Malware Detection Using Hardware Trace Analysis

Jennifer Sheldon
Department of Electrical and Computer Engineering
University of Florida, USA

***Abstract*—Malware (malicious software) is a dangerous threat today due to our increasing reliance on computing devices. Currently, state-of-the-art threat detection methods include machine learning models that identify malware-based on patterns in software. While there are some promising efforts in hardware-assisted malware detection, there is no comprehensive study on efficient hardware trace analysis for malware detection. In this honors thesis, I investigate the impact of hardware traces on explainable machine learning models for hardware-based malware detection. My research has led to generation of useful hardware traces from a state-of-the-art System-on-Chip (SoC) board running real-world malware benchmarks. The experimental results demonstrate that explainable machine learning based on hardware trace analysis can accurately differentiate between malware and benign programs.**

***Index Terms*—Malware Detection, Hardware Security, Machine Learning**

I. Introduction

Computers have inherited the logical capabilities, resourcefulness, and helpfulness of their human creators. Unfortunately, they have also inherited our capacity for malice. Malware, or malicious software, represents an efficient and powerful form of automated malice which we must defend against. Figure 1 shows our typical interactions in daily life with a wide-variety of computing devices, popularly known as Internet-of-Things (IoT) devices. Due to our increasing reliance on IoT devices, we have become more vulnerable to malicious software (malware) attacks.

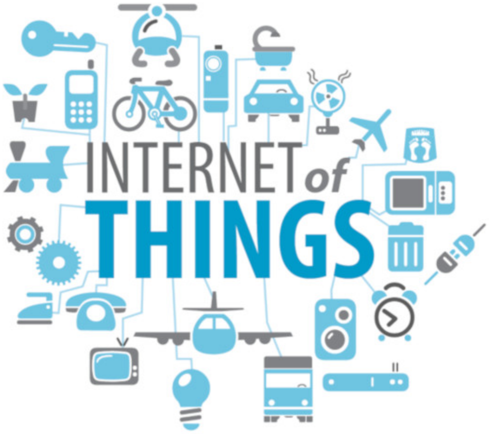
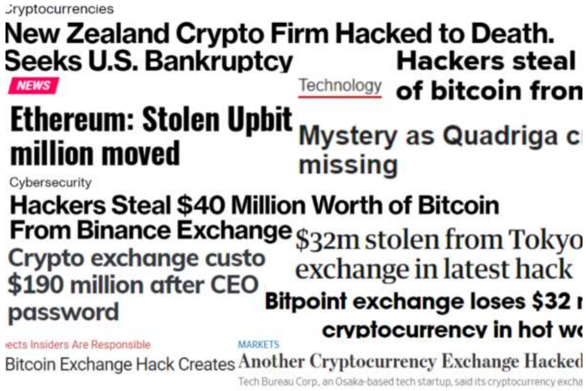


Figure 1: As more aspects of our daily life depend on computing devices, we become more vulnerable to malware attacks [1].

Figure 3 demonstrates that malware has made repeated disturbing appearances in the news. Malware has already established itself as a financial and infrastructural threat in cases where botnets have stolen millions of users' financial and identity information, viruses have reproduced to overload email servers, and servers have been overwhelmed by DDoS (Distributed Denial of Service) attacks [2]. In 2013, malicious developers used information-scanning malware [3] similar to PNScan malware to steal 40 million Target customers' credit card numbers [4]. Unfortunately, this is not an unusual occurrence. Figure 3 shows the average annual cost of malware attacks per surveyed company (from 355 companies) in a survey done by the Ponemon Institute [5] [6].



ryptocurrencies
New Zealand Crypto Firm Hacked to Death. Seeks U.S. Bankruptcy

NEWS
Ethereum: Stolen Upbit million moved

Technology
Hackers steal of bitcoin fron

Mystery as Quadriga c missing

Cybersecurity
Hackers Steal $40 Million Worth of Bitcoin From Binance Exchange

Crypto exchange custo $190 million after CEO password

$32m stolen from Tokyo exchange in latest hack

Bitpoint exchange loses $32 cryptocurrency in hot w

ects Insiders Are Responsible
Bitcoin Exchange Hack Creates

MARKETS
Another Cryptocurrency Exchange Hacked
Tech Bureau Corp., an Osaka-based tech startup, said its cryptocurrency excha

Figure 2: The financial impacts of malware continuously appear in the news [7].

