AFRL-RY-WP-TR-2021-0038

COMPUTATIONAL ACTIVITY MONITORING BY EXTERNALLY LEVERAGING INVOLUNTARY ANALOG SIGNALS (CAMELIA)

Alenka Zajic, Milos Prvulovic, and Alessandro Orso
Georgia Institute of Technology

Matthew Welborn
Northrop Grumman Information Systems

MAY 2021
Final Report

STINFO COPY

AIR FORCE RESEARCH LABORATORY
SENSORS DIRECTORATE
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7320
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with The Under Secretary of Defense memorandum dated 24 May 2010 and AFRL/DSO policy clarification email dated 13 January 2020. This report is available to the general public, including foreign nationals.

Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RY-WP-TR-2021-0038 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

*//Signature//

IGOR V. TERNOVSKIY
Program Manager
Decision Sciences Branch
Multi-Domain Sensing Autonomy Division

*//Signature//

OLGA L. MENDOZA-SCHROCK, Chief
Decision Sciences Branch
Multi-Domain Sensing Autonomy Division

*//Signature//

ROY L. BALLARD
Division Chief
Multi-Domain Sensing Autonomy Division
Sensors Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

*Disseminated copies will show "//Signature//" stamped or typed above the signature

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
May 2021	Final	26 May 2016 – 30 August 2020

4. TITLE AND SUBTITLE
COMPUTATIONAL ACTIVITY MONITORING BY EXTERNALLY LEVERAGING INVOLUNTARY ANALOG SIGNALS (CAMELIA)

5a. CONTRACT NUMBER
FA8650-16-C-7620

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
62303E

6. AUTHOR(S)
Alenka Zajic, Milos Prvulovic, and Alessandro Orso (Georgia Institute of Technology)
Matthew Welborn (Northrop Grumman Information Systems)

5d. PROJECT NUMBER
N/A

5e. TASK NUMBER
N/A

5f. WORK UNIT NUMBER
Y1H5

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Georgia Institute of Technology
85 5th Street NW
Atlanta, GA 30308

Northrop Grumman Information Systems
3005 Carrington Mill Parkway
Morrisville, NC 27560

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory
Sensors Directorate
Wright-Patterson Air Force Base, OH
45433-7320
Air Force Materiel Command
United States Air Force

Defense Advanced Research Projects Agency
(DARPA/I20)
675 North Randolph Street
Arlington, VA 22203

10. SPONSORING/MONITORING AGENCY ACRONYM(S)
AFRL/RYAT

11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S)
AFRL-RY-WP-TR-2021-0038

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with The Under Secretary of Defense memorandum dated 24 May 2010 and AFRL/DSO policy clarification email dated 13 January 2020. This report is available to the general public, including foreign nationals. This material is based on research sponsored by the Air Force Research Lab (AFRL) and the Defense Advanced Research Projects Agency (DARPA) under agreement number FA8650-16-C-7620. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Labs (AFRL), the Defense Advanced Research Projects Agency (DARPA) or the U.S. Government. Report contains color.

14. ABSTRACT
In this project, we have developed a system called CAMELIA that monitors computation in an EMSD (or phone/laptop/server) device by leveraging the involuntary electromagnetic (EM) emanations from the monitored device.

15. SUBJECT TERMS
Side-channels, analog side-channels, electromagnetic side-channels, program monitoring, malware dedection

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT:	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON (Monitor)
a. REPORT	b. ABSTRACT	c. THIS PAGE			Igor Ternovskiy
Unclassified	Unclassified	Unclassified	SAR	286	19b. TELEPHONE NUMBER *(Include Area Code)* N/A

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Page

1.0 SUMMARY

Security monitoring of embedded and mission-specific devices (EMSDs) is challenging as many of these devices cannot support the memory and performance overheads of traditional security monitoring. In fact, monitoring activity may interfere with the primary function of EMSDs. Moreover, many important classes of threats allow attackers to compromise/subvert both the protected functionality and the monitoring functionality of the system.

In this project, we have developed a system called CAMELIA that monitors computation in an EMSD (or phone/laptop/server) device by leveraging the involuntary electromagnetic (EM) emanations from the monitored device. CAMELIA does not require changes to the monitored device or its software, and its monitoring ability remains intact even after a complete compromise of the monitored system. CAMELIA collects signals using purpose-designed antennas, then pre-processes the signals and separates them into sub-channels that carry information about different aspects of the system's state. CAMELIA uses models of valid software behavior and software/system/hardware interactions to form hypothesis about the sequence of execution and software/system/hardware events in the monitored system, then updates these hypotheses by matching the expected signals for each hypothesis to the observed signals. This allows CAMELIA to maintain high accuracy and fidelity even when monitoring large codes, and even in the presence of interrupts, input/output activity, cache misses, branch miss-predictions, and other events that

change emanated EM signals significantly in a way that is seemingly random but that CAMELIA can account for end even use to improve monitoring.

To manage the tradeoff between fidelity, computational cost of modeling, and timeliness of reporting, CAMELIA operates at three levels of fidelity. More precisely, CAMELIA can (1) discover loop/module-level anomalies immediately, (2) detect basic-block-level control flow violations and anomalies at the granularity of several instructions very rapidly (after one or few dynamic instances of the violation are observed) and (3) uncover anomalous execution/event patterns and even "below noise level" problems (e.g., when a valid instruction is replaced by a similar instructions) after enough dynamic instances are observed.

The accuracy and fidelity of CAMELIA has been assessed not only against the goals listed in the DARPA LADS solicitation, but also against information-theoretic limits that have been derived and refined concurrently with the CAMELIA design.

CAMELIA has a potential to revolutionize security monitoring by eliminating the need for monitoring-related resources and mechanisms on the monitored device itself, and by preventing even a full compromise of the monitored device from compromising the monitoring functionality. We have evaluated CAMELIA at 1ft, 3ft, 10ft, and 600ft distance from the monitored systems for accuracy, fidelity, and reporting timeliness, relative to the targets specified in the DARPA LADS solicitation and also relative to the limits derived from the information-theoretic models. CAMELIA meets and exceeds all targets specified in DARPA LADS solicitation.

2.0 INTRODUCTION

Figure 1 illustrates the overall CAMELIA approach that uses the inadvertent EM emanations of the target system to wirelessly monitor its security. The real-time monitoring parts of CAMELIA are shown in green and describe tasks T1 and T2. The signals are received using a sophisticated purpose-designed antenna/probe array. Signals from different antenna/probe elements are then amplified and processed to identify from which direction the signal is coming, and beamforming and interference cancellation are applied to enhance the signal from the target device while suppressing other signals. The enhanced signal is then demodulated and separated by frequency (and possibly other characteristics) into sub-channels that correspond to different aspects of monitored activity (processor, memory, etc.). The signal is then subjected to real-time "spectral monitoring", where quick analyses are applied to the overall spectrum of the signal and to the most prominent features in the time-domain signal. The spectral monitoring relies on a coarse-grain (loop/module level) model of the monitored system's software to identify large deviations from the program's behavior (e.g., when the permissions control code executes right after string-copy code in a program where such a transition should not occur). The spectral monitoring also uses typical spectra (acquired during training) to identify deviations, e.g., when the time-per-iteration of a loop should be very consistent but the observed behavior contradicts

that. Spectral monitoring immediately reports major anomalies that correspond to the "known/unknown code" level of fidelity described in the DARPA LADS BAA., i.e. when the observed signal is very unlikely to occur during correct execution in the monitored system. It also identifies potential anomalies and triggers high-fidelity analysis of the corresponding signal from the signal buffer.

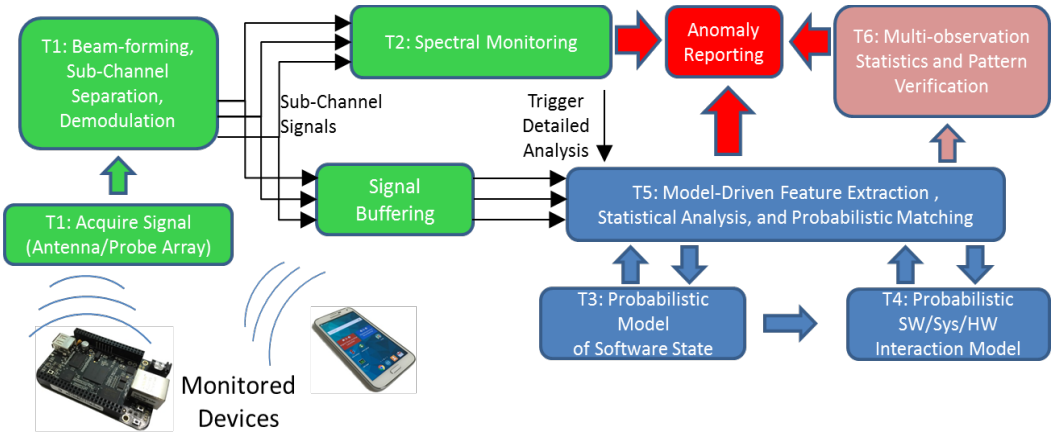


Figure 1 Camelia design

Shown in blue is the high-fidelity verification that corresponds to tasks T3, T4, and T5: basic-block-level control flow and confirmation of execution of individual instructions. It is highly accurate and practically feasible because it does not track execution of instructions and basic blocks in isolation. Instead, it uses a model of the program and hardware characteristics (both obtained during training) to continuously maintain a probabilistic model of the software state and software/system/hardware (SW/Sys/HW) interactions. The model of the software state, for example, keeps track of the probably/likely recent path through the program, and identifies the

possible/likely next-step execution, which allows the feature extraction and matching algorithm to limit its search space to only those that are possible, and the matching is performed in order of likelihood. For example, if the most probable "current" execution point is just before entering an if-then-else hammock, and the "else" side is more likely to be taken, the signal matching is first attempted against the signal template (from training) that corresponds to the "else" path through this hammock. The model of SW/sys/HW interactions allows the matching algorithms to recognize and account for significant events, such interrupts, I/O, cache misses, etc., that may significantly change the timing and signal shape. This eliminates the need to obtain training data for all possible combinations of events that may (but need not) occur in a particular part of the code. The interaction model is also informed by the software model, e.g., it allows the cache-miss-like part of the signal to be matched as a cache miss only if the program actually contains a memory access at that point in execution, and it monitors the occurrences of such events against expectations to allow detection anomalies, such as too many cache misses in the part of the code where cache misses should be rare. Like the spectral monitoring module, the high-fidelity verification module immediately reports anomalies for which it has enough confidence. But this module also forwards statistics of seemingly-normal and mildly-anomalous signal-to-execution matches to the multi-observation verifier.

This multi-observation verifier T6 (shown in pale red) maintains statistics across multiple dynamic occurrences for each point in the code and verifies that these statistics match expectations.

This allows it to detect small but persistent deviations from expected behavior, e.g., when a single instruction in the program has been replaced with a very similar one.

In the Section 2, we detail tasks T1-T6 and how are the implemented, and in Section 3 we present results and discuss outcomes.

3.0 METHODS, ASSUMPTIONS, AND PROCEDURES

3.1 Task 1.1 - Purpose-Designed Antenna Arrays

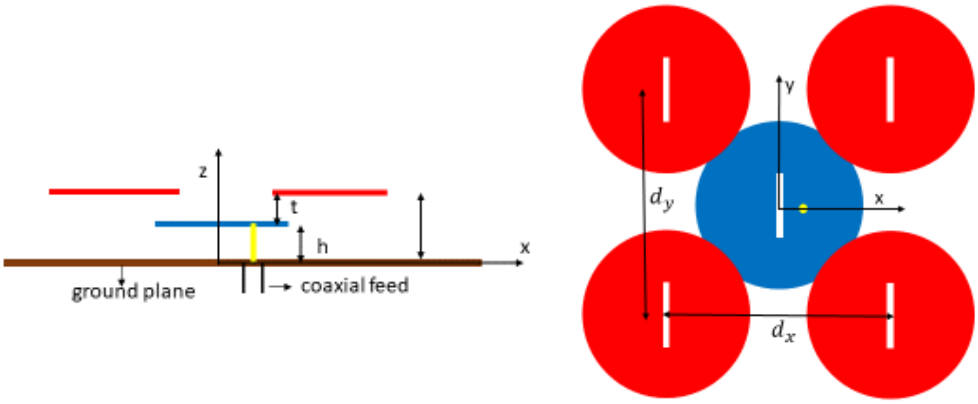
In this section, we describe a novel high gain antenna design specifically tailored for long range EM side-channel detection [1]. Scalability was a key factor so that the detection range can be extended in a straightforward manner by increasing the number of elements cost effectively. We have also designed helical antenna array for long range EM side-channel detection which is described in [2], but is omitted here for brevity.

3.1.1. Planar Antenna Array Design for Long Range EM Side-channel Detection

The proposed antenna consists of two layers of slotted conducting metal discs suspended on air and placed above the ground plane using teflon screws [1]. The circular discs are designed to operate in the higher order TM_{12} mode, this allows for each element to have higher directivity with a simpler feed network. The screws' locations correspond to the electric field nulls along the disc radius. The upper layer is 2×2 array of slotted circular discs electromagnetically coupled by a lower identical disc which is fed directly by a single coaxial feed. The complete fabrication of the antenna is done using aluminum sheets and involves no of dielectric substrate. The antenna has a peak gain of 19 dBi with impedance bandwidth ($S_{11} \leq -6$ dB) of 6.7%. The antenna is tested for the application of receiving unintentional EM emanations generated by one or multiple embedded,

"smart" electronic systems. Finally, the antenna was used to characterize the complex SNR behavior of EM emanation detection at a distance.

The antenna is a two layer stacked configuration as shown in Figure2 (a).



