Utilization of Impedance Disparity Incurred from Switching Activities to Monitor and Characterize Firmware Activities

Md Sadik Awal
ECE Department
Florida International University
Miami, Florida, USA
mawal003@fiu.edu

Christopher Thompson
ECE Department
Weber State University
Ogden, Utah, USA
cthompson2@mail.weber.edu

Md Tauhidur Rahman
ECE Department
Florida International University
Miami, Florida, USA
mdtrahma@fiu.edu

2022 IEEE Physical Assurance and Inspection of Electronics (PAINE) | 979-8-3503-9909-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/PAINE56030.2022.10014924

***Abstract*—The massive trend toward embedded systems introduces new security threats to prevent. Malicious firmware makes it easier to launch cyberattacks against embedded systems. Systems infected with malicious firmware maintain the appearance of normal firmware operation but execute undesirable activities, which is usually a security risk. Traditionally, cybercriminals use malicious firmware to develop possible back-doors for future attacks. Due to the restricted resources of embedded systems, it is difficult to thwart these attacks using the majority of contemporary standard security protocols. In addition, monitoring the firmware operations using existing side channels from outside the processing unit, such as electromagnetic radiation, necessitates a complicated hardware configuration and in-depth technical understanding. In this paper, we propose a physical side channel that is formed by detecting the overall impedance changes induced by the firmware actions of a central processing unit. To demonstrate how this side channel can be exploited for detecting firmware activities, we experimentally validate it using impedance measurements to distinguish between distinct firmware operations with an accuracy of greater than 90%. These findings are the product of classifiers that are trained via machine learning. The implementation of our proposed methodology also leaves room for the use of hardware authentication.**

***Index Terms*—Hardware security, firmware activity, malware, reflection coefficient, switching activity, impedance difference**

I. Introduction

Current estimates of the number of internet-connected devices speculate that there will be over 40 billion smart devices in 2025, with more than 30 billion of those being Internet of Things (IoT) devices [1]. This is a major increase from the total and distribution of devices in 2015—of the 13.3 billion internet-connected devices in 2015, only 3.6 billion were considered IoT. The underlying reason for this drastic increase in devices is attributed to the popularization of personal smart devices, home assistants, and the development of low-power wide-area networks. With this surge in linked devices, the possibility of cyberattacks increases. One of the most severe threats to IoT devices is a firmware attack. Recent work [2], which encompasses the attack surfaces found in IoT devices, describes how firmware-based attacks can be leveraged for control hijacking, reverse engineering to obtain sensitive data, eavesdropping on sensitive packets, and creating system vulnerabilities to insert malware. Even if the firmware cannot be reverse-engineered, firmware updates can be exploited to distribute malware [3].

Firmware is a form of embedded software that supports fundamental device functionalities [4]. To begin with, when a device is turned on, the firmware is the first one to run and send the necessary instructions for the device to communicate with other devices or function properly. Without it, even the most basic of devices will be rendered inoperable. To prevent this from happening, the firmware is typically stored on an Erasable Programmable Read Only Memory (EPROM) or flash memory chip. But with technological advancement, firmware becomes obsolete even before the hardware does, as it needs to be updated to improve security, add new features, address issues, and support new protocols and standards. It is possible to exploit this flexibility, however, to allow malicious firmware updates [5].

Malicious firmware and hardware components create an unacceptable security channel on embedded systems. According to [6], firmware patches are capable of thwarting potential threats. This solution is appropriate only when the manufacturer is aware of the problem. It would be great if the system could detect and respond to threats on its own. However, embedded systems typically lack the same anti-malware protection measures as a standard personal computer. Since the embedded central processing unit (CPU) only has a limited number of resources, it is difficult to apply the majority of the standard security strategies. For instance, the authors of [7] modify the operating system to prevent USB-based attacks with direct human supervision, in which a USB stick is used to inject malicious code into secured device firmware in order to covertly compromise the system's confidentiality and reliability. The underlying cause of the issue is the implicit assumption that all extraterrestrial hardware is inherently trustworthy. Another work in [8] uses a remote software-based technique to validate the integrity of the peripherals linked to the system. Similarly, in the paper [9], software-based attestation is introduced to authenticate the software

of a device. Here, the complete contents of the memory are checked using a checksum function. Therefore, a less resource-intensive solution that can monitor the firmware's performance in order to detect any anomalies is necessary.