The threat of malware is not, however, limited to cyberspace. As the world has become more computerized, malware attacks grow as a physical threat. For example, let us consider the Therac-25 radiation therapy machine and the Boeing 737 aircraft. The Therac-25 overdosed patients with radiation due to a software error [8], and multiple Boeing 737 aircrafts have crashed, likely due to a software error in the aircrafts' MCAS in-flight control software [9]. While these disasters were the results of human error rather than malice, they highlight the vulnerability of mission-critical devices that an attacker can exploit. If accidents can lead to such catastrophic results, what dangers could actively malicious developers hide in devices?

Malware detection is a "cat and mouse" game wherein researchers develop novel methods for malware detection, and malicious developers attempt to circumvent detection. As a result, malicious developers have employed methods

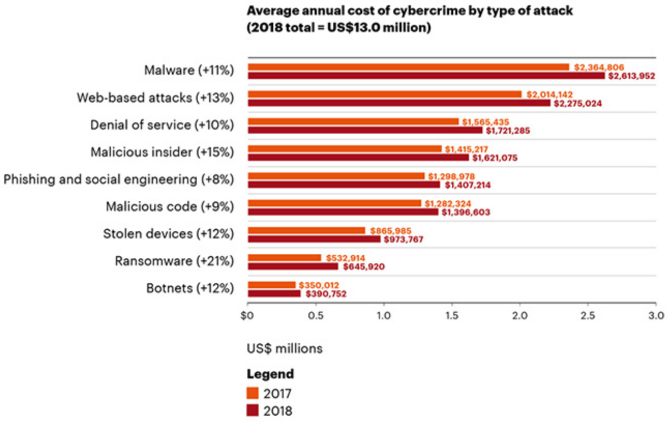


Figure 3: Malware has significant financial impacts on companies. In this context, botnets ransomware, and denial of service are associated with malware [5] [6].

such as code obfuscation to avoid discovery by detection methods which depend on software analysis. To address this, researchers have developed new methods using machine learning to analyze hardware traces. The trace data I generated for this honors thesis tested the accuracy of a hardware-trace-based explainable machine learning model for malware detection.

In this honors thesis, I make the following contributions:

1) Setting up a state-of-the-art System-on-Chip (SoC) board running real-world malware benchmarks.
2) Identification and generation of a limited set of useful trace signals for effective malware detection.
3) Utilization of generated hardware traces using explainable machine learning model to accurately differentiate between malware and benign programs.

The remainder of this thesis is organized as follows. Section II provides an overview of existing malware detection and machine learning techniques. Section III describes my proposed method for malware detection using hardware traces. Section IV presents the experimental results. Finally, Section VI concludes the thesis.

II. Background and Related Work

This section provides an overview of existing approaches to highlight the contributions of this thesis. First, I provide a brief background for machine learning based malware detection. Next, I outline related efforts in malware detection.

A. Background: Machine Learning

Many researchers and companies aim to efficiently identify malware developers' ever-expanding arsenal of attack techniques using machine learning. Machine learning generates predictions about input data after being "trained" using sample data. For example, machine learning models that predict whether a given program is malicious can be "trained" by supplying the model with a collection of malicious and benign program data and correctly tagging data in each group [10]. In theory, the model should be able to identify data patterns specific to malicious programs and use these patterns to predict whether input data came from a malicious or benign program based on the identified patterns.

Machine learning models can theoretically sift through questionable program data more quickly than human experts [10]. For the time and efficiency benefits of machine learning to be useful, however, the models must be accurate. We can demonstrate the accuracy of these models through rigorous testing. If the tested model performs at or above an acceptable threshold, it is a functional model.

B. Related Work in Malware Detection

Early standards of malware detection used static analysis [11], [12], which depends on software filters to detect malicious software signatures. These signatures are identified using machine learning and/or human knowledge. As malicious developers recognized which patterns would be caught by static analysis, they began obfuscating malicious software to avoid detection [13]. Malware may also exploit vulnerabilities in the static analysis software itself to avoid detection [14]. Researchers then suggested dynamic analysis (which detects malicious behavior during runtime) as a solution to this problem [15], [16].

To solve the obfuscation and exploitation problems associated with software-based malware detection, researchers have turned to hardware-based malware detection. Recent efforts in hardware-based malware detection have utilized Hardware Performance Counters [17]–[19] and Embedded Trace Buffers [20]. Of these, the Hardware Performance Counter-based detection method shows frequent false-positives [17] and performance penalties. On the other hand, detection methods utilizing Embedded Trace Buffers are favorable because they show high detection accuracy (as high as 94%) [20].

However, hardware-based malware detection methods are not without fault. They have issues identifying multi-cycle malicious behavior, and data generated by hardware traces must be pre-processed to differentiate between benign and malicious use of common constructs (for instance, a benign vs a malicious use of netstat, as discussed later). Current hardware-based detection methods also obfuscate intermediate processing, so the final prediction result may be questionable without the user realizing that this is the case because the steps taken to make the prediction may be faulty.