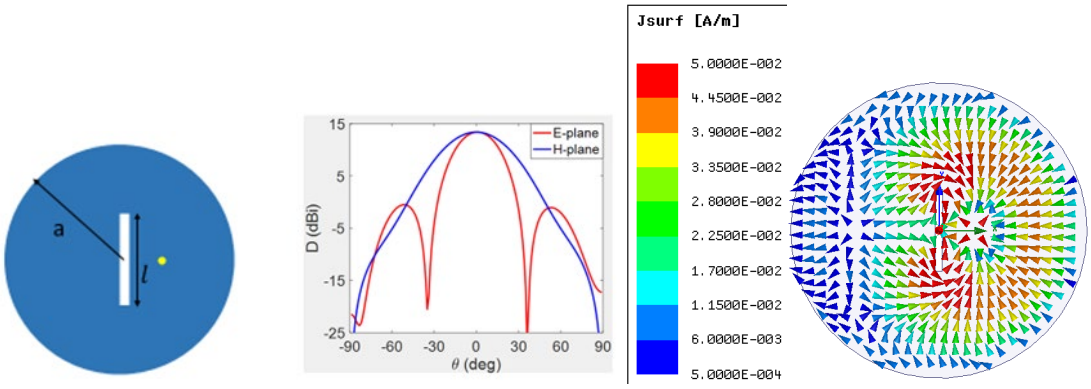
(a) (b)

Figure 2 Antenna geometry (a) side view and (b) top view

The upper layer is 2×2 array of slotted circular discs in TM_{12} mode, shown in Figure 2 (b), fed by an identical disc in the lower layer, which is directly fed by coaxial probe. A similar feeding technique was proposed in [3] where a 2×2 array of rectangular patches was excited by a microstrip fed and centrally located patch in the lower layer. This technique removes dependency on feed lines. Here, to avoid feed lines, we use coaxial feed to excite the lower disc. All circular discs have identical geometrical dimensions. The individual circular disc is loaded with narrow rectangular slot at the center. Slot loading is used to reduce the high sidelobes in the E-plane radiation pattern of TM_{12} mode, as explained and discussed in [4].

3.1.2. Design Procedure

The design procedure is described as follows. Based on the peak directivity requirements, the single element is designed first as shown in Figure 3 (a). In the present case, the slot length is selected for maximum directivity, which is 13.4 dBi. The corresponding disc radius and slot length, *l,* are 20.5 and 11.3 cm respectively. Since it is a narrow slot, the slot width, *w,* is selected to be 1 cm. The thickness, *h,* is chosen to be 5 mm. Higher thickness values will result in increase in Side-Lobe Level (SLL) of the element as explained in [4]. The directivity pattern of the single element in the E and H-plane and its current distribution is shown in the Figure 3 (b) and Figure 3 (c) respectively. The current density is higher in the region adjacent to slots compared to the other parts of the patch as the narrow slot at the center intercepts the flow lines of current and gets excited. This produces the out of phase electric field at the slot aperture, which leads to sidelobe cancellation as explained in detail in [4]. The 2×2 array of identical elements is then placed at the height *t* above this layer as shown in Fgiure 3 (a). In this case, the *t* is selected to be 5 mm. The array spacing d_x and d_y is chosen to reduce E and H-plane sidelobe and to improve impedance match ($S_{11} \leq -6$ dB). Additionally, the parameters of the center disc in the lower level can also be adjusted to improve the impedance match which also results in the reduced H-plane sidelobe.



(a) (b) (c)

Figure 3 Element design at 1.03GHz: (a) Slot loaded disc, (b) directivity pattern in E and H-plane, (c) the current distribution of the patch

To complete the design, we investigate the effect of array spacing and positioning of the center disc on the sidelobes in the radiation pattern. This is required since the element radius is ~ $0.7\lambda_0$ and hence the minimum array spacing will be greater than $1.4\lambda_0$. For the spacing greater than $1.4\lambda_0$, array theory predicts that the sidelobe will be high in the radiation pattern, which reduces the aperture efficiency and the directivity [5]. In antenna arrays, several methods have been used in the past for sidelobe suppression [6], [7]. In this design, the side lobe in the E-plane is suppressed by the slot loaded in the disc. In the H-plane the sidelobe is suppressed by the center disc.

3.1.3. Element Spacing

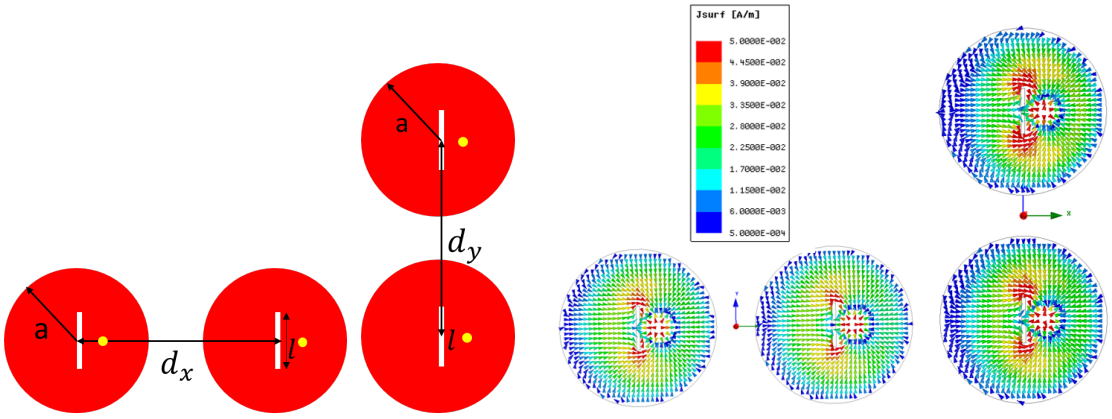
To explain how element spacing impacts the sidelobe, we investigate E & H-plane radiation pattern of 2×1 array and 2×2 array of the element shown in Figure 4 (a), assuming

infinite ground plane configuration. Figure 4 (a) & (b) shows the geometry of 2×1 and 1×2 array. Simulations were performed for the various spacing between array elements for both the geometries. Current distribution for both array geometries are shown in Figure 4 (c) & (d). For both elements, the excitation amplitudes are equal with zero phase difference. Current density scaling is the same as used in Figure 3 (c). It is observed that 2X1 array compared to 1X2 array has strong current density around the slot edges. The reason for this is the aperture field vector of the slot, which is in the direction of x-axis, as explained in [14], and hence can have possible coupling effects in the 2X1 configuration.

Figure 4 (a) & (b) shows the E & H-plane pattern for 2X1 array geometry with element spacing d_x as parameter. As d_x increases from 1.5 to $2\lambda_0$, the first sidelobe in the E-plane increases. For $1.5\lambda_0$, there is one lobe in the visible region while for 1.75 and $2\lambda_0$, there are two lobes. In all cases the minor lobes are 10 dB below the main beam. Figure 5 (b) shows that element spacing has negligible effect on the H-plane pattern.

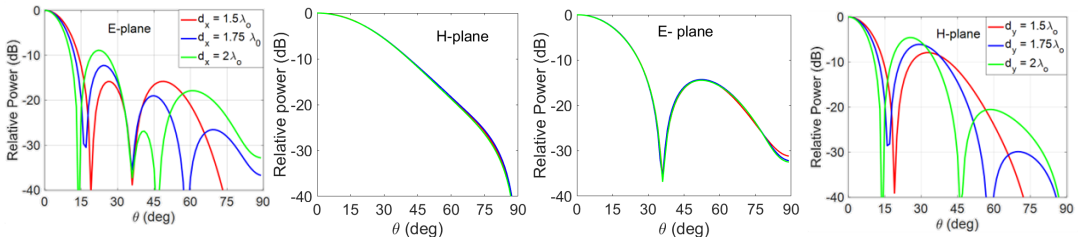
Figure 4 (c) & (d) shows the E & H-plane pattern for 1X2 array geometry shown in Figure 4 (b). Compared to Figure 4 (a), H-plane pattern shown in Figure 4 (d), has higher sidelobes since the sidelobe cancellation effect of slot is less dominant in the H-plane configuration. The first sidelobe is reduced by ~ 3dB in the E-plane of 2X1 array, due to cancellation effect by slot loading, as compared to the H-plane pattern of the 1X2 array. Based on this study of how the array spacing impacts radiation pattern, we chose the value of $1.75\lambda_0$. We have also observed in simulations that

the selected spacing has a good impedance match in the frequency band of interest. This is also shown in Figure 4 (a).



(a) (b) (c) (d)

Figure 4 Array geometry (a) 2X1 array (b) 1X2 array (c) current distribution of 2X1 array, (d) 1X2 array



(a) (b) (c) (d)

Figure 5 Radiation pattern as a function of array spacing (a) & (b) E & H-plane pattern for geometry shown in Figure 4 (a), (c) & (d) for the geometry shown in Figure 4 (b)

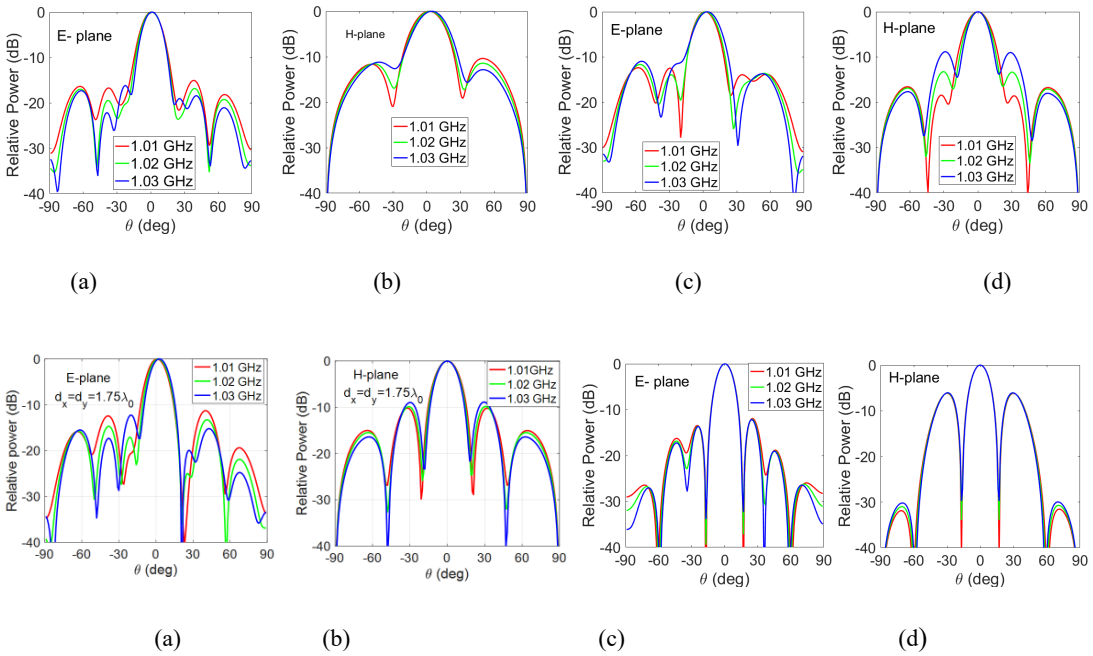
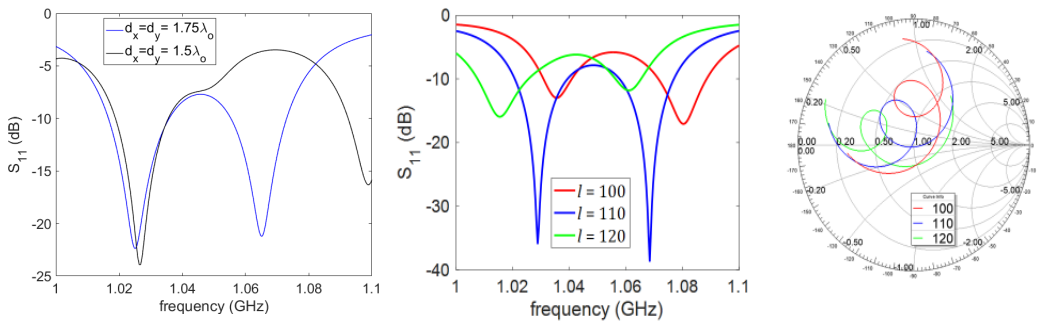


Figure 6 Effect of the center disc on the radiation pattern in (a) 2X1 array E-plane, (b) 2X1 H-plane, (c) 1X2 E-plane, (d) 1X2 H-plane, (e) 2X2 E-plane, (f) 2X2 H-plane, (g) 2X2 array without lower center disc, E-plane, (h) 2X2 array without lower center disc, H-plane

To excite the array, an identical disc is placed at the center of the ground plane, at smaller height than the upper four discs as shown in Figure 5 (a). Figure 6 shows the effect of the lower layer center disc on the radiation pattern of the antenna. Effect of the center disc on the radiation pattern was studied in 2X1, 1X2 and 2X2 array configuration, for the selected element spacing of $1.75\lambda_0$. Figure 6 (a) & (b) show the effect in the 2X1 array geometry. Compared to Figure 5 the sidelobes in the E-plane are reduced. In the H-plane, additional lobe is there in the visible region. Figure 6 (c) & (d) shows the radiation pattern when the 1X2 array geometry is loaded with center disc. Compared to Figure 5 (c) & (d), it is observed that sidelobes are suppressed in the H-plane and in the E-plane additional lobes are introduced with the distorted pattern. Figure 6 (e) & (f)

show the radiation pattern of 2X2 array, with center disc loading. For the comparison, the radiation pattern of unloaded 2X2 array is shown in Figure 6 (g) & (h). Compared to the 2×2 array without center disc, the presence of the center disc reduces the sidelobe in the H-plane by ~ 3 dB. In the E-plane, the pattern peak is off the boresight by 2°, but the sidelobes are still ~ 10 dB below the main lobe.

To make the design practical, the simulations are performed with finite ground plane. We choose 1.04 m×1.04 m squared ground plane made of aluminum, which resembles the fabricated antenna in the next section. It was shown in [3] that stacked configuration has wideband characteristics and the impedance match depends on the overlapping area of the two layers. In the present design, the additional parameter that can affect the impedance is the lower disc slot length *l*. Figure 7 shows the effect of the array spacing and the lower disc slot length on the S_{11} and the impedance over the frequency band. Each case displays two coupled resonances which corresponds to the upper and the lower layer. Figure 7 (a) shows that for the array spacing of $1.75\lambda_0$, the impedance match is obtained in the desired band. For closer spacing of $1.5\lambda_0$, due to resonance split around 1.045 GHz, there is a mismatch in the band. Hence the array spacing of $1.75\lambda_0$was selected for the design. Once the upper 2×2 layer geometry is fixed, the amount of coupling depends on the lower disc slot length. For *l* = 100 mm, the impedance is inductive as shown in the Smith chart in Figure 7 (c). Increasing slot length to 110 mm results in impedance match for the whole band, shown in Figure 7 (b). Further increase to 120 mm reduces the input resistance which results in impedance mismatch.