In this paper, we present a side-channel technique for observing the activities performed by an embedded system. This is accomplished by measuring the effective impedance of the microcontroller unit (MCU) across a range of frequencies while it is operating. As the MCU executes different instructions, the effective impedance of the MCU varies. This is the foundation of our strategy for detecting various firmware operations. After that, we classify the type of firmware that is operating on the system by utilizing several machine learning methods.

II. Related Works

Previous works discuss and show the feasibility of detecting firmware activities through side channel analysis. The authors of [10] demonstrate a work to classify the operating system, distinguish which software was running and differentiate different malware types running on a single board computer through EM emissions and an RF probe with near perfect accuracy. Similarly, authors of [11] propose a method for identifying malware and classifying it while it is executing on a Raspberry Pi. Their method uses comparable methodologies and achieves results that are comparable to those obtained by the authors of [10]. The authors of the paper [12] detect single instruction malicious code injections in the firmware of an Arduino Mega with collected EM emissions. This collected data was passed through a k-Nearest Neighbor model, where they were able to classify single-instruction injections. In [13], a framework for detecting anomalous code executions using a combination of machine learning and statistical training techniques is presented.

The impedance-based side-channel information has been used to detect malicious modifications of hardware [14]–[18]. The authors of [14] use a vector network analyzer (VNA) to detect hardware Trojans, recycled printed circuit boards (PCBs), malicious components, and counterfeit processors. The underlying method to detect hardware anomalies on a board relies on measuring the equivalent impedance from various locations on the board. Their process of finding hardware anomalies used the Fréchet between a standard circuit frequency and an anomalous frequency response. The authors of [15] implement a hardware modification detection method by modeling the circuit as a resistor-capacitor circuit. The basis for the anomaly detection method relies on supplying the circuit with an AC voltage and measuring the current. Capacitance measurements are used to detect malicious hardware modification by comparing the modified device's reactance to a standard. Authors of [16] detect and classify certain types of physical tampering events employing a wide-band antenna as a nearfield probe. The concept is founded on the premise that changes in the radio channel affect the frequency characteristics of an antenna. The work is backed by both a theoretical foundation and empirical validation. The authors in [17] explore the frequency region in which these complementary metal-oxide semiconductor (CMOS) gates emit unintentional side channel information. As current travels through the various metallic traces in the chip, the tiny distances between these metal etchings cause electromagnetic radiation to be unintentionally emitted, which suggests a range of frequencies to be monitored to detect the firmware running on the embedded system. The side channel-based analysis of [18] establishes a relationship between active mode current and maximum operational frequency. The authors show that this relationship can be used to detect hardware Trojans.

In this article, we will present our proposal, which is based on the impedance response of the processing unit and may be used to identify and classify different sorts of firmware activities.

III. Theoretical Analysis

In this section, we discuss the theory that underpins our proposed method for evaluating the viability of detecting firmware activities using a VNA. We propose that this can be accomplished by identifying the change in effective impedance caused by the switching activity arising from the firmware operations of a microcontroller.

The firmware of a device is a specialized type of computer software that offers low-level control for the device's specific hardware in an embedded system. It contains instructions to assist the hardware start-up, communicate with other devices, and execute basic input/output functions. The instructions are executed on the system's hardware, which contains thousands, if not millions, of logic circuits. By switching these logic circuits between their on and off states, instructions are executed.

The architecture of the logic gates in integrated chips relies on CMOS circuits. CMOS inverter/NOT gate, one of the basic universal logic gates, is a fundamental building block in digital logic circuits. Digital logic circuits compute the tasks by regulating the metal–oxide–semiconductor field-effect transistors (MOSFETs) of the CMOS circuits. These MOSFETs are activated by applying a voltage to the gate, which regulates the drain current. A thin silicon oxide layer isolates the gate of a MOSFET from the drain and source, and by inverting the substrate between the drain and source, a parasitic diode is formed. In addition to exhibiting resistance, MOSFETs are characterized by the presence of capacitance between their terminals. This property is a direct consequence of the structure of MOSFETs. The capacitance of the gate oxide film determines the capacitance of the gate to drain and gate to source terminals. Parasitic diode junction capacitance determines drain to source capacitance.

Fig. 1a depicts an equivalent circuit of CMOS containing parasitic capacitances. In this example, $C_{gdp/n}$ represents the gate to drain capacitance, $C_{dbp/n}$ represents the drain to bulk parasitic capacitance (the diffusion capacitance) for the PMOS/NMOS, and CY represents the capacitance at the output wire [19], [20]. The expressions of the capacitance are presented in Table I. The parameters associated with Table I and their definitions are presented in Table II.

