shows the results of a recent cybercrime study involving 355 companies across 11 countries covering 16 industrial sectors. Clearly, malware is the most expensive threat for organizations, with an average revenue loss of $2.6 million per organization in 2018 [6]. Moreover, the cost is increasing over the years, with 11% increase compared to 2017. There is a critical need to develop effective solutions for malware detection to enable a truly secure cyberspace.

	Business disruption	Information loss	Revenue loss	Equipment damage	Total cost by attack type
Malware (+11%)	$ 0.5	$ 1.4	$ 0.6	$ 0.1	$ 2.6
Web-based attacks (+17%)	$ 0.3	$ 1.4	$ 0.6	$ –	$ 2.3
Denial-of-service (+10%)	$ 1.1	$ 0.2	$ 0.4	$ 0.1	$ 1.7
Malicious insiders (+15%)	$ 0.6	$ 0.6	$ 0.3	$ 0.1	$ 1.6
Phishing and social engineering (+8%)	$ 0.4	$ 0.7	$ 0.3	$ –	$ 1.4
Malicious code (+9%)	$ 0.2	$ 0.9	$ 0.2	$ –	$ 1.4
Stolen devices (+12%)	$ 0.4	$ 0.4	$ 0.1	$ 0.1	$ 1.0
Ransomware (+21%)	$ 0.2	$ 0.3	$ 0.1	$ 0.1	$ 0.7
Botnets (+12%)	$ 0.1	$ 0.2	$ 0.1	$ –	$ 0.4
Total cost by consequence	$ 4.0	$ 5.9	$ 2.6	$ 0.5	**$ 13.0**

Fig. 1: Cost of different types of cyberattacks per organization in 2018. The average cost of malware attacks is $2.6 million, which is an 11% increase compared to 2017 [6].

Malware detection is a "cat and mouse" game where researchers design novel methods for malware detection, and attackers develop devious ways to circumvent detection. Traditionally, malware detection is performed by anti-virus software (AVS) using either signature or behavior analysis. Signature-based detection is one of the most popular commercial malware detection techniques [7]. Signature-based detectors compare the signature of a program executable with previously stored malware signatures. However, signature-based AVS is not useful for unknown zero-delay malware since the respective signature is absent in the database. In fact, signature-based AVS is not effective even for known malware with polymorphic or metamorphic features. These morphic malware have either a mutation engine or rewrite themselves in each iteration through various program obfuscation techniques. While behavior-based AVS is promising in detecting unknown and morphing malware, they are computation intensive. As a result, they are not suitable for resource-constrained systems such as IoT edge devices that operate under real-time and energy constraints.

Recent research efforts explored designing hardware-assisted malware detection techniques with the hardware as a root of trust. The underlying assumption is that it is difficult to subvert a hardware-based detector since the malware functionality will remain the same. For example, if a malware is designed to transmit GPS data to an external spy, the malware needs to perform the following four major activities: infect the GPS system, monitor the GPS data, log the GPS data, and transmit it outside the GPS. These activities will be performed by the malware irrespective of the executable format and the code structure (pattern). A signature-based AVS is likely to fail to detect this malware if the code structure is modified. However, a hardware-based detector is likely to detect the malware by monitoring the hardware footprints of these malicious operations since the expected behavior will remain the same.

This paper is organized as follows. The threat model is outlined in Section II. Section III surveys the related efforts. Section IV describes our hardware-assisted malware detection using machine learning. Section V presents the experimental results. Finally, Section VI concludes the paper.

II. Threat Model

The threat model assumes that a system has certain vulnerabilities that an attacker can exploit to install a malicious software (malware). In this paper, we consider a wide variety of vulnerabilities that can be broadly classified into the following categories based on attack complexity. (1) *Easy:* It is easy for an attacker to install malware when the system is not protected (e.g., no password or no antivirus tool) or partially protected (e.g., an IoT device with a factory default password). (2) *Moderate:* An attacker needs to obtain the password from the victim (e.g., by fake/spam emails) before installing the malware. (3) *Hard:* An attacker needs to perform a sequence of tasks to be successful. For example, the attacker needs to get access to one peripheral component by using either of the above methods to get access to the central component, and analyze the underlying system to find any software or hardware-level vulnerabilities.