(a) (b) (c)

Figure 7 (a) Reflection coefficient vs frequency with array spacing as parameters (b) Reflection coefficient vs. frequency and (c) Impedance loci variation with lower slot length as parameter

3.2 Task 1.2 – Automated Discovery of Sub-Channels

In this section, we present a fully automated measurement and analysis method for finding AM and FM modulated EM emanations [8]. Note that the goal of this work is to develop a measurement technique that automatically identifies all frequencies at which at least some information about software activity will leak (the proof is the fact that software activity gets modulated onto the existing carriers), determine the type of modulation (so that it is easy to determine type of demodulation needed to extract the information) and determine quality of the modulated signals (SNR) which will determine if the information extraction will be successful or not. The proposed measurement method is an important tool for both those who want to demonstrate attacks or those who want to defend against the attacks because it allows them to identify mechanisms that lead to EM information leakage.

To find carrier frequencies at which at least some information about software activity will leak we use our SAVAT benchmarks [9] to generate an artificial leakage signal at a specific "baseband" frequency and for a specific duty cycle and record several spectra, generating a different baseband signal in each spectrum. It is not surprising that the real alternation frequency differs from the one set in the benchmarks, because the execution time of a program varies from run to run and cannot be adjusted precisely. Hence, we first propose a method to estimate the real alternation frequency, before we can proceed in finding carrier frequencies. Next, we propose a probabilistic method for separating carrier frequencies from all measured frequencies, then propose a method for identifying if the carrier is AM or FM modulated. To verify the performance of our algorithm, we tested it on a laptop, desktop, and smartphone and found that the algorithm finds the spectral patterns caused by modulated carriers with an accuracy of 99%.

3.2.1 Unintentional FM and AM Carriers in Computer Systems

Amplitude modulations (AM) and frequency modulations (FM) are well-studied and are used in numerous communication systems. Traditional communications rely on carefully designed transmit and receive signaling (i.e., carrier and baseband signals) and thoroughly regulated allocation of the frequency spectrum to optimize communication. In contrast, unintentionally modulated signals in computer systems are generated by many possible "transmitters." Note that many periodic carrier signals in computer systems are generated by digital circuits and clocks, and therefore have sharp transitions that are best approximated by rectangular pulses instead of the

sinusoidal waves used as carriers in communications systems. The spectrum of a pulse train with an arbitrary duty cycle is equivalent via Fourier analysis to a set of sinusoids with various amplitudes at fc and its multiples (harmonics). In other words, for each carrier signal generated by a digital circuit or clock, additional carrier signals will also be present at 2fc, 3fc, 4fc, 5fc, etc. As the duty cycle of a signal approaches 50%, the amplitudes of the odd-numbered harmonics (fc, 3fc, 5fc, etc.) reach their maximum, while amplitudes of the even harmonics (2fc, 4fc, etc.) trend toward zero. For a small duty cycle (i.e., < 10%) the magnitudes of the first few harmonics (both even and odd) decay approximately linearly. Finally, note that these observations imply the amplitudes of all the harmonics are a function of the duty cycle. If program activity modulates the duty cycle of a periodic signal while keeping its period constant (i.e., causes pulse width modulation), all of the signal's harmonics will be amplitude-modulated. Whether the signal is AM or FM modulated can be determined by tracking the carrier signal as the duty cycle of the baseband signal changes. For baseband signals with the highest frequency component much lower than the carrier frequency, the AM and FM spectra look very similar, but FM carrier shifts in frequency with different duty cycles, while AM carrier does not shift.

The reception of unintentional modulation "signals" differs from traditional communication receivers in several ways. Since unintentional signals occur at the frequency of the unintentional carrier, they are mixed in with all the other noise generated by the computer system (other clocks and switching noise) and other communications signals. Unintentional signals are subject to electromagnetic compatibility restrictions which impose a maximum noise power (signal

power from our point of view). Therefore, unintentional signals are typically weaker, and may be diffused across the spectrum by spread spectrum clocking or by using clock sources with inherent variation such as RC oscillators. Also, since the carriers are typically generated by non-sinusoidal sources, the carrier signals may have harmonics. Finally, communication signals have direct and obvious control of the baseband (modulation) signal, while unintentionally modulated signals from computer systems do not. We may be interested in several different system activities (baseband signals). For example, a baseband signal may be caused by processor activity and another baseband signal may be caused by memory activity. In some cases, multiple baseband signals may even modulate the same carrier. These effects complicate the detection of unintentionally modulated signals. The presence of noise generated by the system makes it difficult to determine which signals are AM or FM carriers. Some of the unintentional AM or FM carriers are generated by spread spectrum clocked signals, making them harder to recognize. Existing methods to find AM and FM modulation based on its spectral properties (i.e., without knowing the baseband signals) are not designed to deal with these issues and are not able to identify which carriers are modulated by a specific system activity.

3.2.2 Creating System Activity at Controlled Frequencies

The first step to finding unintentionally generated signals is to create a simple identifiable baseband signal. These baseband signals are generated by system activity such as the execution of particular instructions, memory accesses, etc. While we do not know the exact effect a particular

activity will have on a particular carrier's baseband signal, we can create low frequency f_{alt} variations in a particular activity, and then expect that in aggregate these variations will generate a low frequency component in the baseband signal at f_{alt} frequency.

In [9], we have introduced such microbenchmarks for generating such periodic activity. Here, we just briefly summarize the approach. The loop beginning on line 2 of Figure 8 performs one activity (activity X), and the loop beginning on line 8 performs another activity (activity Y). The outer loop repeatedly alternates activities X and Y, creating periodically changing activity whose period equals the execution time for one iteration of the outer loop. This alternation period T_{alt} is the inverse of the frequency $f_{alt} = 1/T_{alt}$.

Note that prior uses of similar micro-benchmarks [9] used this alternation to generate a carrier signal at some chosen frequency fc, while we use this alternation at f_{alt} to measure FM- and AM-modulation of any potential carrier signals intrinsically generated (and emanated) by the system.

```
1  while(true){
2    // Execute the X activity
3    for(i=0;i<inst_x_count;i++){
4      ptr1=(ptr1&~mask1)|((ptr1+offset)&mask1);
5      // The X-instruction, e.g. a load from L2
6      value=*ptr1;
7    }
8    // Execute the Y activity
9    for(i=0;i<inst_y_count;i++){
10     ptr2=(ptr2&~mask2)|((ptr2+offset)&mask2);
11     // The Y-instruction, e.g a store from L2
12     *ptr2=value;
13   }
14 }
```

Figure 9 Pseudo-code to generate the X/Y alternation activity

It is important to emphasize that while the effect of a single event (i.e. execution of a single memory access or processor instruction) on the baseband signal is unknown, as long as there is some difference between the X and Y activities, there will be a signal generated at the frequency f_{alt} and also at some of the harmonics of f_{alt} ($2f_{alt}$; $3f_{alt}$; …). Furthermore, we can change the duty cycle of the benchmark activity (i.e. (i.e. the percentage of time spent in activity X vs. activity Y) by changing how long the activity X is executed versus activity Y.

3.2.3 Finding AM and FM Unintentional Carriers in Computer Systems

Here we use the benchmarks described in the previous section to create predictable spectral patterns in the sideband of any carrier modulated by the benchmark activity. The benchmarks are run at several different alternation frequencies f_{alt1}, f_{alt2}, …, f_{altN} , for several duty cycles d_1; d_2; …; d_m, and every combination of alternation frequencies and duty cycles is recorded K times. The

frequency spectrum for each run is recorded, the repeated runs are averaged, and the result we denote as $S(f; f_{alti} ; d_j)$, where f is the frequency range at which the spectrum is recorded, f_{alti} denotes the chosen alternation frequency, and d_j denotes the chosen duty cycle. This is an important step to allow robust automated detection of both AM and FM modulations.

To illustrate what measured $S(f; f_{alti} ; d_j)$ looks like, Figure 10 plots a part of one spectrum around a carrier frequency at 382 kHz. This spectrum was recorded with f_{alt} = 23 kHz, so it shows a lower and upper sideband around 359 kHz and 405 kHz, respectively.

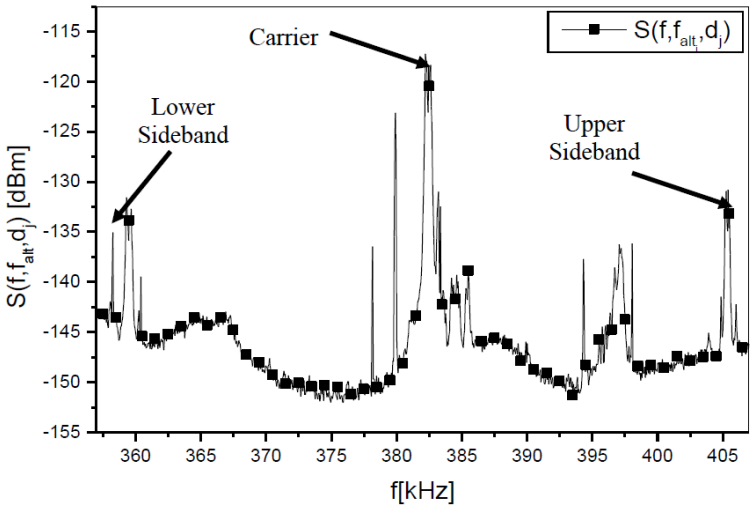


Figure 11 A measured spectrum $S(f; f_{alti} ; d_j)$ at a carrier frequency at 382 kHz and a lower and upper sidebands around 359 kHz and 405 kHz, respectively

It is not surprising that the real alternation frequency differs from the one set in the benchmarks, because the execution time of a program varies from run to run and cannot be adjusted precisely. Hence, we need to estimate the real alternation frequency f_{alt}, before we can proceed in finding carrier frequencies. First, for every duty cycle, we average spectra with different alternation frequencies, i.e.,

$$S_{avg}(f, d_j) = \text{mean}_{f_{alt_i}} S(f, f_{alt_i}, d_j), \tag{1}$$

and create new spectra as a difference between the original and averaged spectra, i.e.,

$$S_{new}(f, f_{alt_i}, d_j) = S(f, f_{alt_i}, d_j) - S_{avg}(f, d_j). \tag{2}$$

This attenuates most spectral features that are not related to modulated signals we are looking for, while preserving most of those that are activity-modulated.

To find the true alternation frequency, we shift all points in the spectrum $S_{new}(f; f_{alti} ; d_j)$ by +/- f_{alti} , and take the pointwise minimum between two shifts i.e. we compute

$$M(f, f_{alt_i}, d_j) = \min \Big[S_{new}(f + f_{alt_i}, f_{alt_i}, d_j), \\ S_{new}(f - f_{alt_i}, f_{alt_i}, d_j) \Big]. \tag{3}$$

Figure 10 plots the spectrum $S_{new}(f; f_{alti} ; d_j)$ shifted up by f_{alti} = 23 kHz (black square curve) and shifted down by f_{alti} = 23 kHz (red circle curve), their pointwise minimum $M(f; f_{alti} ; d_j)$ (blue triangle curve). Also shown (magenta diamond curve) is the pointwise minimum computed in the same way (shifting by 23 kHz) for another spectrum whose alternation frequency is different (e.g., 29 kHz). We observe that, when the spectrum contains sidebands that correspond to f_{alti}, the shift in frequency aligns these sidebands at the frequency that corresponds, in the original spectrum, to the carrier that produced the sidebands (382 kHz in this case).

At points that do not correspond to the modulated carrier or its sidebands, the pointwise minimum will only have a peak if two prominent spectral features (e.g. two radio unrelated signals) happen to be separated by exactly $2f_{alti}$. Finally, when the spectrum is shifted by an amount that does not match the alternation frequency, the sidebands do not align and the pointwise minimum is unlikely to have a peak even at the carrier's frequency.

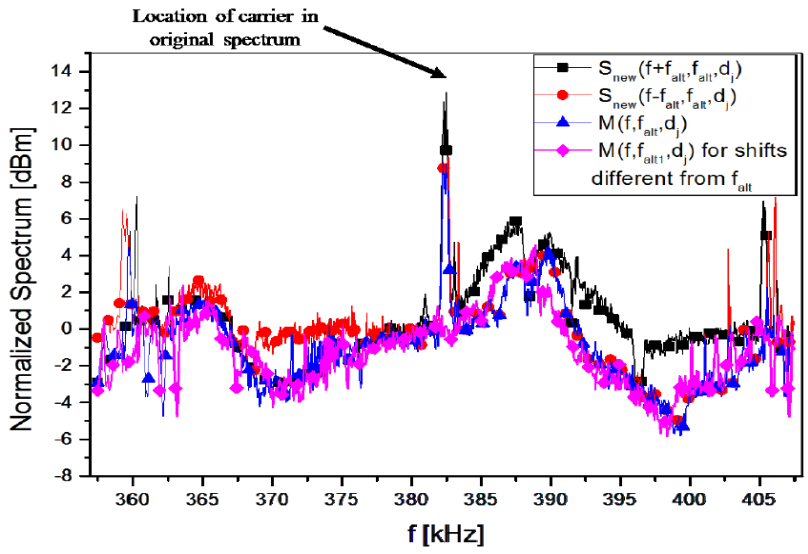


Figure 12 A spectrum $S_{new}(f; f_{alti} ; d_j)$ shifted up and down for 23 kHz, the pointwise minimum between these two spectra, and the pointwise minimum between two spectra with shift different from f_{alti} = 23 kHz

Instabilities in program execution can cause the actual alternation frequency to be different from the intended one. To find that actual alternation frequency, we compute this minimum-of-shifted-spectra operation with all frequency shifts that are within 25% of the intended one, in 50 Hz increments. For each of these $M(f; f_{alti} ; d_j)$ we compute the average across f, and the shift that produced the largest average it taken as the actual alternation frequency. The intuition behind this is that shifts that correspond to the true alternation frequency will produce the stronger peaks at frequencies that correspond to modulated carriers and will possibly have other peaks that come from aligning unrelated signals. In contrast, incorrect shifts will only have the peaks that come

from aligning unrelated signals, but their sideband-induced peaks will be attenuated (or completely eliminated). Thus the shift that corresponds to the actual alternation frequency tends to produce more (and stronger) peaks, which increases its average-over-f relative to other shifts.