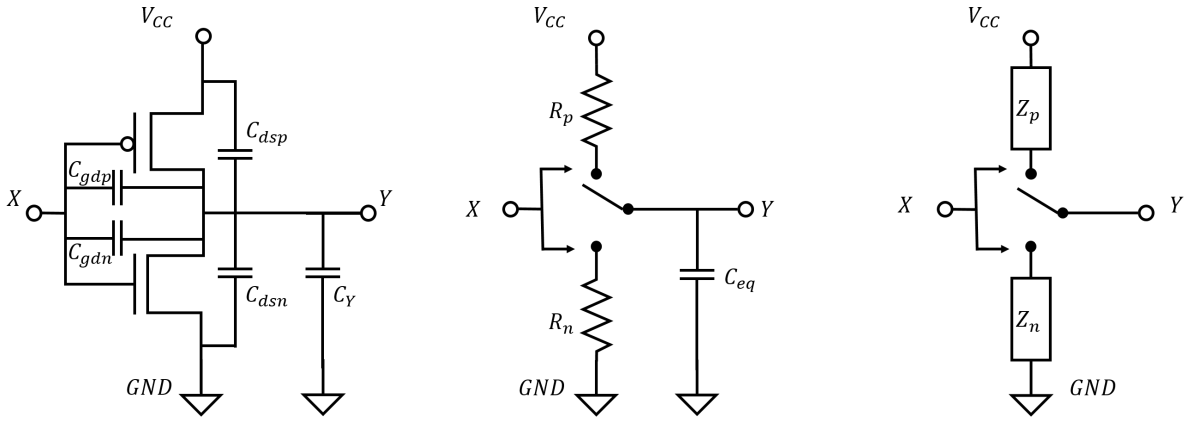

Fig. 1: CMOS inverter (a) with the parasitic capacitances, (b) switch model of dynamic behavior, (c) equivalent impedance circuit.

The output Y of the CMOS inverter varies based on the input X during the steady state. As shown in Fig. 1b, R_p and R_n are the effective on-resistance values for the PMOS and NMOS, respectively. Here, C_{eq} represents the output capacitance, which is the aggregate of all parasitic capacitance in the CMOS inverter. Hence, $C_{eq} = \sum(C_{gdp}, C_{gdn}, C_{dbp}, C_{dbp})$. Let $R_{lin,p/n}$ and $R_{sat,p/n}$ represent the effective on-resistance in the linear and saturation regions, respectively, for the PMOS/NMOS, [21], where,

$$R_{lin,p/n} = \frac{\frac{1}{2}(V_D - V_S - V_{t,p/n})}{\frac{3}{8}k'_{p/n}(\frac{W}{L})_{p/n}(V_D - V_S - V_{t,p/n})^2} \quad (1)$$

$$R_{sat,p/n} = \frac{V_D - V_S}{\frac{1}{2}k'_{p/n}(\frac{W}{L})_{p/n}(V_D - V_S - V_{t,p/n})^2} \quad (2)$$

Here, $k'_{p/n}$ represents the trans-conductance, $(W/L)_{p/n}$ represents the aspect ratio, and $V_{t,p/n}$ represents the threshold voltage of the PMOS/NMOS. The voltage at the drain is denoted by V_D, while the voltage at the source is denoted by V_S. Using (1) and (2), R_p and R_n can be estimated as follows:

$$R_{p/n} = \frac{1}{2}(R_{lin,p/n} + R_{sat,p/n}) \quad (3)$$

Each of the equivalent impedances, Z_p and Z_n in Fig. 1c, consists of equivalent resistance $R_{p/n}$ and equivalent reactance $X_{p/n}$. Therefore, $Z_{p/n} = R_{p/n} + jX_{p/n}$, where $R_{p/n}$ is the resistance and $X_{p/n}$ is the reactance of the PMOS/NMOS. The combined parasitic capacitance of the PMOS/NMOS dominates the equivalent reactance $X_{p/n}$ [19]. Therefore, $X_{p/n} \approx \frac{-1}{\omega C_{eq,p/n}}$, where $C_{eq,p/n}$ is the total equivalent capacitance of the PMOS/NMOS between the output node Y and the ground node GND.