When a malware infects a system, it can launch a wide variety of attacks. We consider the following types of attacks. (1) *Information Leakage*: Malware can eavesdrop and leak secret information (e.g., credit card details or login/password for bank accounts) that can lead to invasion of privacy as well as financial loss. (2) *Erroneous Execution*: Malware can disrupt execution or respond to legitimate requests with wrong information to cause system malfunction. (3) *Denial-of-Service*: Malware can launch a denial-of-service attack on a critical component of the system (e.g., by flooding it with dummy messages) causing significant performance degradation or even system failure. (4) *Adversarial Attack*: Malware can attack the malware detector that can lead to unintended consequences (e.g., failing to prevent a malware or terminating a benign program by mistake).

III. Related Work

There are many promising approaches for hardware-assisted malware detection utilizing Hardware Performance Counters (HPCs) [8]–[15]. HPCs measure hardware events such as number of instructions executed, number of branches taken, number of cache misses, etc. Unfortunately, HPC-based detection leads to an unacceptably high (up to 15%) false positive rate (FPR) [10], [16]. While Virtual Machine (VM) based solutions provide some improvement in FPR [12], they are not useful in practice for several reasons: (i) HPCs measured inside a VM differ from non-VM execution [16], and (ii) only a small set of HPCs can be monitored simultaneously [9]. A recent study reveals that inherent non-determinism in HPCs leads to a high false positive rate [17]. Moreover, adversarial attacks can be launched on HPC-based solutions to further erode the classification accuracy [18]. Most importantly, existing methods have explored only a limited number of hardware features while missing the opportunity of utilizing a wide variety of hardware features for efficient malware detection.

IV. Hardware-Assisted Malware Detection

A major challenge in hardware-assisted malware detection is to maximize detection accuracy while minimizing hardware overhead associated with real-time collection and analysis of hardware traces. Today's System-on-Chip (SoC) designs are equipped with a wide variety of hardware primitives to support various life-cycle activities [19]–[24]. Specifically, we collect traces from three avenues. (1) Embedded Trace Buffer (ETB) is suitable for capturing functional values of a small number of trace signals over a large number of clock cycles. (2) Hardware Performance Counters (HPC) provide statistical behavior in terms of specific features such as cache misses, branch prediction, etc. (3) Network-on-Chip (NoC) traffic provides communication patterns for malware detection. The remainder of this section describes three complementary directions for malware detection utilizing hardware features.

A. HPC-based Malware Detection

Our proposed HPC-based malware detection consists of two important steps: selection of hardware performance counters and training of the machine learning (ML) model. The first step utilized performance analysis (*perf*) tool to monitor various hardware events on a running processor. We explore the following four types of hardware events as critical features.

- *Number of instructions*: The counting of the architecturally executed instructions, which provides a classifier with a global perspective.
- *Number of exceptions*: In this work, we assume that the malware works in a client-server mode, where malicious programs are required to report stolen data to the remote server, which commonly requires interrupting the normal execution. Based on this assumption, the number of exceptions is also taken into consideration.
- *Change of ContextID*: Due to the stealthy nature of malware, implanted malicious code usually works as a side procedure to avoid changing the fundamental functionality of the original program. Therefore, malware usually introduces additional changes in context compared to benign programs during execution.
- *Bus accesses*: It records communications on read and write channels, and provides a comprehensive overview of program's bus accesses.

The next step is applying these data to train the ML classifier. However, this is challenging due to the existence of obfuscation techniques. A sophisticated adversary can always add redundant non-profitable instructions to mess up the system-wide statistics of the running software, which enables the malicious program to mimic the pattern of benign programs. Therefore, the straightforward way of formatting the inputs by crafting vectors composed of the above selected features is vulnerable towards obfuscation techniques. To address this, our approach uses HPC to sample hardware events in multiple timestamps and record the difference, whereby we provide time-sequential information of the running program instead of overall statistics. Based on this, *Recurrent Neural Network* (RNN) is chosen as the ML model due to its ability to handle time-sequential data. Moreover, in the training phase of RNN, information

corresponding to the previous step will also be fed into the architecture to supply extra information as shown in Figure 2. This enables RNN models to extract information concealed in adjacent inputs for self refinement. After a sufficient number of training iterations, the trained RNN model is utilized for malware detection.