III. Malware Detection using Hardware Traces

Figure 4 provides an overview of the proposed malware detection model. It consists of four major steps: (i) running malicious and benign programs on an SoC board, (ii) collecting useful hardware traces, (iii) utilizing machine learning for malware detection using hardware traces, and (iv) evaluating the prediction accuracy to fine tune the model. The remainder of this section describes these steps in detail.

Run Malicious and Benign Programs: This first step is to setup an environment to execute both malware and benign programs. Section IV-A describes our experimental setup with

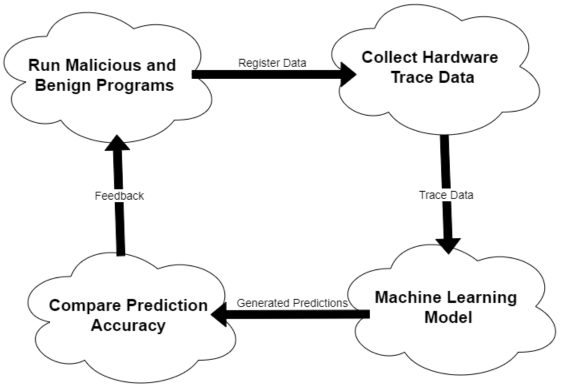

Run Malicious and Benign Programs
Register Data
Collect Hardware Trace Data
Feedback
Trace Data
Compare Prediction Accuracy
Generated Predictions
Machine Learning Model

Figure 4: Overview of my proposed method.

a state-of-the-art SoC board that can run popular malware (described in Section IV-B) as well as benign programs.

Generation of Hardware Traces: The second step is to generate useful hardware traces from execution of malicious and benign programs. In this context, "hardware traces" refers to content of registers, caches as well as any other trace buffers over the course of a program's run time. The data must be collected over time to identify patterns in behavior rather than giving snapshots at only the start and end of the program. For example, registers may be reused over the course of the program's run, so limited snapshots give an inaccurate representation of the program's state at intermediate steps. This provides a trade-off between accuracy and speed. A higher sampling frequency can lead to accurate prediction but it can be slower due to larger volume of data analysis. On the other hand, slower sampling frequency can lead to faster prediction (less data analysis) at the cost of prediction accuracy.

Malware Detection using Machine Learning: The third steps is to feed the trace data to machine learning model for malware detection. Our proposed machine learning model both predicts whether the program which generated the traces is malicious and provides justification for the prediction. As discussed in Section IV, our proposed model consistently performs better compared to the state-of-the-art malware detection method.

Comparison of Prediction Accuracy: The final step is to compare the accuracy (correct predictions, false positives, and false negatives) of our proposed machine learning model to state-of-the-art machine learning models for malware detection. This information can be used as feedback for altering trace collection and analysis techniques for improved malware detection accuracy.

IV. Experiments

This section is organized as follows. First, I describe the experimental setup using an SOC board. Next, I describe three malware benchmarks. Finally, I present the prediction accuracy results.

A. Experimental Setup: SoC Board

I originally planned to use the Embedded Trace Buffer (ETB) on the board from the Xilinx Zynq-7000 SoC ZC702 board to collect data. The standard method for collecting this data involves using expensive external hardware. Another method of collecting this data is the use of a gdb server, but the available evaluation board and OS cannot use a gdb server. I later attempted to access the ETB location directly in memory. While I was able to access this memory area using Xilinx SDK, I could not enable the ETB by directly manipulating the associated control register in Xilinx SDK.

I developed a workaround to address the limitations discussed above. I collected data using the items included in the Xilinx Zynq-7000 SoC ZC702 evaluation kit and a computer with a Windows 10 operating system with Xilinx SDK 2017.3 installed. The computer must have an Ethernet port and at least one USB port to connect to the evaluation board. The proposed model identifies malware for Linux and Android operating systems, so I installed a Linux-based operating system on the evaluation board and ran malware that targets Linux-based operating systems.

In this experimental setup, the Xilinx ZC702 evaluation board (Figure 5) included in the evaluation kit ran the malware, and the connected computer (using Xilinx SDK) accessed the register 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 values in the board processors as programs run.

To prepare the computer to connect to the board, I completed the following steps:

1) Set the computer's IP address to connect to the evaluation board.
2) 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. If this happens, I have two options: either temporarily disable the antivirus program or reconfigure its settings to allow the malware file to run.
3) Once the board is physically connected to the computer using the JTAG and Ethernet connections, I established a serial connection between the computer and board to manipulate the board using the computer. For this connection, I used the serial connection option in Xilinx SDK with a baud rate of 115200, no flow control, 8 data bits, 1 stop bit, 5 second timeout, and no parity bit. The port number depends on the number associated with the USB port connected to JTAG.