The output terminal Y is connected to either the impedance Z_p or Z_n depending on the gate input X. Due to process variation and MOSFET geometry, the impedance of the MOSFETs differs from each other. As a consequence, the CMOS inputs govern the impedance measured between the source voltage node V_{CC} and the ground voltage node GND. Other logic circuits in the CPU exhibit similar input/output impedance shifting characteristics due to switching as well.

As the CPU executes its instruction set, inputs are applied to the gate terminals of logic gates in order to perform operations. In response to changes in the inputs, the impedance between the source and ground nodes of logic gates alters. Therefore, it

TABLE I: Expression of the parasitic capacitance

Capacitor	Expression
C_{gdp}	$2C_{op}W_p$
C_{gdn}	$2C_{on}W_n$
C_{dsp}	$K_{bpp}AD_pCJ_p + K_{swp}PD_pCJSW_p$
C_{dsn}	$K_{bpn}AD_nCJ_n + K_{swn}PD_nCJSW_n$
C_Y	From Extraction

TABLE II: Definitions of the parameters of parasitic capacitance

Parameter Symbol	Definition
C_{op}, C_{on}	Overlap capacitance per unit width
$W_{p/n}$	Width of PMOS/NMOS
$K_{bpp/n}$	Capacitor linearization factor of bottom plate
$AD_{p/n}$	Area of drain
$CJ_{p/n}$	Bottom junction capacitance
$K_{swp/n}$	Capacitor linearization factor of sidewall
$PD_{p/n}$	Perimeter of drain
$CJSW_{p/n}$	Sidewall junction capacitance

is to be expected that the overall impedance that exists between the source node and the ground node would shift in response to the various operations carried out by the CPU. In other words, the switching activity of the CMOS results in a change in the impedance between the source and ground.

To summarize, we conclude that as the CPU changes the output states of the logic gates, the size of the overall effective impedance between the source and ground node should change depending on the task it is performing. Therefore, distinct impedance signatures should be generated by distinct types of firmware activities.

IV. Experimental Setup

The goal of the experimental set-up is to evaluate the feasibility of the technique that we have proposed for monitoring the activities of a CPU. The Arduino Due [1], a VNA [23], and a breadboard with some LEDs and resistors to limit the current drain through the LEDs form up the infrastructure that is used in the study to collect data. A 5V power supply is used to power the Arduino Due. Afterwards, a capacitor is used with the probe to connect the VNA to the 3.3V pin of the microcontroller so that the observation can be recorded. Since the Arduino Due operates on 3.3V, monitoring the virtual impedance of the CPU while the firmware is being executed can be performed by sampling the 3.3V pin.

The VNA has a measuring range of 500 kHz to 4 GHz. We use the VNA to measure the forward reflection coefficient, a parameter that characterizes the amount of reflected wave in the transmission medium, to calculate the impedance at 10,000 linearly divided frequency points throughout its spectral region. To eliminate the DC component from measured signals, we put a capacitor in series. This data collected by the VNA is subsequently processed for classification analysis. Fig. 2 depicts a diagram that describes the experimental setup. The VNA is used to perform the measurements. A computer stores the measured signals and uploads different firmware to the Arduino microcontroller.

For the purpose of developing firmware operations, we compose C/C++ code [2] in the Arduino IDE and upload it to the MCU. This experiment considers the following four possible firmware operations:

1) Case 1: In this instance, the MCU is powered on but no operation is being performed.
2) Case 2: Three resistors and LEDs are connected to the digital pins of the Arduino and switched on and off at a rate of 1 kHz. These pins, as well as eight other digital I/O pins, are connected with additional resistors to ground. The purpose of this is to imitate an operation for the microcontroller that draws the total maximum allowable I/O current from all of the I/O pins (130 mA).
3) Case 3: In this instance, the MCU executes two distinct programs. The first program periodically turns an LED on and off at 1 Hz. In the background, as the MCU performs this operation, another program exponentiates random four-digit numbers. These numerals are raised to powers of one through three hundred. This latter program replicates possibly dangerous firmware operating in the background of the system, and is similar to the device acting as a botnet host. Essentially, the system preserves the illusion of regular operation but executes unwanted activities. This is akin to the system being compromised by the Mirai or BotenaGo malware [24]. The BigNumber Arduino Library [25] is used to reliably store and handle 256-bit values in order to manipulate these numbers.
4) Case 4: The Advanced Encryption Standard (AES) is utilized to replicate the situation in which a MCU is executing a sensitive task. The Arduino Cryptography Library [26] is used to encrypt and decrypt strings of ten ASCII characters that are generated at random.