In our experiments we found that the actual alternation frequency is often 150 to 300 Hz away from the intended one. This difference may seem small, but some sidebands are sharply defined, e.g., the peak is only 100 to 200 Hz wide, so use of the intended rather than true alternation frequency may cause our approach to completely miss the actual sideband signals and thus not report the corresponding modulated carrier signals.

To find the frequencies of carriers that are unintentionally modulated by program activity, we perform the following steps for each duty cycle d_j. First, for every alternation frequency f_{alti}, where $0 < i < N$, the spectrum $S(f; f_{alti} ; d_j)$ (that corresponds to that alternation frequency) is shifted by +/- f_{alti} to the left and by f_{alti} to the right. This creates $2N$ spectra that all correspond to the same duty cycle and whose sideband signals are shifted to the frequency of the carrier that produced that sideband signal. Then, the pointwise minimum among all these shifted spectra is found, i.e.,

$$
\begin{aligned}
&M_{\text{true}}(f, d_j) = \\
&\min\Big[S(f + f_{alt_1}, f_{alt_1}, d_j), S(f - f_{alt_1}, f_{alt_1}, d_j), \\
&\quad S(f + f_{alt_2}, f_{alt_2}, d_j), S(f - f_{alt_2}, f_{alt_2}, d_j), \\
&\qquad\qquad \vdots \\
&\quad S(f + f_{alt_N}, f_{alt_N}, d_j), S(f - f_{alt_N}, f_{alt_N}, d_j)\Big].
\end{aligned}
\tag{4}
$$

Intuitively, at a frequency that corresponds to a modulated carrier, the sidebands that correspond to different f_{alt} will all align, and the minimum will have a peak. At other frequencies, the minimum will have a peak only if other stronger-than-usual signals happen to be present in the original spectra at every one of the $2N$ positions, which becomes increasingly unlikely as we increase N. However, it is still possible that other signals happen to align and create peaks in $M_{true}(f; dj)$. To suppress these peaks, for every alternation frequency, we also compute $M_{false}(f; k; dj)$ by taking each spectrum (collected with f_{alti}) and shifting it by $\pm f_{alt_{i+k\neq 0}}$, then taking the point-wise minimum among such spectra:

$$
\begin{aligned}
&M_{\text{false}}(f, k, d_j) = \\
&\min\Big[S(f + f_{alt_{1+k}}, f_{alt_1}, d_j), S(f - f_{alt_{1+k}}, f_{alt_1}, d_j), \\
&\quad S(f + f_{alt_{2+k}}, f_{alt_2}, d_j), S(f - f_{alt_{2+k}}, f_{alt_2}, d_j), \\
&\qquad\qquad \vdots \\
&\quad S(f + f_{alt_k}, f_{alt_N}, d_j), S(f - f_{alt_k}, f_{alt_N}, d_j)\Big].
\end{aligned}
\tag{5}
$$

The key property of $M_{false}(f; k; dj)$ is that it is computed in exactly the same way as $M_{true}(f; dj)$, but the use of incorrect f_{alt} causes none of the sideband signals to be aligned with each other. This is repeated for different non-zero values of k and compute the permutations of f_{alti+k}, and we compute $M_{false}(f; d_j)$ as the point-wise average among $M_{false}(f; k; d_j)$ across all non-zero values of *k*. Figure 11 plots $M_{true}(f; d_j)$ and $M_{false}(f; d_j)$ for the experiment where there is an activity-modulated carrier at 382 kHz. We can observe that the $M_{true}(f; d_j)$ has a distinctive peak at the carrier frequency, while $M_{false}(f; d_j)$ does not. However, accidental alignment of other (non-sideband) signals would produce similar peaks in $M_{true}(f; d_j)$ and $M_{false}(f; d_j)$. Thus we compute a "modulated carrier score" MCS(f) as the point-wise ratio between $M_{true}(f; d_j)$ and $M_{false}(f; d_j)$:

$$MCS(f) \quad = \quad 10 * \log_{10}\left(\frac{M_{\text{true}}(f, d_j)}{M_{\text{false}}(f, d_j)}\right). \tag{6}$$

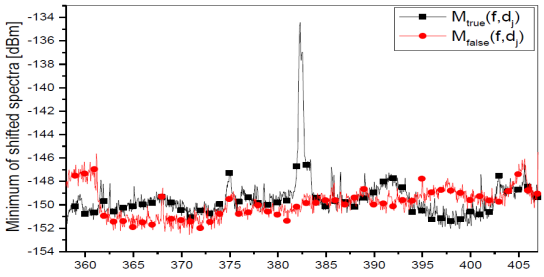


Figure 13 Minimums of shifted spectra, i.e., $M_{true}(f; d_j)$ and $M_{false}(f; d_j)$, with the carrier frequency at 382 kHz and the alternation frequency of 23 kHz.

Intuitively, at each frequency the value of the MCS corresponds to how much stronger (in dB) is the signal that corresponds to the sidebands of that (potential) carrier, relative to the signal that would be computed for that frequency even if no sideband present. To illustrate this, Figure 12 shows the MCS(f) that corresponds to Mtrue(f; dj) and Mfalse(f; dj) from Figure 11.

The MCS(f) shown in Figure 12 has a strong peak that strongly suggests that a modulated carrier is present at 382 kHz, the MCS(f) varies and has many other, smaller, peaks, so it is not easy to determine what value of MCS should be treated as the threshold for reporting a modulated carrier. If the MCS threshold is set to some manually selected value, it will need to be adjusted for each evaluated computer system, environment in which the experiment is carried out, antenna position, etc.

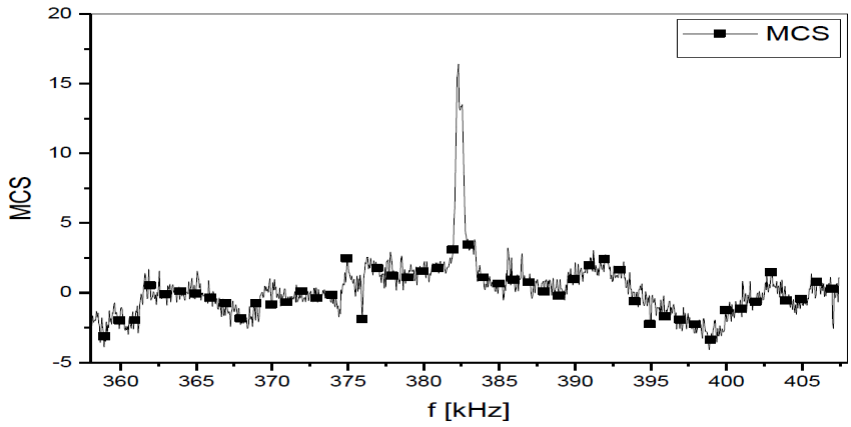


Figure 14 Modulated carrier score as a function of frequency for a spectrum with the carrier frequency at 382 kHz and the alternation frequency of 23 kHz

Instead, it is highly desirable to set a threshold in terms of the probability that a reported carrier is a false positive, and then automatically determine the corresponding threshold for MCS. To accomplish this, we note that $M_{true}(f; d_j)$ and $M_{false}(f; d_j)$ should be statistically equivalent for frequencies hat are not modulated carriers, so for those frequencies the values of MCS(f) should have a zero mean and a CDF that is symmetric around that mean. In contrast, for frequencies that correspond to modulated carriers, the MCS(f) will have a bias toward positive values, and the magnitude of that bias increases as the power of sideband signals increases. Thus, the problem of deciding how likely it is that a particular frequency has a modulated carrier becomes the problem of determining how likely it is that the MCS(f) value for that frequency belongs to the positive-biased "modulated carrier" distribution rather than the symmetric "baseline" (no modulated carrier) distribution.

Although empirical data for the baseline distribution is not available (the MCS(f) contains points from both distributions), the baseline distribution can be closely approximated by noting that 1) the baseline distribution is symmetric around zero and 2) negative values of MCS(f) are very likely to belong to that distribution. The negative-values part of the baseline distribution is thus approximated by simply using the negative-values part of the empirical joint distribution, while the positive side of the baseline distribution is approximated by using the "mirror image" of the empirical joint distribution. Figure 13 shows the empirical joint distribution and the approximated baseline distribution.

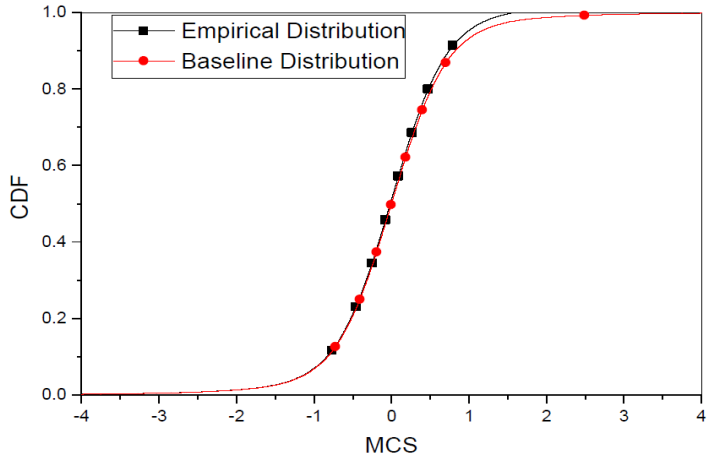


Figure 15 Empirical joint and baseline cumulative distribution functions for MCS score.

It can be observed that the empirical joint distribution has more high-magnitude points than the approximated baseline distribution. Thus, we can now set the probability-of-false-positive threshold (p_{fp}) to a desired value, e.g., $p_{fp} \leq 0.02$, look up the MCS value that corresponds to $1-p_{fp}$, and report carriers whose MCS is no less than that value. For reported MCSs, we than read the actual CDF value and report it as the confidence level. For example, for $p_{fp} \leq 0.02$, we find all MCSs that have value larger than MSC that corresponds to CDF=0.98. Then, for each MCS that satisfies this criteria, we read their actual CDF value. All values should be larger than 0.98.

Section IV-B described how to identify modulated carrier frequencies for a given duty cycle d_j. To identify if the carrier has AM or FM modulated signal, we observe how the carrier's frequency and sideband power change as the duty cycle changes. Note that an amplitude-

modulated carrier should have the same frequency for all duty cycles (although the magnitude of the carrier and baseband signals will vary as the duty cycle changes). For a frequency-modulated carrier, however, the change in the duty cycle changes the DC-value of the baseband signal, which results in shifting the frequency of the carrier and its sidebands in proportion to the duty cycle.

Intuitively, if we plot the modulated carrier's frequency on the Y-axis and the duty cycle on the X-axis, a horizontal line corresponds to AM, while a line with a non-zero slope corresponds to FM whose _f corresponds to the line's slope. To reduce the number of spectra that must be collected, however, we only get discrete points on this line that correspond to duty cycles used in the experiments. Furthermore, the AM/FM identification (and the estimate of Δf for FM) relies on estimating the slope of the frequency-vs.-duty cycle line, so the duty cycles used in the experiments should not be too close to each other. Finally, the linear fit is imperfect – the actual duty cycle may differ from the intended one, the empirically determined frequency of the modulated carrier may contain some error, etc. Thus, the key problem in identifying modulation is how to group together likely-carrier points from different duty cycles, i.e., for a likely-modulated-carrier point found for a given duty cycle, determining which likely-carrier points from other duty cycles belong to the same modulated carrier. Unfortunately, simply using the points that produce the best goodness-of-fit (e.g., squared-sum-or-errors) for the frequency-vs-duty-cycle produces poor results when several modulated carriers that do not have a very sharply defined central frequency are present in the same frequency range. To overcome this, we note that the

sideband power produced by a carrier is also a function of the duty cycle, i.e. the points that belong to the same carrier but with different duty cycles should all have the sideband power $P_j = P_{max} \sin(d_j \pi)/\pi$, so their Mtrue(f; d_j) should also be proportional to $\sin(d_j \pi)/\pi$. Thus our modulation-finding consists of finding, for each likely carrier point, the linear fit (that uses one point from each duty cycle) that produces the smallest product of the squared sum of error for the frequency fit and the squared sum of errors for the (M_{true}) fit.

Because the slope of the linear fit is estimated, it is highly unlikely to be exactly zero. Thus we also determine the 95% confidence interval for the estimated slope, and report the carrier as AM if this confidence includes the zero value. Intuitively, we report a carrier as FM-modulated only if there is a high enough (95%) confidence that its frequency change is duty-cycle-induced rather than caused by other (duty-cycle unrelated) variation in estimated frequencies of modulated carriers.

3.3 Task 2.1 – Spectral Monitoring for Anomaly Detection

In this section, we describe our new framework, REMOTE (Robust External Malware Detection Framework by Using Electromagnetic Signals), that is designed to address practical issues for anomaly detection in resource-constrained devices (e.g., embedded devices, IoTs, CPS, etc.) [10], [11], [12]. We envision that REMOTE can be used in scenarios where the security of

the device is critical, e.g., devices that control critical infrastructures, military systems, hospital equipment, etc.

3.3.1 Spectral Samples (SS)

At a high level, REMOTE has two phases: training and monitoring. In both phases, the EM signal is first transformed into a sequence of spectral samples (SS) by using short-time Fourier transform (STFT), which divides the signal into equal-sized segments (windows), where consecutive segments overlap to some degree. STFT then applies the Fast Fourier Transform (FFT) to each window to obtain its spectrum. In our measurements, we use a 1ms window size with 75% overlap between consecutive windows, which provides a balance between the computation complexity and frequency/time resolution. The rest of the training and monitoring operates on this sequence of spectra, where each spectrum (i.e., the spectrum of one window) is referred to as a Spectral Sample (SS).

3.3.2 Distance Metric for Comparing Spectral Samples

In both training and monitoring, REMOTE will need to compare SSs to each other, and for that, it requires a distance metric – a way to measure the "distance" between SSs in a way that corresponds how likely/unlikely they are to have been produced by execution of the same code. This distance metric should be sensitive to the aspects of the signal that change when executing different code, but insensitive to aspects of the signal that change between physical instances of

the same device or over time on the same device instance. To achieve this, we create a new distance metric, Clock- Adjusted Energy and Peaks (CAPE).