To prepare the evaluation board to connect to the computer and to the Xilinx System Debugger, I completed the following steps:

1) Downloaded a ZC702-compatible Linux OS image to the SD Card. I used $xilinx-zc702-2017_34.9.0-xilinx-$

v2017.3, which was provided by Xilinx and generated using PetaLinux.

2) Boot the board in SD mode as described in the ZC702 user manual [21].
3) Through the serial connection to the computer, set the evaluation board's IP address (the IP address is required when running programs using the System Debugger).
4) Compiled all desired programs in Xilinx SDK, and launched these programs on the board using Debug Configurations. 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, I added breakpoints to the program. The register values update at each breakpoint.
5) In cases where malware requires a server and a client, I ran the server without System Debugger by transferring the compiled server binary to the board and running the server program. I then ran the client program using the System Debugger. The client's run data takes precedence over the server's run data because the machine learning model identifies infected devices, not devices which coordinate malicious attacks.
6) In cases where the server and client cannot connect because the IP addresses do not match, I hardcoded the required IP addresses for the connection.
7) In cases where the server cannot send attack commands because the embedded OS does not have a requested functionality, I hardcoded the attack command into the client program.

B. Malware Benchmarks

The model's ability to correctly detect malware was tested using three real-world malware benchmarks: BASHLITE, PNScan, and Mirai [22].

1) **PNScan** opens a backdoor for other malware. As its name would suggest, PNScan scans the infected device for information. 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 6 shows visuals of some of the primary malicious steps of PNScan.
2) **BASHLITE** infects large number of 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 7 shows a visual representation of the form of botnet used by BASHLITE. This malware utilizes a client-server model in which the attacker's device functions as the 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.

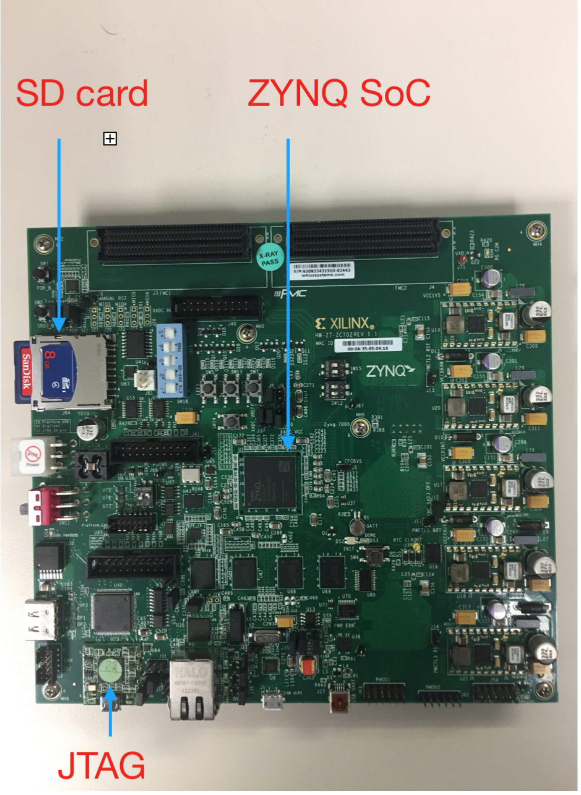
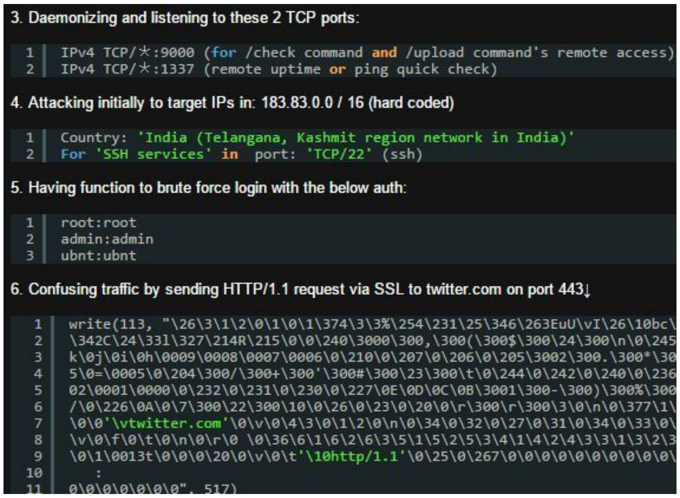


Figure 5: ZYNQ SoC board.