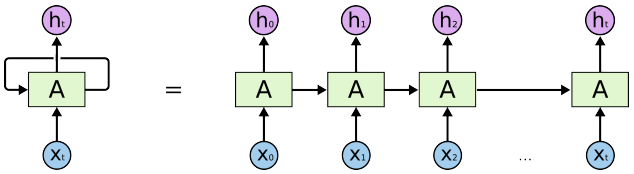


Fig. 2: Recurrent neural network for malware detection.

B. Malware Detection using Embedded Trace Buffer

We utilize embedded trace buffer (ETB) to assist malware detection. The proposed ETB-based malware detection is similar to HPC based approach except the fact that ETB provides a deeper insight into the changes of specific register contents in specific clock cycles. The traces collected by ETB can be viewed as a $w \times d$ table, where w is the *width* and d is the *depth* of the table. It represents the recorded values of w traced signals over d clock cycles. Naturally, each row in the table, i.e, the values of all traced signals in a specific clock cycle, can be considered as an important feature for machine learning model.

The collected traces are then used to train a machine learning classifier. Since we are handling time-sequential data, RNN would be a suitable ML model. Therefore, the subsequent steps would be similar to HPC-based malware detection. Figure 3 provides an overview of ETB-based malware detection. Note that ETB is suitable for capturing functional values of a small number of trace signals over a large number of clock cycles. Therefore, it provides a wider range of supervision over a given time span. Obviously, hardware-assisted malware detection techniques should monitor the behavior of software at runtime. Therefore, relying on single-cycle data is not effective since malicious behavior usually consumes several sequential cycles. Moreover, single-cycle based strategies are likely to mispredict a benign software as malicious. This is due to the fact that malware also contains normal operations, and considering these benign operations as important features of malware can lead to misclassification. By utilizing ETB, our approach is guaranteed to mitigate this mistake.

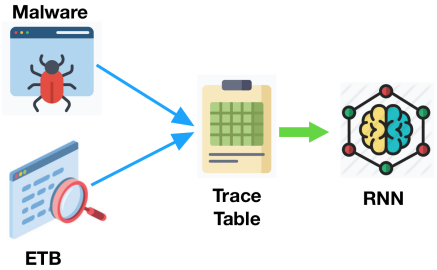


Fig. 3: Overview of ETB-based malware detection.

C. Malware Detection using NoC Traffic Analysis

SoCs utilize the on-chip interconnection network, commonly referred to as the Network-on-Chip (NoC) [25], to communicate with various Intellectual Property (IP) cores such as processor, memory, etc. Due to the distributed nature of the NoC across the chip, attackers can exploit the NoC resources to launch attacks. An implanted malware can launch a wide variety of attacks in NoC-based SoCs ranging from data integrity attacks, eavesdropping attacks, buffer overflow attacks to denial-of-service (DoS) attacks. One commonly explored threat model is a DoS attack that floods the NoC with packets causing NoC congestion leading to performance degradation and deadline violations [26], [27]. In this paper, we explore the mitigation of DoS attacks launched by malware using machine learning.

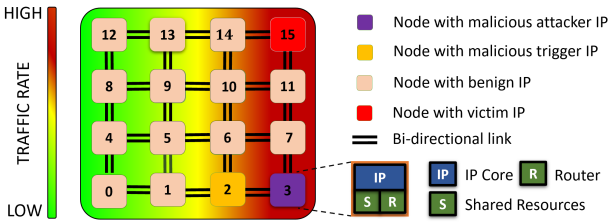


Fig. 4: Example DoS attack from a malicious attacker IP (activated by a malicious trigger IP) to a victim IP in a mesh NoC setup creating high traffic near the victim IP.

Figure 4 shows an illustrative example of an attack scenario. The attack is analogous to the "Mirai" botnet behavior in the computer networks domain. A malicious trigger IP (TIP) at node 2 turns the IP at node 3 into a remotely controlled malicious attacker IP (MIP). The MIP injects a lot of packets targeting its victim IP (VIP) at node 15 that results in a traffic spike in the routing path (path from node 3 to node 15 according to the routing protocol) for a specific period of time. Since the VIP is receiving significantly more packets than it is designed to handle, this leads to performance degradation and in some cases, can cause full system failure.