Using these four scripts, we collect a total of 445 observations per case, for a grand total of 1,780 observations. Each observation contains the forward reflection coefficients at the most relevant and dominating frequency points extending from 500 kHz to 4 GHz (frequency range of the VNA) in the collected dataset. The step-by-step process for locating those frequency locations is described in greater detail in V-A. These frequency points help us to reduce the amount of redundant information in the dataset.

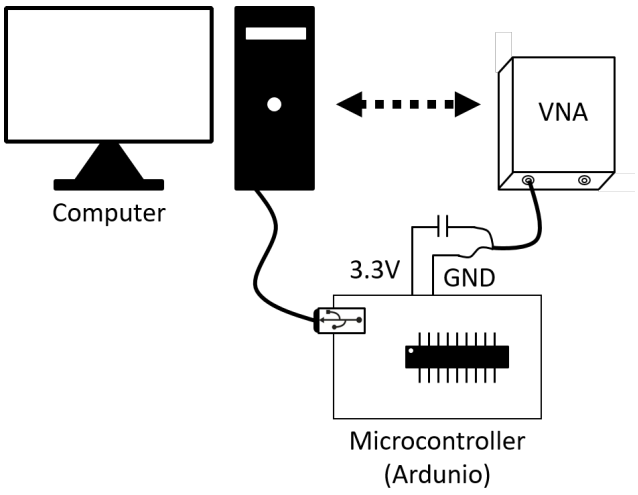

Fig. 2: Data collection from the microcontroller with a VNA.

Let τ represent the reflection coefficient, Z_r represent the reference impedance, and Z_τ represent the impedance obtained from the reflection coefficient of the medium. Eq. (4) describes the relation between Z_τ and τ.

$$Z_\tau = Z_r \left(\frac{1+\tau}{1-\tau} \right) \tag{4}$$

[1]The Arduino Due [22] is powered by the AT91SAM3X8E microcontroller with 96 KB of SRAM and 3.3V of operational voltage. It operates at 84 MHz and has a total DC output current of 130 mA across all of its I/O lines

[2]The firmware activity codes for this experiment are accessible at https://github.com/ChristopherThompsonUT/ArduinoRepo

To compute the impedance, we use the recorded reflection coefficients and eqrefeq:impedance. The reference impedance, Zr, for the VNA in this case is 50 Ω. We employ impedance measurements generated from reflection coefficients in the classification stage.

V. Result and Analysis

This section begins by describing our analysis approach for detecting firmware activity from a dataset. Following this is an overview of what we studied. Fig. 3 illustrates the categorization process in a simplified form. In order to make use of any machine learning model, an appropriate dataset is required that is composed of the fingerprints of the activities carried out by the firmware. The steps involved in this process are outlined in the subsequent subsections.

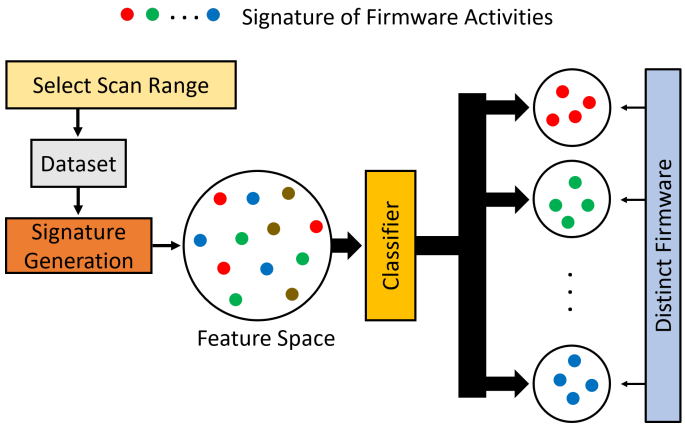

Fig. 3: Steps in firmware activity detection and classification

A. Selecting Scanning Region

The VNA in our experimental setup scans 10,000 frequency points that are linearly split and range from 500 kHz to 4 GHz in frequency. This section focuses on selecting the appropriate frequencies to investigate for signatures generated from various firmware activities. We desire to find the optimal frequency points in order to reduce the computation time and increase the number of relevant sample points. The procedure is depicted graphically in Fig. 4.