3. Daemonizing and listening to these 2 TCP ports:

```
IPv4 TCP/*:9000 (for /check command and /upload command's remote access)
IPv4 TCP/*:1337 (remote uptime or ping quick check)
```

4. Attacking initially to target IPs in: 183.83.0.0 / 16 (hard coded)

```
Country: 'India (Telangana, Kashmit region network in India)'
For 'SSH services' in  port: 'TCP/22' (ssh)
```

5. Having function to brute force login with the below auth:

```
root:root
admin:admin
ubnt:ubnt
```

6. Confusing traffic by sending HTTP/1.1 request via SSL to twitter.com on port 443↓

```
write(113, "\26\3\1\2\0\1\0\1\374\3\3%\254\231\25\346\263EuU\vI\26\10bc\
\342C\24\331\327\214R\215\0\0\240\3000\300,\300(\300$\300\24\300\n\0\245
k\0j\0i\0h\0009\0008\0007\0006\0\210\0\207\0\206\0\205\3002\300.\300*\30
5\0=\0005\0\204\300/\300+\300'\300#\300\23\300\t\0\244\0\242\0\240\0\236
02\0001\0000\0\232\0\231\0\230\0\227\0E\0D\0C\0B\3001\300-\300)\300%\300
/\0\226\0A\0\7\300\22\300\10\0\26\0\23\0\20\0\r\300\r\300\3\0\n\0\377\1\
\0\0'\vtwitter.com'\0\v\0\4\3\0\1\2\0\n\0\34\0\32\0\27\0\31\0\34\0\33\0\
\v\0\f\0\t\0\n\0\r\0 \0\36\6\1\6\2\6\3\5\1\5\2\5\3\4\1\4\2\4\3\3\1\3\2\3
\0\1\0013t\0\0\0\20\0\v\0\t'\10http/1.1'\0\25\0\267\0\0\0\0\0\0\0\0\0\
   :
0\0\0\0\0\0\0", 517)
```

Figure 6: Visual code snippets of PNScan's malicious behavior. [23]

3) **Mirai** 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

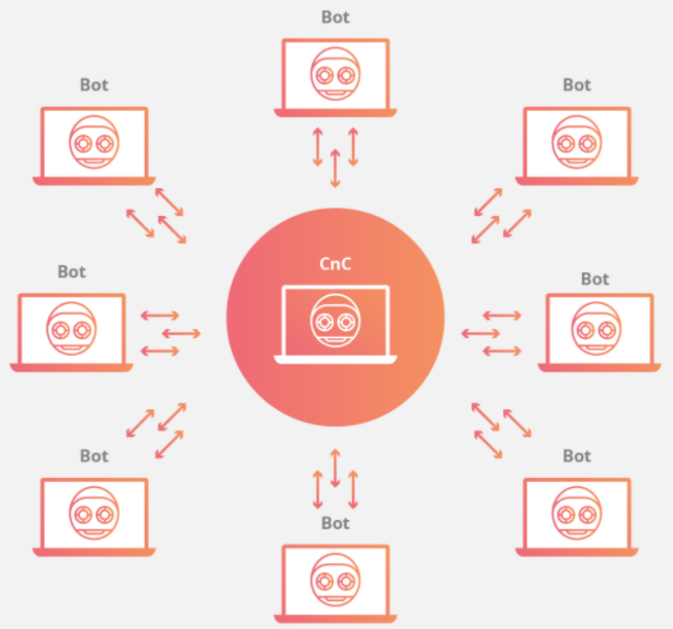


Figure 7: Graphic demonstrating how botnets, BASHLITE botnets included, function. The CNC (Command and Control) server is the human attacker's device, which is used to send attack commands [24].

from these devices. Figure 8 shows an overview of a Mirai botnet.

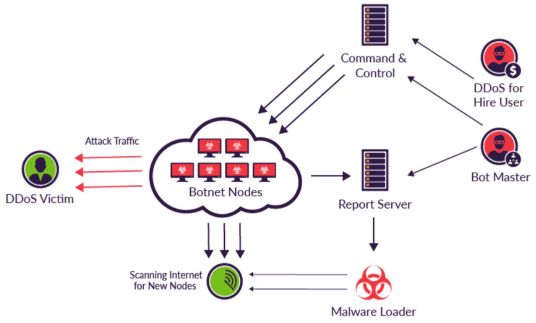


Figure 8: Graphic demonstrating Mirai functionality [25]. Mirai is a more sophisticated botnet.

System binaries like *netstat* and *ping* represent benign software when examining the accuracy of the provided machine learning model. I 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 contexts. For example, bots in a botnet may frequently run ping to check their connection to a malicious server, but legitimate applications may run the exact same function to check a connection to a benign server. Similarly, developers can use netstat in a completely legitimate 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 to filter out benign use of this functionality, or it will continuously generate false positive predictions.