Based on the insights from our prior work [10], [11], the frequencies of the peaks in the signal around the clock frequency are an excellent foundation for constructing a distance metric that is sensitive to which region of code is executing. Unfortunately, our experiments have shown that the clock frequency can vary over time and among device instances, and a change in clock frequency also changes the frequencies of loop-related peaks around it. One difference is that, because the peaks' frequencies are all relative to the carrier frequency, any shift in the clock frequency also shifts the frequencies of the loop-related peaks by the same amount. The second change is caused by the relationship between clock frequency and program performance. Specifically, as the clock frequency increases, the program executes faster, leading to a lower per-iteration time T; higher frequency of the loop ($f_l = 1/T$), and thus moving the loop-related peaks away from the clock's frequency. Similarly, lower clock frequency moves the loop-related peaks closer to the clock frequency.

Thus, the first step in computing our CAPE distance metric is to, for each frequency f that is of interest in an SS, compute the corresponding normalized frequency as $f_{norm} = \frac{f - f_{clk}}{f_{clk}}$, where f_{clk} is the clock frequency for that SS. This normalized frequency is expressed as an offset from the clock frequency so that a shift in clock frequency does not change f_{norm} with it and is

normalized to the clock, so it accounts for the clock frequency's first-order effect on execution time.

To make CAPE robust to weak signals and/or signals that have no well-defined peaks, we first consider the overall signal power (sum of magnitudes in the spectrum) of the signal outside the vicinity of the clock. The power of a poorly-defined peak is spread across a range of frequencies – visually it is a wide and not-very-tall "hump" rather than a narrow and tall "peak". When comparing two SSs that are different but each contain only "humps" and no sharp peaks, if we only consider whether the SSs have power concentrations at the same (clock-adjusted) frequencies, the overlap among their "humps" causes these SSs to match much better than they otherwise should, and this can prevent detection of malware-induced changes in signals.

Moreover, under poor signal-to-noise conditions (e.g., when the signal is received at a distance) sharp peaks are likely to still stand out of the noise, so due to random variation in noise, some "humps" end up below the noise level and some do not. For two SSs that should be the same (except for the noise), this causes poor matches, and this can lead to false positives. Thus, to make our CAPE distance metric more robust against weak/noisy signals, we use a new insight, called "non-clock-energy" test, that non-clock power varies very little among SSs that do belong to the same region, and that increases/decreases in a loop's overall per iteration time concentrate less/more power toward the clock frequency in an SS. Therefore, SSs whose non-clock power

differs by more than 0.5 dB are considered dissimilar by CAPE, and no further comparison between them is needed.

If the two SSs pass the "non-clock-energy" test, REMOTE compares them according to the (clock-adjusted) frequencies of their most prominent peaks. Specifically, we take N highest-magnitude frequency bins from the spectral sample (SS) that are each (i) not part of the *NoiseList*, and (ii) not within *D* spectral bins of a higher-amplitude spectral bin. The number *N* is determined differently for training and monitoring, as will be described shortly. The *NoiseList* contains frequencies of signals that are present regardless of which specific region of the application is executing.

For finding the *NoiseList*, we record the EM signal several times and average them while no program is being executed (the system is idle). We then choose 10 random SSs in the recorded signal, and then for each SS, sort it and find all the spikes that are at least 5 dB above the noise floor and put them in the *NoiseList*. We empirically find that choosing 10 points is sufficient to find all the strong peaks since it can accurately capture the transient behavior of the environmental noise. It is also important to point that, using this method, our detection algorithm is robust to interference from nearby devices (that are not identical to the monitored device), as their clock and other frequently-occurring peaks will end up on the *NoiseList*. The reason for ignoring *D* spectral bins that are too close to even-higher-magnitude ones is that a very prominent peak in the spectrum typically has "slopes" whose magnitude can exceed the magnitude of other peaks, and we found

that REMOTE is more robust when its decisions are based on separate peaks rather than just a few (possibly even one) very strong peaks and a number of frequency bins that belong to their "slopes".

Finally, REMOTE combines the information about the frequencies of the peaks in the two SSs into a single value that represents the distance among the SSs. For each peak in one SS, REMOTE finds the closest-frequency-peak the other. If the frequency difference is large enough, the peak votes for a mismatch, and the ratio of the mismatch votes to the number of all (mismatch and match) votes is used as the distance metric between the two SSs.

3.3.3 Black Box Training

To train REMOTE, signals are collected as the unmodified monitored device emanates them. However, care should be taken to achieve good coverage of the software behaviors, e.g., by using the same methods that are used to test program correctness. The problem of achieving good coverage tends to be easier for many applications in the CPS domain, especially those where correct operation is critical, because correctness concerns and the need for easy verification of correct operation motivates developers to produce code that has relatively few code regions, and with very stable patterns for how the execution transitions between them. In such cases, normal use of the device is likely to provide good coverage of the application's code regions after a while.

After signals are obtained and converted into SSs, a key part of training is to associate SSs with the code regions they correspond to. To achieve this without using instrumentation or other

on-the-monitored-device infrastructure, REMOTE relies on a general observation that a given region of code tends to produce EM signals whose SSs are similar to each other, while the SSs from different regions tend to differ from each other to various degrees. This observation allows us to group SSs according to similarity, and for that we use Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN), a technique that performs clustering without any a priory knowledge about which cluster (region) each sample (SS) corresponds to, and with no a priori knowledge about the number of clusters (regions). Like other clustering algorithms, HDBCASN needs a distance metric, and in REMOTE that distance metric is the new CAPE metric, using N = 10 peaks. Using this variation-robust metric allows training signals to be collected over time (e.g., over many hours), and/or on multiple device instances.

Because HDBSCAN clustering is based solely on similarity among SSs, its result may not precisely correspond to actual regions of the code, e.g., one region may produce more than one cluster if there are several distinct ways in which the region can execute, or two regions may end up in the same cluster if their execution produces very similar signals. Neither of these possibilities is a problem for REMOTE, and in fact, they result in improved sensitivity and performance. If separate clusters for distinct behaviors were forced into a single cluster, the resulting unified cluster would allow a wide variety of SSs to match - all the valid SS options and also everything that lays in-between in the distance-space used for clustering. By creating a separate cluster for each distinct possibility, REMOTE will detect anomalies that produce SSs that are not valid but lay in between the valid ones. Conversely, when multiple regions are clustered together, they have very similar

(practically indistinguishable) signals and it is more efficient and robust to treat them as one cluster. During monitoring, a Finite-State Machine (FSM) is used to keep track of the current region of the code. For each test, REMOTE compares the new SS to either the current region or any valid "next region" that has been seen during training.

3.3.4 Monitoring

During monitoring, REMOTE receives the signal and converts it to SSs in the same way it was done in training (same window size and overlap). After that, REMOTE can be viewed as a classifier that places each spectral sample (SS) into either one of the known categories (clusters identified during training) or into the "unknown" category that represents anomalous behavior, according to our CAPE distance metric (Fig. 14 shows the flow-chart of REMOTE's monitoring algorithm). Specifically, a candidate region is rejected if its distance metric is above 50% (fewer than half of the peaks match). If all candidates are rejected, the observed SS is categorized as "unknown," otherwise it is categorized into the candidate category with the lowest CAPE distance metric. The number of peaks used for each cluster is no longer fixed at 10 – instead, it is identified for each cluster during training. We start with ten peaks, but then remove those that occur in fewer than 10% of the SS in the cluster. If this results in removing all peaks, we still retain the two most frequently occurring (among SSs from that cluster) peaks. This helps matching accuracy when the SSs in a cluster have few prominent peaks and a number of very weak peaks – in such cases it is more robust to use only the overall non-clock energy and the prominent peaks for matching than

to use the peaks that may “disappear into the noise” due to changes in distance, antenna position, etc.

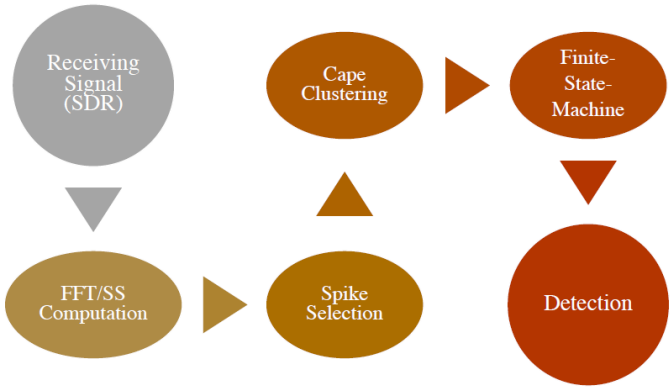


Figure 16 Remote’s monitoring flow-chart

However, if the overall decision to report malware is based on only one SS, brief occurrences of strong transient noise can result in false positives. To avoid that, REMOTE only reports an anomaly if N consecutive SSs are classified as “unknown.” The value of N should be chosen depending on the EM noise characteristics of the environment, but we found that N between 3 and 5 tends to work well in all our experiments. We use N=5 because it biases REMOTE toward avoiding false positives, while still maintaining an excellent detection latency (N=5 corresponds to only 1.25 ms detection latency in our setup). An FSM is used to count N, report an anomaly,

and to keep track of current valid region of code to ensure that the program follow a correct ordering of regions.

Finally, we found that in the presence of an OS, interrupts and other system activity that occurs during an SS can make that SS dissimilar to those from training. For example, an interrupt that lasts <1ms can affect four consecutive SSs (recall that we use 1ms windows with 75% overlap), so a naive solution would be to add 4 to N (number of consecutive "unknown" SSs that are needed to trigger anomaly reporting). Using N = 9 indeed eliminates interrupt-induced false positives, but also prevents detection of attacks that are brief. Unfortunately, real-world malware (e.g., the attack on Syringe-pump that will be described in Section 3) can introduce only a short burst of activity into the otherwise-normal activity of the application. Fortunately, our experiments indicate that spectral features of interrupt activity are similar to each other, so during training interrupt activity can be clustered. During monitoring, REMOTE includes these clusters as candidates, allowing it to tolerate interrupts without becoming tolerant of similar-duration deviations from expected execution.

3.4 Task 2.2 – Spectral Monitoring for Multi-core Anomaly Detection

In this section, we propose a mixture of Markov and convolutional neural network (CNN) models, called MarCNNet, to monitor multi-core systems to detect malware [13]. In this framework, the CNN provides the likelihood of executing any state of any program by exploiting the features that are learned during the training phase, and the Markov Model monitors programs

by investigating the possibility of state transitions. Compared to other CNN models that have the same procedure for training and testing, these phases follow different paths in MarCNNet to simplify collection of training signals and decrease complexity of the framework. We propose to use Markov Model to carry dependency information among paths. States of the model are assumed to be hot paths or loops, and transitions between any two states are only allowed if the program follows this path. Since the Markov Model is responsible for tracking execution order, neural network inputs are assumed to be independent of each other during training. The outputs of the CNN are used as an indicator of the current state. In the testing phase, based on the likelihood of states obtained from the CNN, the Markov Model tracks whether the estimate-stream of the CNN coheres with the allowed transitions. The methodology alerts users if the estimates and corresponding transitions do not adhere. In summary, the work makes the following contributions:

- Proposes a zero-overhead methodology to profile multicore devices while multiple programs are executed at the same time.
- Proposes a CNN model that extracts the features of program states and provides likelihood estimates of each state at a given time.
- Defines a new structure that combines Markov and CNN models to simplify and decrease the complexity of the training phase of the network.
- Defines procedures for generating training data to mimic multi-core systems by exploiting signals collected only when a single core is active.

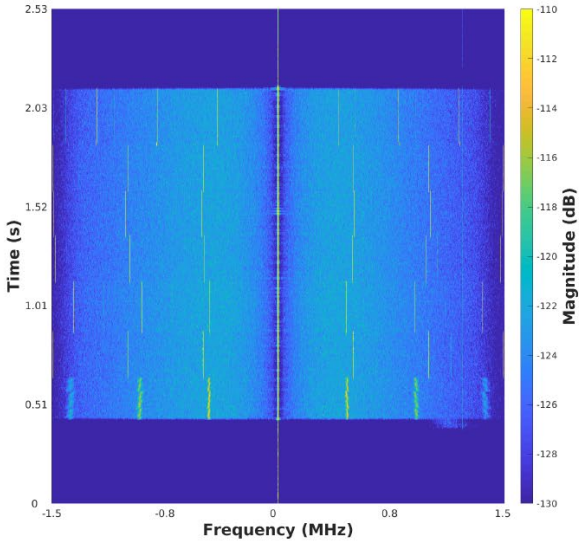
- Provides a proof-of-concept implementation of the proposed method to demonstrate its practicality on various devices.

3.4.1 Emanated EM Signals During Program Execution

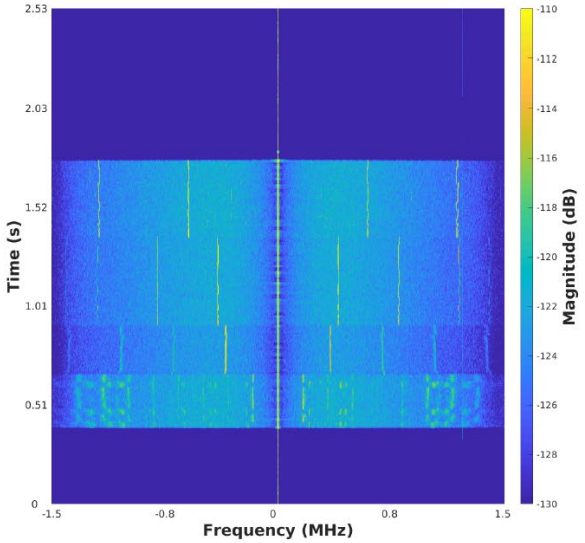
Program executions change states of transistors in a processor and yield radiation of EM signals. The correlation between the executions and transitions is the main underlying reason for these signals to convey sensitive information. It has been shown that these emanated signals are modulated by the clock frequency of a device [11], i.e., loop activities in a program generate peaks around the clock frequency. Therefore, with a proper choice of antenna and a clock harmonic that diminishes the effect of disruptive signals, it is possible to capture informative signals that are correlated with the program activities even from a considerable distance [14], [15].