In this paper, we explore a mitigation technique based on ML that statically (design time) trains ML models and then uses the trained models to detect attacks during runtime. The features to train the models are extracted from NoC traffic when flits[1] are transferred between routers. The extracted features only relate to packet meta-data and does not rely on the packet content. For example, the maximum number of flit arrivals in the NoC within a time window, the cumulative number of flit arrivals in the NoC within a time window, number of hops of a flit from the source to the destination, and ports used by flits to enter and exit the routers are used as features. Our method can be implemented together with other security schemes because the ability to observe the packet content is not required [26]–[31].

V. Experiments

This section is organized as follows. First, we describe the experimental setup. Next, we discuss malware and benign benchmarks. Finally, we present malware detection results using various hardware features.

A. Experimental Setup

We have developed a hardware trace collection and malware detection framework using Xilinx Zynq-7000 SoC

[1]each packet is broken into small pieces called flits (flow control units).

ZC702 evaluation kit connected to a computer running Windows 10 operating system with Xilinx SDK 2017.3 installed. Our controlling computer has an Ethernet port and a USB port to connect to the evaluation board. We installed a Linux-based operating system on the evaluation board SD card and ran malware that targets Linux-based operating systems. The Xilinx ZC702 evaluation board (Figure 5) ran the malware, and the connected computer (using Xilinx SDK) accessed the trace data from the pair of ARM Cortex-A9 processors on the evaluation board. Xilinx SDK includes a Xilinx System Debugger, which launches selected programs on the board and allows access to register/trace values in the board processors as programs run.

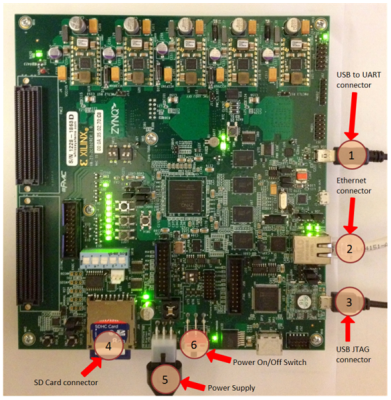


Fig. 5: ZYNQ SoC evaluation board.

To prepare the computer to connect to the board, we performed the following three steps. (1) We set the computer's IP address to connect to the evaluation board. (2) We disabled the computer's firewall at the Ethernet port to connect to the evaluation board (if not, the firewall may block communication between the board and computer). Antivirus programs on the host computer may also block malware from running through Xilinx SDK. We temporarily disabled the antivirus program on the computer to resolve this issue. (3) Once the board was physically connected to the computer using the Ethernet and JTAG ports (numbered as 2 and 3, respectively, in SFigure 5), we established a serial connection between the computer and board to manipulate the board using the computer. For this connection, we used the serial connection option in Xilinx SDK.

To prepare the evaluation board to connect to the computer and to the Xilinx System Debugger, we performed the following six steps. (1) We downloaded a ZC702-compatible Linux OS image to the SD card. We used $xilinx-zc702-2017_34.9.0-xilinx-v2017.3$, which was provided by Xilinx and generated using PetaLinux. (2) We booted the board in SD mode as described in the ZC702 user manual [32]. (3) Through the serial connection to the computer, we set the evaluation board's IP address (the IP address is required when running programs using the System Debugger). (4) We compiled all desired programs in Xilinx SDK, and launched these programs on the board using debug configurations. We then set the target in debug configurations to the board IP address, which was set in the previous step. To access register data at different points in the program's runtime, we added breakpoints to the program. The register values update at each breakpoint. (5) For malware with a client-server model, we configured the server address by hard-coding its IP address in the associated location in the malware source code. (6) In cases where malware requires a server and a client, we ran the server without system debugger by transferring the compiled server binary to the board and running the server program. We then ran the client program using the system debugger.

B. Malware Benchmarks

The machine learning model's ability to correctly detect malware was tested using three real-world malware benchmarks: Bashlite, PNScan, and Mirai [33].