C. *Results: Prediction Accuracy*

To analyze the accuracy of the proposed malware detection model, we compare its accurate and false-positive detection rate against that of the state-of-the-art method, PREEMPT, which uses the hardware data from the Embedded Trace Buffer (ETB) to make black-box predictions [20]. In this case, we compared the model under test to both the "Decision Tree" and "Random Forest" versions of PREEMPT. Figure 9 shows the accuracy of each model when making predictions about BASHLITE malware. The proposed method had a maximum accuracy of 98.9% while PREEMPT had a maximum accuracy of 94.8%. The proposed method performed consistently better than PREEMPT when identifying BASHLITE malware.

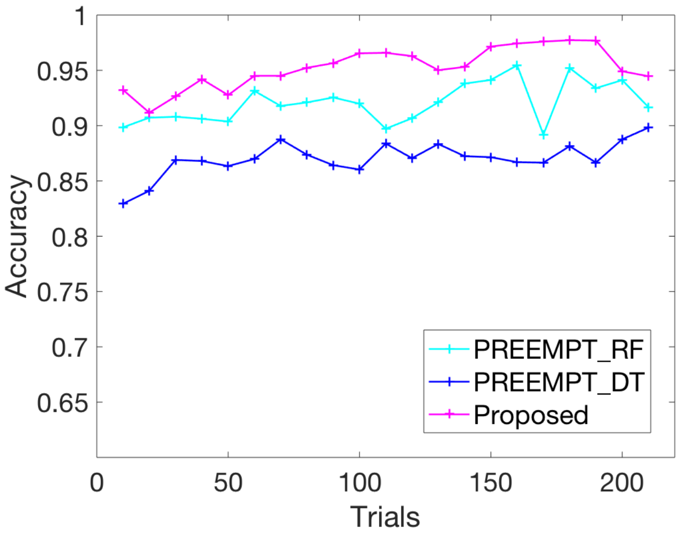


Figure 9: Model Performance on BASHLITE.

Figure 10 shows the accuracy of each model when making predictions about PNScan malware. When identifying PNScan malware from my hardware trace data, PREEMPT's highest accuracy reached about 78% while the proposed model reached a maximum accuracy of 98.9%.

Figure 11 shows the accuracy of each model when making predictions about Mirai malware. The proposed model predicted Mirai malware presence with at best 97.5% accuracy while PREEMPT predicted Mirai malware presence with at best 92.5% accuracy.

Figure 12 shows the rate of false positives for each model when making predictions BASHLITE, PNScan, and Mirai. These results emphasized PREEMPT's tendency to produce more false positives than the proposed model. When running benign programs, the more accurate version of PREEMPT (RF) produced a maximum false positive rate of 44.6%, and the proposed model produced a maximum false positive rate

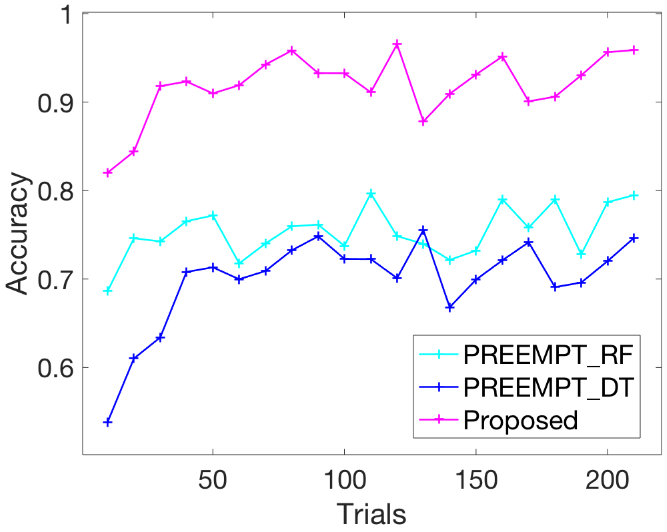


Figure 10: Model Performance on PNScan.