Programs or applications are written to serve a specific task, thereby each task within the program shows some dependency to others. These tasks can be considered as the hot spots since execution run-time is mainly spent on these regions. Therefore, we can claim that programs are combinations of these hot spots, and the execution of each of these regions depends on previously executed ones. To illustrate the modulation of signals and the domination of hot regions, we consider the Bit_count benchmark from MiBench suit [16]. The spectrogram of the received EM signal is given in Fig. 15a when only one core of the device is active. To obtain the figure, we first demodulate the signal by the clock frequency of the device, then take STFT of the signal and plot in time. The lighter regions in the figure represent the frequency components that are relatively

strong. Here, the most powerful frequency component at the center corresponds to the clock frequency of the device, while the other strong frequency bins around the clock frequency represent the modulated frequency components due to looping operations that exist in the benchmark. The main observation is that the benchmark contains seven dominant regions which are executed in an order. The frequency components observed in the spectrogram are related to execution time of a single iteration of a loop. For example, if the iteration takes T_{alt} seconds, we observe an RF signal component at the alternation frequency, $f_{alt} = 1/T_{alt}$. However, if the iteration time varies in time due to program activities, we observe a smearing around the expected frequency, as seen in the first loop of Bit_count.



(a) Bit_count.



(b) Basicmath.

Figure 17 Hot regions for the profiled micro-benchmarks from MiBench [16]

Similarly, the spectrogram of the received signal when a single core is running Basicmath (another benchmark from MiBench) is given in Fig. 15b. This time, four different regions are observed with non-overlapping frequency components. We also observe that these regions follow a sequential order such that each hot region has only one path like in Bit_count.

The question here is how emanated signals are composed when multi-cores are active. To investigate the mixtures of signals while multiple programs are executed at the same time, we perform an experiment while running Basicmath and Bit_count parallel on different cores of a device. The spectrogram of the received signal is given in Fig. 16. We observe that the received signal is the superimposed version of the signals when Bit_count and Basicmath are executed in a single core with lower signal-to-noise ratio (SNR). Therefore, it is possible to reconstruct the spectrogram of emanated signals when both cores are active if the relative initialization times of the programs are known. Note here that a perfect superposition of signals is not possible because whenever multi-cores are active, it draws extra power that causes a certain amount of decrease in SNR of the received signals. However, the received signal in Fig. 16 illustrates that it still preserves the characteristics of both programs. The same frequency components with single-core measurements are still active and relatively stronger. Therefore, the STFT magnitudes of the received signals for multi-core devices can be modeled as the summation of STFT outputs of single core measurements with some additive white noise that represents the extra power drawn by the device. As long as one core measurements are available for each program, any possible signal combination can be generated irrespective of the time that programs are initialized. Also, this

model simplifies training data collection phase since it does not require performing experiments by varying initialization times of programs.

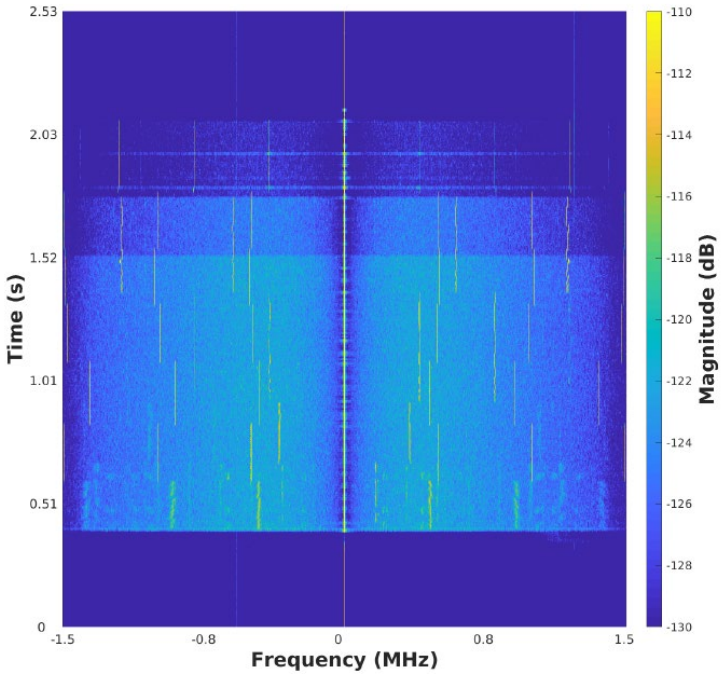


Figure 18 Hot regions when both micro-benchmarks running at different cores

3.4.2 Markov Model Based Program Profiling: MarCNNet

In this section, we introduce Markovian Convolutional Neural Network Model, called MarCNNet, to monitor systems with no overhead and detect malware when injected. In this section, for the simplicity of discussion, we consider a device with two cores and the following programs: Bit_count and Basicmath.

Remembering that execution of a program follows a path, we first investigate the spectrograms given in Fig. 15a and Fig. 15b. For both cases, hot paths dominate the run-time of the programs and demonstrate a similar pattern during a loop execution. These loops can be considered the states of a given program as they activate distinct frequency components which could be the first candidates to reveal the current state. Moreover, these distinct features follow an order/path that is defined by the program. To represent this dependency among hot regions, we propose utilizing Markov Models. The states of the model are considered as hot regions, and transitions among states are only allowed if there exists a branching operation that enables consecutive execution of corresponding hot regions.

The state transition diagrams of Markov Models for both benchmarks are given in Fig. 18. As given in the figure, the states of each benchmark depend only on the previous hot region, meaning that there is only one path from each state for these benchmarks. Please note that these benchmarks are state-of-the-art and the proposed model can track a program with many branching operations hence more complicated Markov Models.

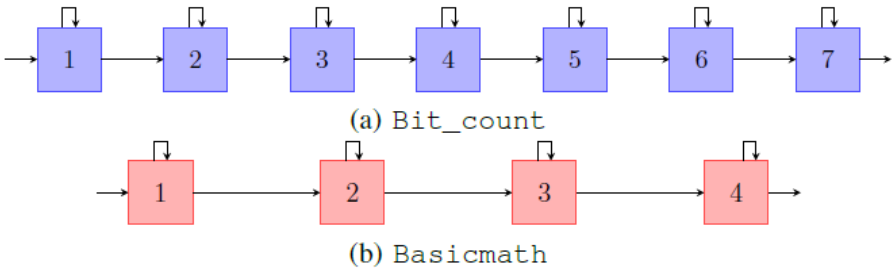


Figure 19 Markov models representing the execution of benchmarks

The model is designed to have N parallel units under the assumption that the system is running N sets of programs. However, the question is how to combine the CNN with Markov Models. Although the CNN can classify the inputs by extracting the distinctive features, it does not consider the dependency among its seemingly-independent inputs. To address this dependency, we exploit Markov Models as the state monitoring machine. In other words, the CNN is considered as the likelihood estimator of the current state of a given program and the Markov Model behaves as the inspector of state transitions. The Markov Model is responsible for warning the system that there could be an anomaly. Therefore, the combination of these two models provides a tool for monitoring systems against malware.

After having a model to represent the dependency of the execution paths, the question is how to identify the current execution point of a given program. Please remember that hot paths are generally a result of looping activities within a program. Assuming each iteration of the loop takes equal time to execute, we expect that the same frequencies are activated until the end of the loop. However, the execution times of software activities vary, which causes shifts in the frequency domain [17]. To be able to deal with problems that arise due to working on multi-core systems and the spread of the frequency components due to execution time variation, we use a convolutional neural network, which has better built-in invariance to local variations. Convolutional neural networks are generally used for image, speech and time series, and the structure of the overall model is important to achieve better true classification rates.

The proposed model, MarCNNet, is given in Figure 19. The model comprises N parallel units which correspond to the N different programs that are monitored. The detailed CNN model at each branch is given in Figure 19. Each of these branches contain:

- 3 convolutional layers,
- 3 dense layers,
- 1 output layer whose size changes based on the number of states of the corresponding program.

After the first and second convolutional layers, we apply max-pooling operations with a stride of 5 and a kernel size of 10. The kernel sizes of the convolutional layers are 55, 35, 15, respectively. We also utilize dropout layers before each convolutional layers with a dropout rate of 0.05. ReLU activation function is exploited at each layer except the output layer. We exploit cross entropy loss function which is defined as

$$loss[\mathbf{o}, class\_id] = -\log\left(\frac{\exp(\mathbf{o}[class\_id])}{\sum_{n}\exp(\mathbf{o}[n])}\right) \tag{7}$$

where class id is the state label of the considered training signal and o is a vector containing the outputs of the neurons at the output layer. We analyze the model with various hyperparameters, i.e. # of layers, stride, etc., and come up with the parameters given above. However, we do not claim that this model is the optimal model because there are infinitely many options for the model selection, yet the proposed CNN model extract features that are distinctive to profile a system.

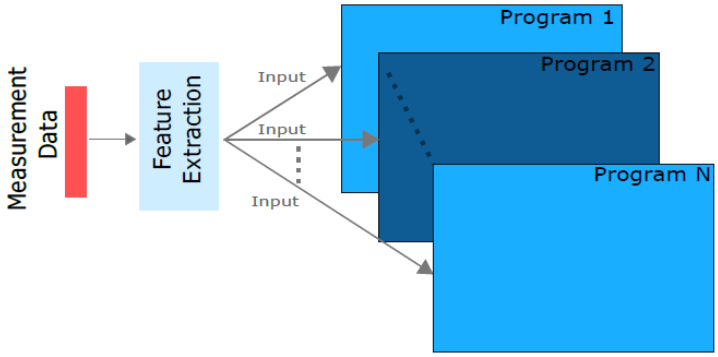


Figure 120 Convolutional neural network model to track N different programs

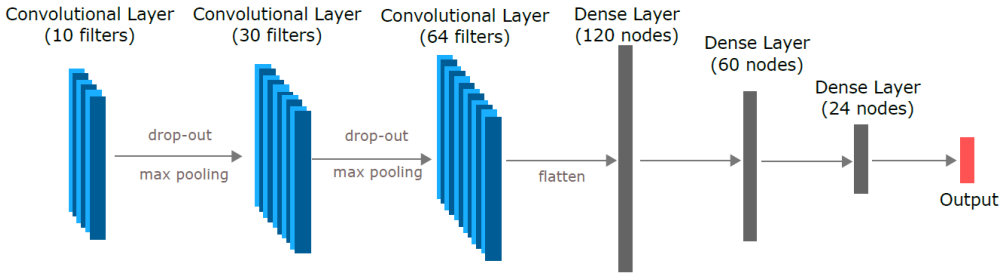


Figure 21 Detailed convolutional neural network model for the branches

3.4.3 Input Signal and Training Phase

In this section, we first describe the input signal that is fed to the neural network, and then explain the training phase of the overall framework that extracts and learns the distinctive features of "hot" paths of each program. These learned features of programs constitute the main structure of the profiling scheme by providing the matching probabilities of testing signals.

Monitoring devices in real-time with high sampling rates results in very large input data dimensions and requires preprocessing before getting fed to the neural network. In addition, the measured signal is corrupted by non-relevant activities and this corruption should be minimized for accurate program tracking. One straightforward approach is to use time series samples with a specified window. However, because of the high sampling rates of the measuring devices (which increases the dimension of the input layer), interrupts and corruption due to environmental signals, this method can return in lower accuracy rates, hence false malware alert [18]. Moreover, this approach gives a larger number of parameters to optimize. To reduce the size of the data and increase the SNR level, we utilize the first phase of the Two-Phase-Dimension-Reduction methodology proposed in [19]. This method first calculates the STFT of the signal and averages the magnitudes of STFT outputs. For a better explanation of the method, let $\mathcal{F}$ be the STFT window size, O_S be the number of non-overlapping samples at each STFT calculation, Ξ be the number of STFT operations to average, and $\mathbf{e}_F$ be the extracted features that are inputs to the neural network. The features can be written as

$$\mathbf{e}_F[k] = \sum_{n=1}^{\Xi} |X_n[k]| \tag{8}$$

where $k \in \{0, 1, \cdots, \mathcal{F}-1\}$, and

$$X_n[k] = \sum_{\xi=0}^{\mathcal{F}-1} \mathbf{\Theta}\left[\xi + (n-1)O_S\right] \exp\left(-j2\pi k\xi/\mathcal{F}\right) \tag{9}$$

where Θ is the measured raw signal. To generate $\mathbf{e}_F$ [k], the number of time series samples used in the framework is Φ where $\Phi = \mathcal{F} + (\Xi - 1) \times O_S$.

Therefore, the size of the input vector for the neural network reduces to $\mathcal{F}$ from Φ, which helps to:

- Improve SNR by diminishing the power of unrelated activities,
- Reduce the number of parameters of the neural network, hence decrease the complexity.

However, we do not provide $\mathbf{e}_F$ to the neural network in linear domain because most of the power is dominated by the DC component and other frequency components have relatively small amplitude which can results in floating errors. These frequency components can be disregarded while training the network, hence a normalization operation is required to pay more attention on each individual frequency component. The normalization is done by converting the input vector into dB domain as $\hat{\mathbf{e}}_F = 20\log_{10}(|\mathbf{e}_F|)$. The normalization is generally done by dividing the signal by its mean and standard deviation or subtracting the mean from the input vector. However, converting the input into the dB domain introduces extra non-linearity to the problem, hence increases the accuracy of profiling.

The proposed neural network has many parallel units that are independent of each other (there is no interconnection among different units). Therefore, in the training phase, the framework can be divided into N different CNN models, and treated as separate problems. However, the

question is how to obtain training signals to feed the neural network since the collected training signals correspond to single core measurements. The process to generate training signals with the available one-core measurements representing various scenarios is given in Algorithm 1. The possible scenarios and corresponding strategies can be listed as follows:

STRATEGY - 1: The training signal is kept the same assuming that the test signal is collected when only a single core is active with the same noise figure.

STRATEGY - 2: The training and testing data can be collected in different environments with different noise figures. To make the proposed model more resilient to noise due to environment, we add additive noise to the input of the CNN.

STRATEGY - 3: This mimics the behavior of a multi-core signal that has the same noise figure with the collected training signals. It superimposes signals from different states of different programs by employing random weights to consider the destructive/constructive effect of multi-core signals on each other. This approach decreases the time required to collect experimental data because it does not require performing experiments with random initialization of program combinations. Another benefit of such a combination is that when more than one program is running, the neural network can still be able to monitor the system by producing a more confident estimate.

STRATEGY - 4: This strategy is considered to reflect the scenario that multiple cores are active, and the test signal is measured in an environment with a different noise figure. Therefore,

to consider such a scenario, we combine STRATEGY - 2 and STRATEGY - 3. Therefore, the generated input is the noisy version of the weighted sum of different training signals.