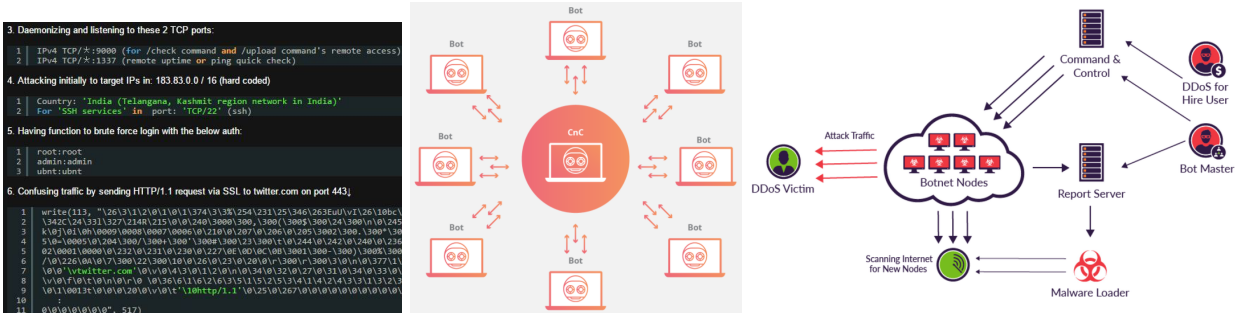
PNScan: It opens a backdoor for other malware. As its name would suggest, PNScan scans the infected device for information to infect more devices in its network. This Trojan uses brute force to obtain the victim's router's access password. Upon finding this password, the PNScan program will bypass password protection to download other malicious programs to the router. PNScan makes vulnerable devices more vulnerable by actively weakening security against other malware. Figure 6a shows visuals of some of the malicious steps of PNScan.

Bashlite: It infects many devices to form a botnet. If a botnet is formed successfully, the attacker may remotely orchestrate DDoS (Distributed Denial of Service) attacks and download other malware by sending commands to infected devices (aka "bots"). Figure 6b shows a visual representation of the form of botnet used by Bashlite (also known as Gafgyt). This malware utilizes a client-server model in which the attacker's device functions as the command-and-control (CnC) server, and the infected devices function as clients. The client bots constantly poll for server commands. Large botnets can overwhelm target servers by simultaneously making requests when the attacker sends the command.

Mirai: It is a more sophisticated version of Bashlite. Mirai includes a wider variety of commands, and can infect a wider variety of IoT devices. Because Mirai is compatible with more devices, it has the potential to build a larger botnet. The number of devices included in the botnet improves the botnet's ability to overwhelm target servers. The wider range of vulnerable devices also improves Mirai malware's ability to steal information from these devices. Figure 6c shows an overview of a Mirai botnet.

C. Benign Benchmarks

System binaries like **netstat** and **ping** represent benign software when examining the accuracy of the provided machine learning model. We generated hardware trace data for these system binaries because malicious developers often use these binaries in malware, but benign programs can use them in a completely legitimate and harmless context. For example, bots in a botnet may frequently run *ping* to check their connection to a malicious server, but harmless applications may run the exact same function to check a connection to a benign server. Similarly, developers may use *netstat* in the context of checking on the status of a network or in the context of accessing a device's information for malicious use. Because many malicious programs rely on the misuse of common system functionality, any machine learning model intended to detect malware should be able



(a) Code snippets of PNScan's malicious behavior [34].

(b) The CnC server communicates with bots in Bashlite [35].

(c) Mirai is a sophisticated botnet [36].

Fig. 6: Three popular malware benchmarks: Bashlite, PNScan, and Mirai

to filter out benign use of this functionality, or it will continuously generate false positive predictions.

D. Results using ETB and HPC based Analysis

We have explored the effectiveness of embedded trace buffer (ETB)-based malware detection compared to using hardware performance counters (HPC). We ran malicious (Bashlite and Mirai) and benign (ping and netstat) applications. The traces are fed to the four classifiers: K-nearest neighbor (KNN), Random forest (RF), Decision tree (DT) and Neural Networks (NN) [37]. Figure 7 shows that HPC-based detection leads to high false positive rate (11.3% on average), while ETB based analysis reduces the false positive rate (2.5% on average).