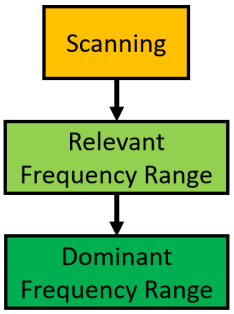

Fig. 4: Selection of appropriate frequency points

To investigate the frequency points, we collect the responses of 10,000 of them while performing various tasks on the microcontroller. A subset of about 20% of the total frequency points is then investigated to determine which ones are most strongly connected to various firmware operations. This allows us to determine the frequencies that are relevant. In order to do this, we compute the Pearson correlation coefficients between the responses of each frequency point and specific firmware activity. On the basis of the correlation coefficients, the first 1968 frequency points (about 20% of all frequencies points) that are highly correlated with firmware activities are selected. These 1968 frequency points also reflect the 70% correlation coefficients for the correlation coefficient with the highest value. After that, we select the maximum number of frequencies that are linearly independent of one another in order to evaluate the dominant frequency points. This allows us to reduce the number by an additional factor. In this step, we once again utilize the Pearson correlation coefficients between the frequencies. These subsequently lowered frequency groups exhibit correlation values of less than 90%. Following this two-step process of frequency selection, we are able to identify 338 frequency points that adequately describe the activities of the firmware without significantly compromising accuracy. These 338 frequency points represent only a 3.38% of the frequency range of the VNA. During the data collection process with the VNA, we concentrate on only these specific frequency points.

B. Feature Selection and Classification

We train and test the classifier models with the help of the MATLAB Statistics and Machine Learning Toolbox. As discussed in the preceding subsection, we construct two distinct datasets, one for the training and the other one for the testing, utilizing the 338 frequency points that are the most significant and dominant in explaining the actions of the firmware. One of these datasets is used for training, and the other for testing.

As depicted in Fig. 3, we first use the training dataset to produce signatures from various firmware actions in order to deduce any meaningful use from the dataset. In order to get this collection of signatures prepared for the machine learning (ML) classifier, we run a principal component analysis (PCA) [27] on the training data to determine which features are accountable for 95% of the variance in the data. This allows us to prepare the signatures for the ML classifier. To reduce training time and improve the classifier's accuracy, it is desirable to select a subset of the available features. PCA facilitates the generation of a reduced collection of features without compromising the variance of the data. Using PCA, we can explain at least 95% of the variation in the training dataset using only 34 principal components. In the training phase of the machine learning classifier, these 34 principal components serve as features. To prevent overtraining and to stabilize the models, classifiers are trained using five-fold cross-validation. For the purposes of validating the models, we use the testing dataset, which comprises 30% of the total collected data.

TABLE III: Comparison of accuracy and precision metric

Classifier	F1 Score (Train)	F1 Score (Test)	Precision	Recall	Specificity	Accuracy
SVM (Kernel: Gaussian) [28]	92.4%	93.7%	93.9%	93.6%	97.9%	93.6%
SVM (Kernel: Cubic) [29]	92.1%	93.3%	93.7%	93.2%	97.7%	93.3%
SVM (Kernel: Quadratic) [30]	92.4%	91.8%	92.2%	91.7%	97.2%	91.8%
Quadratic Discriminant [31]	92.1%	91.5%	91.7%	91.4%	97.1%	91.4%
Subspace KNN [32]	91%	90.9%	91.2%	90.8%	96.9%	90.8%

To evaluate the performance of the ML classifiers, we calculate the precision, recall, specificity, accuracy and F1 scores [33], [34]. Let TP_i, TN_i, FP_i, and FN_i be the true positives, true negatives, false positives, and false negatives predicted by a given classifier for prediction class i, respectively. In this context, a true positive is the number of correctly predicted positive values, whereas a false positive is the number of inaccurately predicted positive values. Additionally, a false negative is the number of values incorrectly predicted as negative, whereas a true negative is the number of correctly predicted negative values. Precision measures the proportion of positive class predictions that correspond to the actual positive class. Thus, precision evaluates the accuracy for the minority class and is calculated as [33],

$$Precision = \frac{1}{i}\sum_{i}\left(\frac{TP_i}{TP_i + FP_i}\right) \tag{5}$$

Recall quantifies the number of correct class predictions generated for all positive examples in the training set [33]. The recall indicates that positive forecasts were missed.