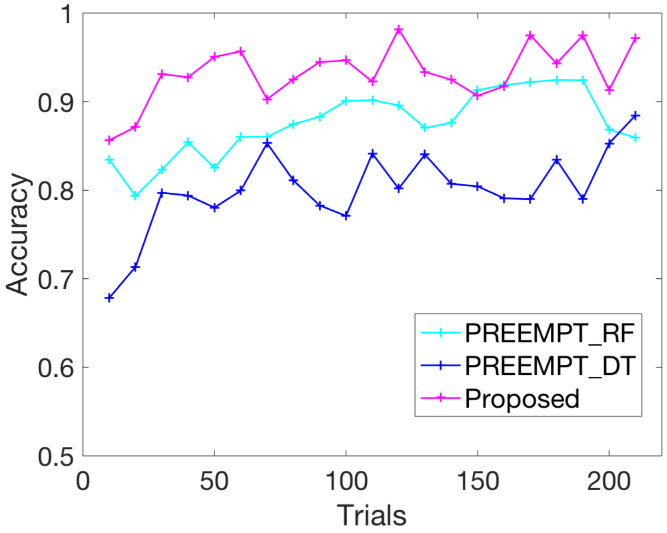


Figure 11: Model Performance on Mirai.

of 9.2%. The results demonstrated that the proposed model performance significantly better than PREEMPT in malware detection.

V. Discussion

In this section, I discuss three important issues related to hardware trace generation and analysis. First, I discuss how to generate traces from the embedded trace buffer without altering the malware/benign programs. Next, I discuss how to model external botnet server. Finally, I discuss the scenarios when minor code changes are needed.

To generate more trace data without interrupting the flow of the tested programs, this method could be modified to utilize the Embedded Trace Buffer (ETB) to provide data for machine learning. This may be done by modifying the standard CoreSight Access Library (a library provided by ARM to manipulate the ETB) to interact with the Xilinx evaluation board.

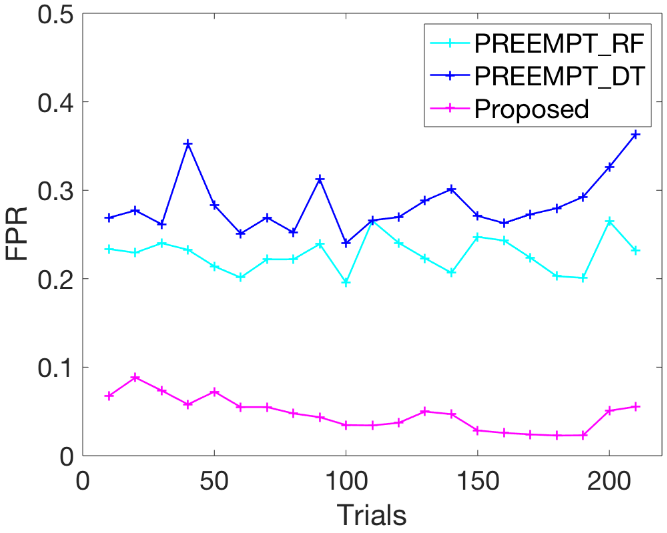


Figure 12: False Positive Rates of all models

Because I was using the Ethernet port to connect the evaluation board to the host computer, I could not connect to an external botnet server. To demonstrate the functionality of the client bot when connected and receiving commands, I ran both server and client on the same board. Running both processes at once could have impacted the generated data. With the aforementioned ETB functionality, I could free the Ethernet port by dumping the trace data to a file on the evaluation board to view later.

In some cases, the evaluation board did not have the capability to fully reproduce part of the malware attack. For example, actual Mirai servers utilize a SQL database, but the evaluation board cannot utilize this functionality. The evaluation board also could not directly send Mirai commands to clients, so these commands had to be hardcoded into the client. In cases where the server could not function fully, I modified the client programs to bypass checks of the server's functionality and behave as if a fully-functional server were sending commands. While the client processes still exhibited malicious behavior, the modifications changed the characteristics of the programs and may decrease the quality of the data.

VI. Conclusions and Future Directions

Malware (malicious software) is widely acknowledged as a major threat to trustworthy computing. There are a wide-variety of malware detection techniques in the literature. The state-of-the-art threat detection methods include machine learning models that identify malware based on patterns in software. While there are some promising efforts on hardware-assisted malware detection, there is no comprehensive study on efficient hardware trace analysis for malware detection. In this honors thesis, I investigated the impact of hardware traces on machine learning based malware detection. I generated hardware traces from a state-of-the-art System-on-Chip (SoC) board running three real-world malware benchmarks and several benign programs. Experimental results demonstrated that explainable machine learning based on hardware trace analysis can accurately differentiate between malware and benign programs. The idea proposed in this thesis can be extended to efficient and automated trace selection and minimization to make faster prediction without losing the accuracy. Moreover, future research can explore hardware performance counters as well as design-for-debug architecture (such as embedded trace buffer) to find profitable traces for effective malware detection using machine learning.

References

[1] Jack Wallen. Why the internet of things needs open source. https://www.techrepublic.com/article/why-the-internet-of-things-needs-open-source/.

[2] Cyber crime: Major cases. https://www.fbi.gov/investigate/cyber/major-cases.