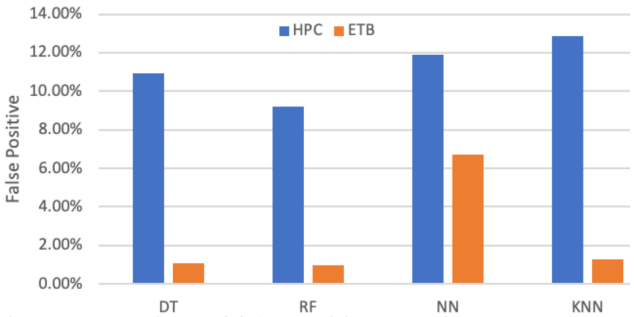


Fig. 7: Comparison of false positive rates for HPC and ETB-based malware detection

Note that ETB and HPC provide complementary abilities - ETB provides detailed insight into internal signal states whereas HPC provides global perspectives. In reality, an effective combination of HPC and ETB based analysis would combine their inherent advantages.

E. Results using NoC Traffic Analysis

To explore the efficiency of our ML-based DoS attack detection using NoC traffic features, we model a 4×4 mesh NoCs with 16 IP cores using the GARNET2.0 [38] interconnection network model. NoC traffic features were extracted using the gem5 cycle-accurate full-system simulator [39] for both normal and attack scenarios as shown in Figure 8. To train the ML models, datasets were developed by simulating communication for 500,000 cycles with synthetic traffic between IP 0 and IP 15, and IP 3 and IP 12 as the normal scenario. To simulate the attack scenario, IP 3 sent a burst of packets targeting IP 15. The burst lasted for 1,000 cycles and was triggered by a packet from IP 2 periodically every 10,000 cycles. Furthermore, we explored the attack behavior by varying the burst packet injection rate (x) from $x = 0.1$ to $x = 0.5$, while keeping the packet injection rate of the normal scenario at 0.01. The obtained traces are fed to the four classifiers: Decision tree (DT), Random forest (RF), Logistic regression (LR), and K-nearest neighbor (KNN).

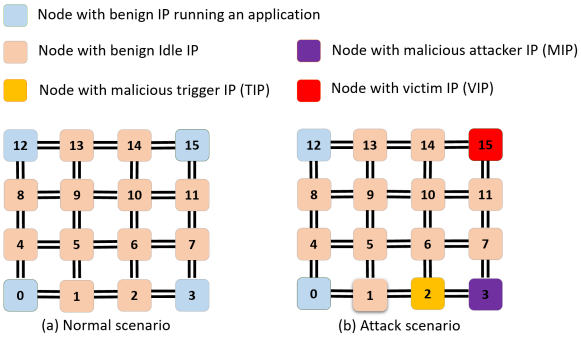


Fig. 8: Architectures used to extract NoC traffic features.

The trained models were tested against different attack scenarios by changing the placement of the TIP and the MIP. Figure 9 shows attack detection accuracy given by different classifiers with different burst packet injection rates (x). Each bar in the figure shows the average detection accuracy considering all the test cases. Figure 10 shows the false positive rate, which reduces from 32.83% (on average) to 4.62% (on average) with increasing x. As expected, a more severe attack scenario (represented by the increased traffic rate) is classified as an attack with high accuracy and less false positives.

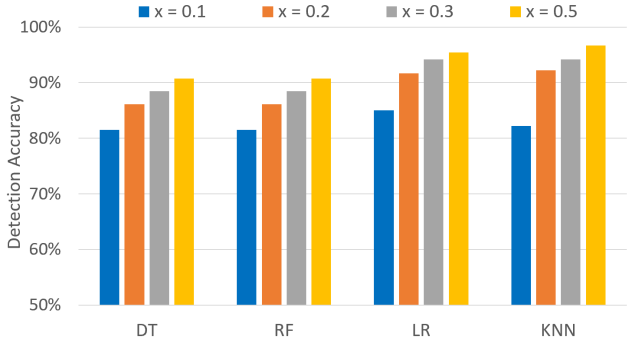


Fig. 9: Comparison of accuracy of DoS attacks with different packet injection rates

Compared to previous approaches that explored traffic latency comparison [40] and packet arrival monitoring [27], our ML-based approach exhibits good performance not only for application mappings and TIP/MIP/VIP placements that

the model was trained on but also for new configurations. In other words, our proposed ML-based approach can be used to detect malware irrespective of their location on the NoC.