$$Recall = \frac{1}{i}\sum_{i}\left(\frac{TP_i}{TP_i + FN_i}\right) \tag{6}$$

The specificity of a classifier is the ratio between the amount of data that is accurately classified as negative and the actual amount of data that is negative [33].

$$Specificity = \frac{1}{i}\sum_{i}\left(\frac{TN_i}{TN_i + FP_i}\right) \tag{7}$$

The accuracy of a dataset's predictions is the ratio of correct predictions to the total number of predictions. The accuracy increases as the number of correct predictions made by the system increases [34].

$$Accuracy = \frac{1}{i}\sum_{i}\left(\frac{TP_i + TN_i}{TP_i + TN_i + FP_i + FN_i}\right) \tag{8}$$

The F1 score provides a single value that addresses both precision and recall concerns in a single number. This is the mean harmonic of the two fractions [34]. A classifier's performance improves as its F1 score increases.

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{9}$$

We use (5)-(9) to calculate the performance metrics and compare the performance of the classifiers. Table III illustrates the F1 score, precision, recall, specificity, and accuracy for five ML classifiers. The classifiers perform very well and classify firmware activities with an F1 score, precision, recall, specificity, and accuracy of more than 90%, indicating that our proposed method to detect firmware operations is statistically viable.

VI. Conclusion and Future Work

In this study, we demonstrate that a VNA is capable of detecting aberrant activity in digital logic systems. This is accomplished by monitoring the effective impedance of an Arduino Due, measuring the values of the forward reflection coefficients using a VNA over a range of significant frequencies, extracting features from these, and passing these feature sets to various machine learning models. The top-performing classifier is able to determine which firmware is currently being executed by the system by making use of 34 features, achieving an accuracy of 93.2% during training and 94.8% during testing, respectively.

In the future, we plan to apply more feature extraction techniques to determine if this can potentially improve the classifiers' accuracy. In addition, we intend to determine if our proposed method may be utilized in reverse engineering to recover critical hardware and algorithmic data. Since we can discover firmware anomalies in this way, we can investigate hardware Trojans. Lastly, our proposed strategy may also investigate how injected components influence the computing processes of embedded systems.

Acknowledgment

This work was supported partly by the National Science Foundation under Grant Nos. 2150248, 2214108, and 2114200.

References

[1] "State of the IoT 2020: 12 billion IoT connections, surpassing non-IoT for the first time," https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/, Nov 2021, (Accessed: 3 September 2022).

[2] S. Schmidt, M. Tausig, M. Hudler, and G. Simhandl, "Secure firmware update over the air in the internet of things focusing on flexibility and feasibility," in *Internet of Things Software Update Workshop (IoTSU). Proceeding*, 2016.

[3] H. A. Abdul-Ghani, D. Konstantas, and M. Mahyoub, "A comprehensive IoT attacks survey based on a building-blocked reference model," *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 3, 2018.

[4] C. J. Tan, J. Mohamad-Saleh, K. A. M. Zain, and Z. A. A. Aziz, "Review on firmware," in *ICISPC 2017*, 2017.

[5] A. Cui, M. Costello, and S. Stolfo, "When firmware modifications attack: A case study of embedded exploitation," 2013.

[6] O. Caspi, "AT&T Alien Labs finds new golang malware (BotenaGo) targeting millions of routers and IoT devices with more than 30 exploits," https://cybersecurity.att.com/blogs/labs-research/att-alien-labs-finds-new-golang-malwarebotenago-targeting-millions-of-routers-and-iot-devices-with-more-than-30-exploits, Nov 2021, (Accessed: 3 September 2022).

[7] D. J. Tian, A. Bates, and K. Butler, "Defending against malicious USB firmware with GoodUSB," in *Proceedings of the 31st Annual Computer Security Applications Conference*, 2015, pp. 261–270.

[8] Y. Li, J. M. McCune, and A. Perrig, "SBAP: software-based attestation for peripherals," in *International Conference on Trust and Trustworthy Computing*. Springer, 2010, pp. 16–29.

[9] A. Seshadri, A. Perrig, L. Van Doorn, and P. Khosla, "SWATT: software-based attestation for embedded devices," in *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*. IEEE, 2004, pp. 272–282.

[10] M. L. Wilt, M. M. Baker, and S. J. Papadakis, "Toward an RF side-channel reverse engineering tool," *2020 IEEE Physical Assurance and Inspection of Electronics (PAINE)*, 2020.