After establishing such a training procedure, the CNN model needs to be trained to learn the weights of the layers by applying backpropagation algorithm. Please note that we do not consider the Markovian structure of the problem in the training phase of the CNN. In other words, the training is performed by ignoring the Markov part of the proposed framework. The Markov Model is exploited in the testing phase while we profile the system.

Table I Description of Algorithm 1

Algorithm 1: Overview of the Training Generation Algorithm

Data: the_state_id, the_program_id, $\mathbf{E}_F$

// Generate two matrices based on the data labels.

$\mathbf{M}_0 = \mathbf{E}_F^{\text{the\_program\_id, the\_state\_id}}$

$\mathbf{M}_1 = \mathbf{E}_F^{\text{!the\_program\_id, !the\_state\_id}}$

// $\mathbf{M}_0$ contains all the data that belongs to the_state_id of the program the_program_id.

// $\mathbf{M}_{T_i}$ contains a subset of data belonging to i^{th} program.

// ttt: Generated training signals.

Randomly choose the training data generation strategy

if STRATEGY - 1 *is chosen* **then**

  ttt = $\mathbf{M}_0$

end

else if STRATEGY - 2 *is chosen* **then**

  Choose a random value for the white noise power, σ^2, such that SNR $>$ 10 dB.

  Generate the additive noise components with mean zero and standard deviation σ.

  Assign $\mathbf{N}$ as the matrix that contains these noise components.

  ttt = $\mathbf{M}_0$ + $\mathbf{N}$

end

else if STRATEGY - 3 *is chosen* **then**

  Choose a random value, p, from a uniform distribution.

  Choose data from $\mathbf{M}_1$ to generate matrices $\mathbf{M}_{T_i}$ where $i \in 1, \cdots, N_C$ and N_C is the number of cores. The sizes of $\mathbf{M}_{T_i}$ are exactly equal to $\mathbf{M}_0$.

  ttt = $p_0\mathbf{M}_0 + \sum_{i=1} p_i\mathbf{M}_{T_i}$ where $\sum_{i=0} p_i = 1$.

end

else

  // STRATEGY - 4

  Combine STRATEGY - 2 and STRATEGY - 3:

  ttt = $p_0\mathbf{M}_0 + \sum_{i=1} p_i\mathbf{M}_{T_i} + \mathbf{N}$

end

Result: ttt

3.4.4 Testing While Multiple Cores Are Active

In this section, we introduce our profiling procedure to detect malware. Although the goal is to alert malware, we can extend it further to identify the program which has the malware. In other words, the primary goal of the paper can be restated as malware detection irrespective of the program and the ultimate goal is as detecting which program contains malware irrespective of the number of active cores and programs.

However, multi-core activity causes some drawbacks as opposed to single-core. These drawbacks can be listed as follows:

- Some frequency components can be activated by several cores at the same time and this can result in misclassification and/or false malware alert.
- Activating more than one core causes extra power consumption, thereby increasing white noise signal power. This decreases the SNR of the received signal.

When the malware affects all spectrum, the proposed framework could not identify which program contains malware. However, it still alerts malware because the malware signal distorts the features that all parallel units of the CNN promote. Therefore, once malware is detected and the program with the malware could not be identified, these programs can be executed in a single core to reveal the program with malware.

We summarize the profiling procedure in Algorithm 2. The main intuition behind the algorithm is that the program has to follow an order to execute the states (hot regions) of the

program. Therefore, even if some neurons rather than the expected neuron produce more powerful outputs, we still choose the expected one as long as the value of this neuron is larger than the given threshold. Note that we do not apply any operation to the output layer, i.e., softmax, etc. As an example, we provide output layer values for two states in Figure 20. We only demonstrate these two states for illustration purposes and a similar pattern is obtained for other states as well. Whenever a state is activated, the value peaks at the neuron for the corresponding state and the outcomes for the activated regions are above some threshold. Therefore, we can claim that the program is in a state at a given time if the transition complies with the Markov Model and the throughput is larger than a given threshold. The false firings can be a result of another unrelated activity distorting the actual signal. The switching occurs only if the current state cannot pass beyond the threshold, i.e. there is another state whose value is larger than the threshold and transition from the current state to this state is possible.

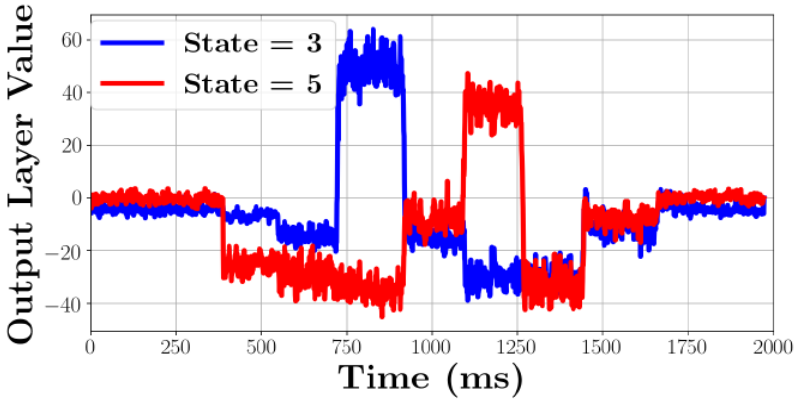


Figure 22 The values of output layers for the states 3 and 5

Because of other computer activities that stall the execution of monitored programs, the CNN output cannot provide accurate estimates and the Markov Model can alert false positives. Although false positives are not as critical as false negatives, they still increase the maintenance cost of the system. Therefore, the algorithm defines two parameters: t_M and t_S. t_M is the mimimum time that the Markov Model needs to report untrusted estimates before announcing the malware. t_S is the minimum time that each state takes. Utilization of these parameters avoids inaccurate transition of states and a false alarm because they prevent inaccurate transitions due to misleading activation of neurons by other software activities. Note here that these parameters also define the sensitivity of the anomaly detection framework. Therefore, they have to be selected cautiously to prevent many false positives/negatives.

Algorithm 2: Profiling Procedure

Data: Θ, $\mathcal{M}_M$, $\mathcal{M}_N$

```
// M_M: Markov Model.
// M_N: CNN Model after training.
// t_M: Threshold for announcing
   malware.
// t_S: Minumum time for execution of
   any state.
// t_L: Threshold for announcing a
   signal belongs to a cluster.
// o_F: Output of M_N.
not_detected = # of states + 1
current_state = not_detected
transitions = []
while true do
    Apply the procedure in Section IV-A to obtain ê_F.
    Feed ê_F to M_N to obtain o_F.
    if current_state != not_detected then
        -Check whether the values of o_F for
         current_state and the state candidates of M_M
         are larger then threshold.
        -Apply softmax function for the states that are
         above the threshold.
        -Choose the most likely state as the next state
         candidate.
        if The chosen state is current_state then
            Append current_state to transitions
        end
        else if the time spent in current_state is larger
         than t_S then
            Update current_state and append to
             transitions
        end
        else
            Append not_detected to transitions
        end
    end
    else
        if o_F[1] is larger than t_L then
            // The program is started!
            Append 1 to transitions
            current_state = 1
        end
        else
            Append not_detected to transitions
        end
    end
    if A program execution is started and the last
     entries of transitions corresponding to the last
     t_M seconds are equivalent to not_detected then
        if Program not ended then
            Alert malware!!!!
        end
    end
end
```

Result: `transitions`

3.5 Tasks 3-5 Basic Block Tracking

In this section we propose TESLA – program Tracing through Electromagnetic Side-channeL Analysis [20]. TESLA exploits device's electromagnetic (EM) emanations to reconstruct detailed (basic-block-level) execution path with high accuracy. For this, TESLA has to overcome the following challenges: 1) train a signal emanation model that associates signal patterns (or signatures) with code segments or sub-paths, and 2) represent the test signal using such signal patterns to reconstruct the program execution path. Specifically, we use a two-step training process that exploits instrumented training to annotate the un-instrumented training signals and identify which signal snippets correspond to which code segments. We also propose a novel signal matching technique that efficiently establishes correspondence between the test signal and the training signals, and exploit this signal correspondence to reconstruct execution path.

The main contributions of this papers are:

- TESLA - a novel framework for zero-overhead and noninvasive program tracing.
- A training process that exploits instrumented executions to annotate un-instrumented training signals.
- An efficient signal matching algorithm that establishes correspondence between the training and the test signals and reconstructs the execution path based on the signal correspondence.
- Empirical evaluations that demonstrate that (1) TESLA achieves high accuracy, and the predicted timestamps are highly precise (2) TESLA can monitor devices with fast

processors and operating systems, and (3) TESLA is able to monitor devices from 1 m distance.

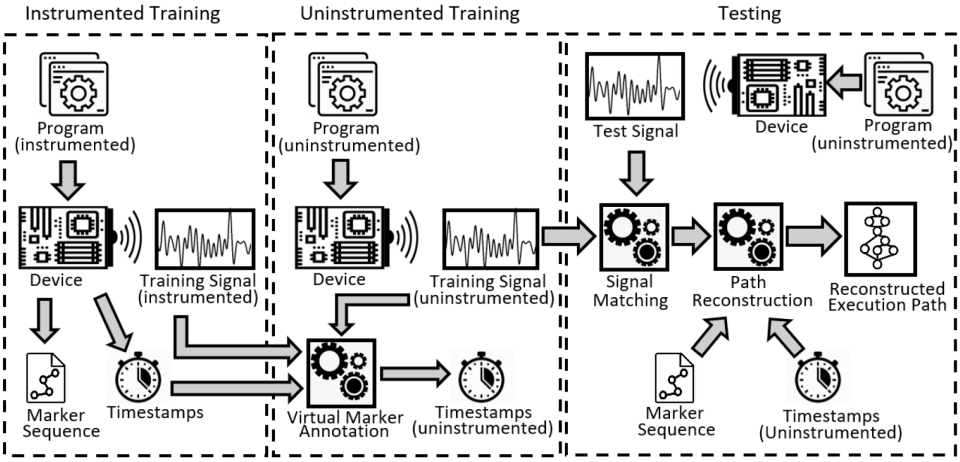


Figure 23 Overview of the TESLA execution path reconstruction framework.

TESLA provides non-intrusive and zero-overhead program tracing by monitoring the EM side-channel signal. A high-level overview of TESLA is demonstrated in Figure 21. In the training phase, TESLA first executes an instrumented version of the program, and records the corresponding EM emanations. The instrumented program also outputs a marker sequence and their execution timestamps that indicate the program execution path. TESLA, next executes an un-instrumented (unaltered) program, and compares the un-instrumented EM signal with the instrumented EM signal to map (un-instrumented) signal fragments to the underlying program sub-paths. We call this mapping Virtual Marker Annotation, as it mimics the markers, however, without any code instrumentation. This process also exploits instrumented markers and timestamps

for efficient and precise signal mapping. Next, in the testing phase, TESLA monitors the EM side-channel signal caused by the execution of the vanilla (i.e., un-instrumented and unaltered) version of the program. TESLA then matches the test signal with the training signals and exploits virtual markers to reconstruct the program execution path. In the following sections, we explain the different steps of TESLA in detail.

3.5.1 Signal Pre-processing: Amplitude Demodulation

To monitor the program execution, TESLA first receives the emanated EM signal through an antenna, performs amplitude demodulation of the received signal, and then digitizes the demodulated analog signal using an analog-to-digital converter. The digitized signal is next scale normalized before any further signal analysis. These pre-processing steps are applied to both training and testing phases.

We have demonstrated that embedded devices emanate amplitude modulated signals at the device's clock frequency [8], [9]. At each processor cycle, the CPU executes instructions, and thus, changes the states of its internal digital circuits (i.e., switches on and off). This causes an instruction dependent current at the CPU clock frequency. Here, the CPU clock acts as the carrier, whose amplitude (i.e., the pulse shape) is modulated by the variations of the executed instructions [21]. As the current flows within the processor, and through the device's printed circuit board (PCB), the device acts as an unintentional and inefficient antenna, and emanates amplitude

modulated EM signal [22]. Thus, to monitor program execution, we demodulate the received signal r(t) at CPU clock frequency f_c.

$$x_a(t) = |r(t) \times e^{j2\pi f_c t}| \tag{10}$$

Here, $x_a(t)$ is the amplitude demodulated analog signal, and t denotes the time. The demodulated signal $x_a(t)$ is then passed through an anti-aliasing filter with bandwidth B, and sampled at a sampling period T_s.

$$x_d(n) = x_a(nT_s) \tag{11}$$

Here, $x_d(n)$ denotes the sampled signal at sample index n. The anti-aliasing filter cancels unwanted signals with frequencies beyond $f_c \pm B$. Note that, the sampling period T_s is determined by the well-known Nyquist criterion $\frac{1}{T_s} > 2B$. Next, we scale normalize $x_d(n)$.

$$x(n) = \frac{x_d(n)}{max(x_d(n))} \tag{12}$$

Scale normalization ensures that the system is robust against changes in amplitude of the EM signals (e.g., due to change in the antenna's distance, position, etc.). The scale normalized signal x(n) is then used for further signal analysis by TESLA in the training and testing phases.

3.5.2 Instrumented Training

TESLA is first trained with instrumented program executions and their corresponding EM emanations. We execute an instrumented version of the program, in which the source code is instrumented by inserting markers (i.e., special probe functions) at selected program locations. Each marker has a unique identification number (ID) that identifies its position in the program's control-flow-graph (CFG). The marker function execution records the marker ID along with the execution timestamp. Thus, the markers perform as program execution checkpoints that partition the CFG into smaller code-segments which we refer to as marker-to-marker code-segments or sub-paths.

We insert these markers in strategic program locations. The marker insertion is dictated by the following criteria: (1) any program execution control-flow path must be uniquely and unambiguously represented by a sequence of marker-to-marker sub-paths, and (2) all marker-to-marker sub-paths must be acyclic and intra-procedural. Based on these criteria, we inserted markers in the following code locations: entry and exit nodes of functions, loop heads, and target nodes of go-to statements.

CFG partitioning helps to provide training coverage for program execution. Specifically, any practical program has a large number of feasible program execution paths. In fact, due to cyclic paths in CFG, programs can have infinite number of unique execution paths. Hence, it is neither practical nor possible to provide training for all unique execution paths for any practical program.