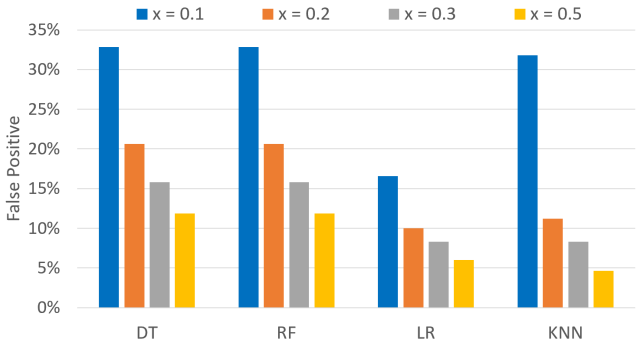


Fig. 10: Comparison of false positive rate of DoS attacks with different packet injection rates

VI. Conclusion

Malware is a serious threat to modern computing systems. Existing software-based solutions, such as anti-virus programs, are not effective since they rely on matching patterns (signatures) that can be easily fooled by carefully crafted malware with obfuscation or other deviation capabilities. This paper explored hardware-assisted malware detection to address the limitation of software-based solutions. Our experimental results demonstrated that machine learning can be effective in malware detection utilizing hardware performance counters, embedded trace buffer as well as on-chip network traffic analysis. Our future work will explore the effectiveness of explainable machine learning in accurate classification of benign and malicious programs as well as interpretation of malware detection results for localization of malicious behaviors [41].

Acknowledgements

This work was partially supported by the National Science Foundation (NSF) grants SaTC-1936040 and CCF-1908131.