[11] D. P. Pham, D. Marion, M. Mastio, and A. Heuser, "Obfuscation revealed: Leveraging electromagnetic signals for obfuscated malware classification," *Annual Computer Security Applications Conference*, 2021.

[12] K. Vedros, G. M. Makrakis, C. Kolias, M. Xian, D. Barbara, and C. Rieger, "On the limits of em based detection of control logic injection attacks in noisy environments," *2021 Resilience Week (RWS)*, 2021.

[13] H. Agrawal, R. Chen, J. K. Hollingsworth, C. Hung, R. Izmailov, J. Koshy, J. Liberti, C. Mesterharm, J. Morman, T. Panagos, and et al., "CASPER: An efficient approach to detect anomalous code execution from unintended electronic device emissions," *Cyber Sensing 2018*, 2018.

[14] H. Zhu, H. Shan, D. Sullivan, X. Guo, Y. Jin, and X. Zhang, "PDNPulse: sensing PCB anomaly with the intrinsic power delivery network," *CoRR*, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2204.02482

[15] M. Nishizawa, K. Hasegawa, and N. Togawa, "Capacitance measurement of running hardware devices and its application to malicious modification detection," in *2018 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, 2018, pp. 362–365.

[16] M. S. Awal, A. Madanayake, and M. T. Rahman, "Nearfield RF sensing for feature-detection and algorithmic classification of tamper attacks," *IEEE Journal of Radio Frequency Identification*, vol. 6, pp. 490–499, 2022.

[17] D. Das, M. Nath, B. Chatterjee, S. Ghosh, and S. Sen, "STELLAR: A generic EM side-channel attack protection through ground-up root-cause analysis," in *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, Tysons, Virginia, 2019.

[18] S. Narasimhan, D. Du, R. S. Chakraborty, S. Paul, F. Wolff, C. Papachristou, K. Roy, and S. Bhunia, "Multiple-parameter side-channel analysis: A non-invasive hardware Trojan detection approach," in *2010 IEEE international symposium on hardware-oriented security and trust (HOST)*. IEEE, 2010, pp. 13–18.

[19] J. M. Rabaey, A. P. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits*. Prentice hall, 2002.

[20] A. S. Smith and K. C. Smith, *Microelectronic Circuits*, 5th ed. Oxford University Press, 2004.

[21] W. Wolf, *FPGA-based system design*. Pearson Education, 2004.

[22] "Arduino Due product page," https://docs.arduino.cc/hardware/due, (Accessed: 3 September 2022).

[23] "Portable VNA - Vector Network Analyzer," https://pocketvna.com/, (Accessed: 3 September 2022).

[24] I. Van der Elzen and J. van Heugten, "Techniques for detecting compromised IoT devices," *University of Amsterdam*, 2017.

[25] N. Gammon, "Bignumber," https://github.com/nickgammon/BigNumber, March 2019, (Accessed: 3 September 2022).

[26] Rweather, "Arduino cryptography library," https://github.com/rweather/arduinolibs, 2019, (Accessed: 3 September 2022).

[27] J. E. Jackson, *A user's guide to principal components*. John Wiley & Sons, 2005.

[28] W. Wang, Z. Xu, W. Lu, and X. Zhang, "Determination of the spread parameter in the Gaussian kernel for classification and regression," *Neurocomputing*, vol. 55, no. 3-4, pp. 643–663, 2003.

[29] A. R. Bagasta, Z. Rustam, J. Pandelaki, and W. A. Nugroho, "Comparison of cubic SVM with Gaussian SVM: classification of infarction for detecting ischemic stroke," in *IOP Conference Series: Materials Science and Engineering*, vol. 546, no. 5. IOP Publishing, 2019, p. 052016.

[30] A. Patle and D. S. Chouhan, "SVM kernel functions for classification," in *2013 International Conference on Advances in Technology and Engineering (ICATE)*. IEEE, 2013, pp. 1–9.

[31] A. Tharwat, "Linear vs. quadratic discriminant analysis classifier: a tutorial," *International Journal of Applied Pattern Recognition*, vol. 3, no. 2, pp. 145–180, 2016.

[32] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE transactions on pattern analysis and machine intelligence*, vol. 20, no. 8, pp. 832–844, 1998.

[33] D. M. Powers, "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation," *arXiv preprint arXiv:2010.16061*, 2020.

[34] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2011.