[3] Jaikumar Vijayan. Target breach happened because of a basic network segmentation error. https://www.computerworld.com/article/2487425/target-breach-happened-because-of-a-basic-network-segmentation-error.html, 2014.

[4] Kevin McCoy. Target to pay $18.5m for 2013 data breach that affected 41 million consumers. https://www.usatoday.com/story/money/2017/05/23/target-pay-185m-2013-data-breach-affected-consumers/102063932/, 2017.

[5] For enterprises, malware is the most expensive type of attack. https://www.helpnetsecurity.com/2019/03/07/cyberattack-cost-2018/, 2019.

[6] Larry Ponemon Kelly Bissell. The cost of cybercrime. https://www.accenture.com/t20190305T185301Z__w__/us-en/_acnmedia/PDF-96/Accenture-2019-Cost-of-Cybercrime-Study-Final.pdf#zoom=50, 2019.

[7] Cryptocurrency exchanges lost nearly $170 million in 2019 and they're still vulnerable. https://m.dailyhunt.in/news/india/english/ambcrypto-epaper-ambcrypt/cryptocurrency+exchanges+lost+nearly+170+million+in+2019+and+they+re+still+vulnerable-newsid-153300584.

[8] Joanne Lim. An engineering disaster: Therac-25. https://www.bowdoin.edu/~allen/courses/cs260/readings/therac.pdf, 2018.

[9] David Schaper. Boeing pilots detected 737 max flight control glitch 2 years before deadly crash. https://www.npr.org/2019/10/18/771451904/boeing-pilots-detected-737-max-flight -control-glitch-two-years-before-deadly-cra.

[10] Alexander Polyakov. Machine learning for cybersecurity 101. https://towardsdatascience.com/machine-learning-for-cybersecurity-101-7822b802790b, 2018.

[11] Nwokedi Idika and Aditya Mathur. A survey of malware detection techniques. *Purdue University*, 03 2007.

[12] A. Moser, C. Kruegel, and E. Kirda. Limits of static analysis for malware detection. pages 421–430, Dec 2007.

[13] I. You and K. Yim. Malware obfuscation techniques: A brief survey. In *International Conference on Broadband and Wireless Computing, Communication and Applications*. IEEE Computer Society, 2010.

[14] Suman Jana and Vitaly Shmatikov. Abusing file processing in malware detectors for fun and profit. In *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA*, pages 80–94, 2012.

[15] Grégoire Jacob, Hervé Debar, and Eric Filiol. Behavioral detection of malware: from a survey towards an established taxonomy. *Journal in Computer Virology*, 4(3):251–266, 2008.

[16] Effective and efficient malware detection at the end host. In *18th USENIX Security Symposium (USENIX Security 09)*, Montreal, Quebec, August 2009.

[17] John Demme, Matthew Maycock, Jared Schmitz, Adrian Tang, Adam Waksman, Simha Sethumadhavan, and Salvatore J. Stolfo. On the feasibility of online malware detection with performance counters. In *The 40th Annual International Symposium on Computer Architecture, ISCA'13, Tel-Aviv, Israel, June 23-27, 2013*, pages 559–570, 2013.

[18] Mikhail Kazdagli, Vijay Janapa Reddi, and Mohit Tiwari. Quantifying and improving the efficiency of hardware-based mobile malware detectors. In *49th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2016, Taipei, Taiwan, October 15-19, 2016*, pages 37:1–37:13, 2016.

[19] Xueyang Wang and Ramesh Karri. Numchecker: detecting kernel control-flow modifying rootkits by using hardware performance counters. In *The 50th Annual Design Automation Conference 2013, DAC '13, Austin, TX, USA, May 29 - June 07, 2013*, pages 79:1–79:7, 2013.

[20] Kanad Basu, Rana Elnaggar, Krishnendu Chakrabarty, and Ramesh Karri. PREEMPT: preempting malware by examining embedded processor traces. In *Design Automation Conference*, 2019.

[21] Xilinx. *ZC702 Evaluation Board for the Zynq-7000 XC7Z020 SoC User Guide*.

[22] Kishore Angrishi. Turning internet of things(iot) into internet of vulnerabilities (iov) : Iot botnets. *CoRR*, abs/1702.03681, 2017.

[23] Linux.pnscan trojan is back to compromise routers and install backdoors. https://securityaffairs.co/wordpress/50607/malware/linux-pnscan-return.html.

[24] What is a ddos botnet? https://www.cloudflare.com/learning/ddos/what-is-a-ddos-botnet/.

[25] Andrew Shoemaker. How to identify a mirai-style ddos attack. https://www.imperva.com/blog/how-to-identify-a-mirai-style-ddos-attack/.