References

[1] G. McGraw and G. Morrisett, "Attacking malicious code: A report to the infosec research council," *IEEE software, 17(5)*, pp. 33–41, 2000.
[2] A. Kapravelos *et al.*, "Revolver: An automated approach to the detection of evasive web-based malware." in *USENIX Security Symposium*, 2013, pp. 637–652.
[3] R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," *IEEE Security & Privacy*, vol. 9, no. 3, pp. 49–51, 2011.
[4] E. Chien, L. O'Murchu, and N. Falliere, "W32. duqu: The precursor to the next stuxnet." in *LEET*, 2012.
[5] E. Cozzi *et al.*, "Understanding linux malware," in *IEEE Symposium on Security & Privacy*, 2018.
[6] K. Bissell and L. Ponemon, "The cost of cybercrime," 2019.
[7] Z. Bazrafshan *et al.*, "A survey on heuristic malware detection techniques," in *IEEE CIKT*, 2013, pp. 113–120.
[8] C. Malone, M. Zahran, and R. Karri, "Are hardware performance counters a cost effective way for integrity checking of programs," in *ACM Workshop on Scalable trusted computing*, 2011, pp. 71–76.
[9] X. Wang and R. Karri, "Numchecker: Detecting kernel control-flow modifying rootkits by using hardware performance counters," in *ACM/IEEE Design Automation Conference*, 2013, pp. 1–7.
[10] J. Demme *et al.*, "On the feasibility of online malware detection with performance counters," in *ACM SIGARCH Computer Architecture News*, vol. 41, 2013, pp. 559–570.
[11] A. Tang *et al.*, "Unsupervised anomaly-based malware detection using hardware features," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2014, pp. 109–129.
[12] M. Kazdagli *et al.*, "Quantifying and improving the efficiency of hardware-based mobile malware detectors," in *IEEE/ACM International Symposium on Microarchitecture*, 2016, p. 37.
[13] X. Wang *et al.*, "Confirm: Detecting firmware modifications in embedded systems using hardware performance counters," in *International Conference on Computer-Aided Design*, 2015, pp. 544–551.
[14] X. Wang et al., "Hardware performance counter-based malware identification and detection with adaptive compressive sensing," *ACM TACO*, vol. 13, no. 1, p. 3, 2016.
[15] V. Jyothi *et al.*, "Brain: Behavior based adaptive intrusion detection in networks: Using hardware performance counters to detect ddos attacks," in *VLSID*, 2016, pp. 587–588.
[16] B. Zhou *et al.*, "Hardware performance counters can detect malware: Myth or fact?" in *ACM Asia Conference on Computer and Communications Security*, 2018, pp. 457–468.
[17] K. Basu *et al.*, "A theoretical study of hardware performance counters-based malware detection," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 512–525, 2019.
[18] S. M. P. Dinakarrao *et al.*, "Adversarial attack on microarchitectural events based malware detectors," in *DAC*, 2019, p. 164.
[19] P. Mishra and F. Farahmandi (Editors), *Post-Silicon Validation and Debug*. Springer, 2018.
[20] K. Rahmani and P. Mishra, "Feature-based signal selection for post-silicon debug using machine learning," *IEEE Transactions on Emerging Topics in Computing*, 2017.
[21] K. Rahmani, S. Ray, and P. Mishra, "Postsilicon trace signal selection using machine learning techniques," *IEEE Transactions on VLSI Systems*, vol. 25, no. 2, pp. 570–580, 2016.
[22] P. Mishra *et al.*, "Post-silicon validation in the soc era: A tutorial introduction," *IEEE Design & Test*, vol. 34, no. 3, pp. 68–92, 2017.
[23] K. Rahmani, S. Proch, and P. Mishra, "Efficient selection of trace and scan signals for post-silicon debug," *IEEE Transactions on VLSI Systems*, vol. 24, no. 1, pp. 313–323, 2015.
[24] K. Basu and P. Mishra, "Rats: Restoration-aware trace signal selection for post-silicon validation," *IEEE Transactions on VLSI Systems*, vol. 21, no. 4, pp. 605–613, 2012.
[25] S. Charles, C. A. Patil, U. Y. Ogras, and P. Mishra, "Exploration of memory and cluster modes in directory-based many-core cmps," in *International Symposium on Networks-on-Chip (NOCS)*. IEEE, 2018, pp. 1–8.
[26] S. Charles, Y. Lyu, and P. Mishra, "Real-time Detection and Localization of DoS Attacks in NoC based SoCs," in *DATE*, 2019.
[27] S. Charles, Y. Lyu and P. Mishra, "Real-time detection and localization of distributed dos attacks in noc based socs," *IEEE Transactions on CAD (TCAD)*, 2020.
[28] S. Charles, M. Logan, and P. Mishra, "Lightweight Anonymous Routing in NoC based SoCs," in *Design Automation & Test in Europe (DATE)*, 2020.
[29] S. Charles and P. Mishra, "Reconfigurable network-on-chip security architecture," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 25, no. 6, pp. 1–25, 2020.
[30] S. Charles and P. Mishra, "Lightweight and trust-aware routing in noc based socs," in *IEEE Annual Symposium on VLSI (ISVLSI)*, 2020.
[31] S. Charles and P. Mishra, "Securing network-on-chip using incremental cryptography," in *IEEE Annual Symposium on VLSI*, 2020.
[32] "Zc702 soc evaluation board," https://www.xilinx.com/.
[33] K. Angrishi, "Turning internet of things(iot) into internet of vulnerabilities (iov) : Iot botnets," *CoRR*, vol. abs/1702.03681, 2017.
[34] "Linux.pnscan trojan is back to compromise routers and install backdoors," https://securityaffairs.co/wordpress/50607/malware/linux-pnscan-return.html.
[35] "What is a ddos botnet?" https://www.cloudflare.com/learning/ddos/what-is-a-ddos-botnet/.
[36] A. Shoemaker, "How to identify a mirai-style ddos attack," https://www.imperva.com/blog/how-to-identify-a-mirai-style-ddos-attack/.
[37] K. Basu *et al.*, "Preempt: Preempting malware by examining embedded processor traces," in *Proceedings of the 56th Annual Design Automation Conference 2019*. ACM, 2019, p. 166.
[38] N. Agarwal *et al.*, "Garnet: A detailed on-chip network model inside a full-system simulator," in *ISPASS*, 2009, pp. 33–42.
[39] N. Binkert *et al.*, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, p. 1–7, 2011.
[40] R. JS, D. M. Ancajas, K. Chakraborty, and S. Roy, "Runtime detection of a bandwidth denial attack from a rogue network-on-chip," in *International Symposium on Networks-on-Chip*, 2015, p. 8.
[41] Z. Pan, J. Sheldon, and P. Mishra, "Hardware-assisted malware detection using explainable machine learning," in *IEEE International Conference on Computer Design (ICCD)*, 2020.