However, the markers enable us to represent any execution path as a concatenation of marker-to-marker sub-paths. Thus, instead of providing training for all unique execution paths, we provide training that covers all marker-to-marker sub-paths. Note that the number of marker-to-marker sub-paths is limited and can be exercised using a relatively fewer number of strategic executions.

The markers also enable us to annotate the monitored EM signal. The marker execution timestamps help to establish a correspondence between executed code segments (i.e., marker-to-marker sub-paths) and the signal fragments they generate. It is important to emphasize that while the markers provide an abstract partitioning, the program execution (for both training and testing) follows a single contiguous trace (from program's start to end) and generates a continuous EM signal. Consequently, it is not visually identifiable that which marker-to-marker sub-path generated which part of the emanated signal. So, we use the marker execution timestamps to annotate the start and the end of each marker-to-marker sub-path in the monitored EM signal.

At the beginning of the program execution, we reset the processor's Time Stamp Counter (TSC) to zero. The markers record the TSC values as timestamps, which then indicate the time-interval (in clock-cycles) from the program's start. We convert these timestamps to sample-index using the following equation.

$$n = round(\frac{t \times f_s}{f_c}) \tag{13}$$

Here, n indicates the sample-index corresponding to the timestamp t, f_s is the sampling rate of the monitored signal, and f_c is the clock frequency of the monitored CPU. The rounding operation ensures that the resultant n is an integer value.

We also identify the program's start (i.e., sample-index n = 0) in the signal. To facilitates the automatic detection of program's start (shown in Figure 22), we insert a for-loop just before the beginning of the program execution. The for-loop executes a periodic activity (e.g., increments a loop counter variable) and generates a periodic and identifiable signal pattern. End of this periodic pattern indicates the end of the for-loop (i.e., the start of the program). Furthermore, at the beginning of the program execution, the program is loaded into the system memory. This leads to memory access, which in turn stalls the processor and causes a dip in the signal amplitude [23]. We identify this transition (from the end of for loop to the beginning of program execution) when the moving median of the signal drops below a predefined threshold (as shown in Figure 22). This acts as the reference point (i.e., sample-index n = 0). All markers are then annotated according to their sample-indices.

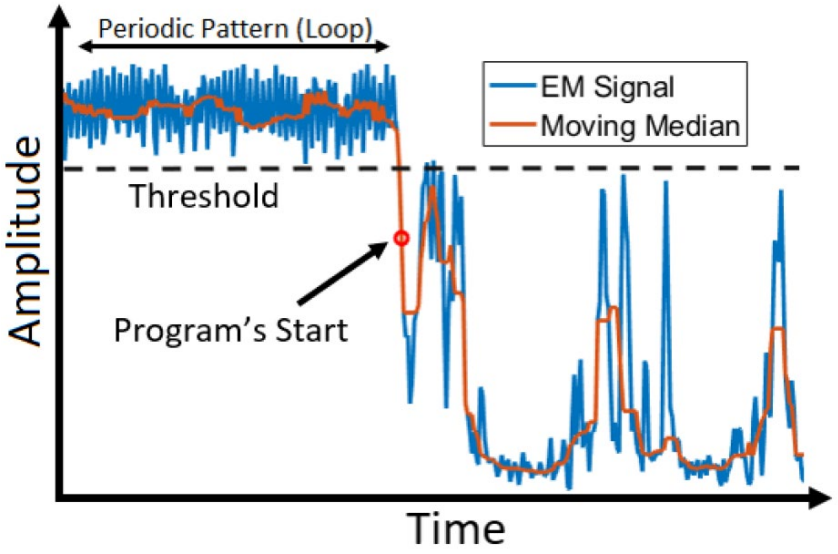


Figure 24 Automatic detection of program's start: the end of the periodic pattern (for-loop) indicates the program's start. We identify this when the moving average of the EM signal drops below the threshold

Figure 24 shows an annotated instrumented signal with each marker represented by a vertical red line. Markers m_0, m_1, m_2, and m_3 are placed at sample-index n_0, n_1, n_2, and n_3 respectively. The marker ID sequence (e.g., m_0, m_1, m_2, . . .) indicates the program execution path, while execution timestamps or sample-index sequence (e.g., n_0, n_1, n_2, . . .) identify the start/end of the marker-to-marker code-segments. Thus, marker annotation establishes a correspondence between code-segments and emanated EM signal snippets. For instance, in Figure 24, the signal snippet between sample-index n_0 and n_1 corresponds to the execution of the code-segment or subpath between marker m_0 and m_1.

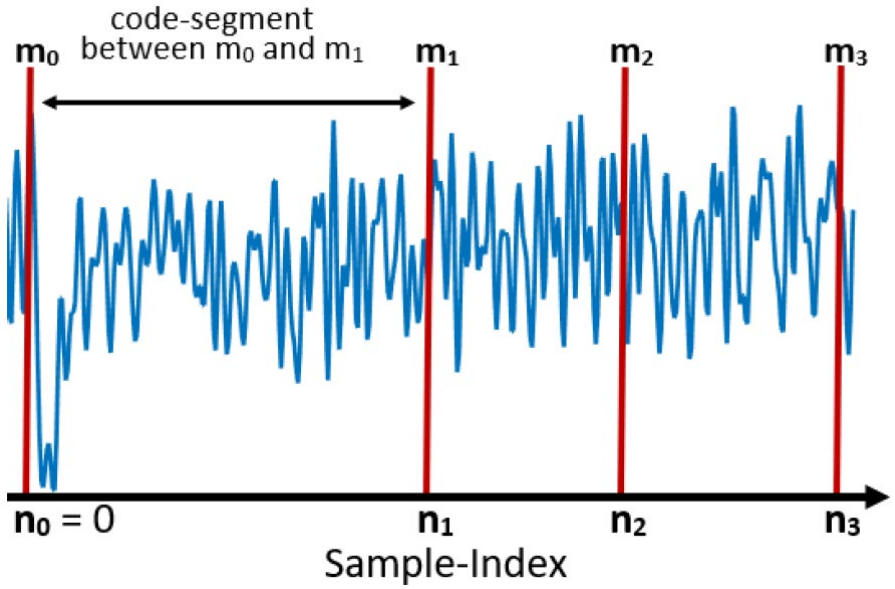


Figure 25 Markers (vertical red lines) are placed on the signal according to their execution timestamps. The signal snippet between two consecutive markers corresponds to the EM emanation from the marker-to-marker code-segment

While the instrumented training helps us to partition the CFG and to annotate the signal, the instrumentation alters the original signal emanation patterns. Thus, the instrumented training signals and corresponding signal emanation models cannot be used in the testing phase, in which the device executes a vanilla version (i.e., unaltered and un-instrumented) program. Specifically, the instrumentation adds overheads to the original program (i.e., function calls that record marker ID and execution timestamps). The execution of these overhead codes (i.e., marker functions)

requires additional computation (and computational time), and in turn, causes extraneous EM emanations that are irrelevant to the original program.

To evaluate the impact of instrumentation, we investigate the EM signature of the marker functions. We identify the marker functions in the instrumented signal using their timestamps. We then crop out and compare these signal snippets. Figure 24 overlays 100 signal snippets corresponding to the marker function execution. We observe that the marker functions emanate very similar signal patterns. This is expected as the marker functions execute the same code segment. However, the beginning and the end of the marker signals demonstrate a marked variability. This is due to the microprocessor's instruction pipeline architecture that overlaps multiple instructions during execution. Thus, the processor's EM emanation at any instance depends on all instructions that are moving through different stages of the pipeline, rather than just one single instruction. Consequently, the execution of the same marker function may demonstrate signal variability towards the beginning and the end of the function call depending on the variations in the preceding and the following code segments (i.e., other instructions in the pipeline).

Likewise, the marker functions themselves also affect the EM emanations of the adjacent code segments. In the instrumented execution, all marker-to-marker code-segments are separated by marker functions. As such, the EM emanation patterns from the code-segments are altered at the boundaries due to the "cross-over" effect from the marker functions. For larger code segments (e.g., consisting of a few hundred instructions), the duration of the emanated signal is much larger

compared to the altered boundaries. Thus, the impact of the instrumentation is trivial. However, for smaller code segments such as basic blocks consisting of only a few instructions, instrumentation can alter the overall signal emanation pattern significantly. Thus, cropping out the marker signal-snippets from the instrumented signals would not replicate the un-instrumented signals. Therefore, we exploit un-instrumented executions to create better signal emanation model.

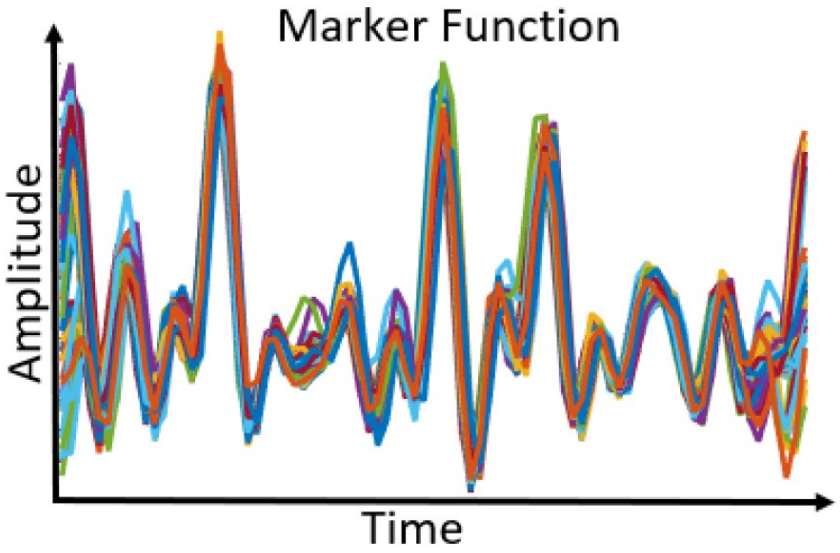


Figure 26 EM signals corresponding to marker function execution

3.5.3 Uninstrumented Training

TESLA is next trained with uninstrumented program executions. However, before we can use the uninstrumented training for program execution monitoring, we must first annotate the signal. Unlike the instrumented executions, the uninstrumented executions do not have markers or

timestamps. Thus, we cannot directly annotate or identify which signal snippet corresponds to which code segment. Instead, we compare uninstrumented execution with the instrumented execution to identify and demarcate the marker-to-marker code segments in the signal. We call these demarcations "virtual markers" as they play the same role as the marker functions, albeit, without adding any overhead code or altering the original program or its signal emanation patterns.

Virtual Marker Annotation: In the instrumented execution, code segments are separated by marker functions. Each marker function execution records a pair of information m and t, where m represents the marker ID that indicates the execution point in the CFG, and t is the execution timestamp. We then convert the timestamp t to its equivalent sample-index n (using equation 13). If the program executes k marker-to- marker code-segments, the instrumented execution records a sequence of k + 1 markers (including the starting and the ending markers). Thus, instrumented execution outputs a maker ID sequence $M = \{m_0, m_1, m_2, ..., m_k\}$ and corresponding sample-index sequence $N = \{n_0, n_1, n_2, ..., n_k\}$, with M uniquely identifying the program execution path, and N indicating which signal snippet corresponds to which code segment. Thus, the task of virtual marker annotation is to generate marker ID sequence M' and sample-index sequence N' for the uninsrumented training signal, without actually using instrumentation or marker functions. To annotate the virtual markers, we execute the instrumented and the uninstrumented programs with the same input. Thus, the executions follow identical paths through CFG (i.e., execute the same marker-to-marker code segments in exact same order). This ensures the marker ID sequence is

identical for instrumented and uninstrumented executions (i.e., $M' = M$). However, due to the overhead computations (i.e., marker functions), the timestamps or sample-indices for the virtual markers are significantly different (i.e., $N' \neq N$).

To estimate the virtual marker sample-index sequence N', we compare the uninstrumented and instrumented EM signals (Figure 25). We notice that the execution of the same code segment (e.g., m0-m1-m2) requires more computational time in the instrumented version due to the marker function overheads. Thus, we estimate the sample-indices for the virtual markers by adjusting for the overhead computational time using the following equation

$$n'_i = n'_{i-1} + (n_i - n_{i-1} - n_{oh}) \quad \text{for } i \in \{1, \ldots, k\} \tag{14}$$

Here, n'_{i-1} and n'_i indicate the sample-indices for the (i-1)-th and i-th virtual marker in the uninstrumented signal, n_{i-1} and n_i indicate the sample-indices for the (i-1)-th and i-th marker in the instrumented signal, and n_{oh} is the overhead computational time (in samples) for the marker function execution. Thus, ($n_i - n_{i-1} - n_{oh}$) is the overheadsubtracted execution time for the i-th code segment. Note that, sample-index $n'_0 = 0$ (indicating the starting point of the program), and we iteratively estimate $n'_1, n'_2, \ldots, n'_k$.

While Equation 14 gives a good initial estimation for the virtual marker annotation, it does not account for the execution-to-execution hardware variabilities such as cache hits or misses that may lead to variabilities in computational time. To mitigate this issue, we fine-tune the initial sample index estimations by matching uninstrumented signal with its instrumented counterpart.

First, we identify the signal snippet corresponding to a given code segment in the instrumented signal using its timestamps.

Let x(n) be the instrumented signal with n indicating its sample-index. Thus, the signal snippet between sample index $n = n_{i-1}$ and $n = n_i$ corresponds to the i-th code segment (i.e., the subpath between markers m_{i-1} and m_i). We then exclude or crop-out the first n_{oh} samples from this signal snippet as they correspond to the marker function not the original code segment. In Figure 25, the dotted lines indicate the correspondence between the uninstrumented and instrumented signals. Thus, this overhead-subtracted signal snippet $s_i(n)$ acts as the EM signature or template for the code segment. We search for this signal template by sliding it across the uninstrumented signal $y(n')$. We limit our search within $\pm d$ samples of the initial estimations (i.e., between sample-index $n' = n'_{i-1} - d$ to $n' = n'_i + d$). This makes the search computationally efficient, and also helps to avoid false signal matches. At each search position, we compute the Euclidean distance between the template and the uninstrumented signal. We then choose least Euclidean distance match for updating the initial estimations

$$\hat{l} = \arg\min_{l} e(l) \quad \text{for } l \in [-d, d] \tag{15}$$

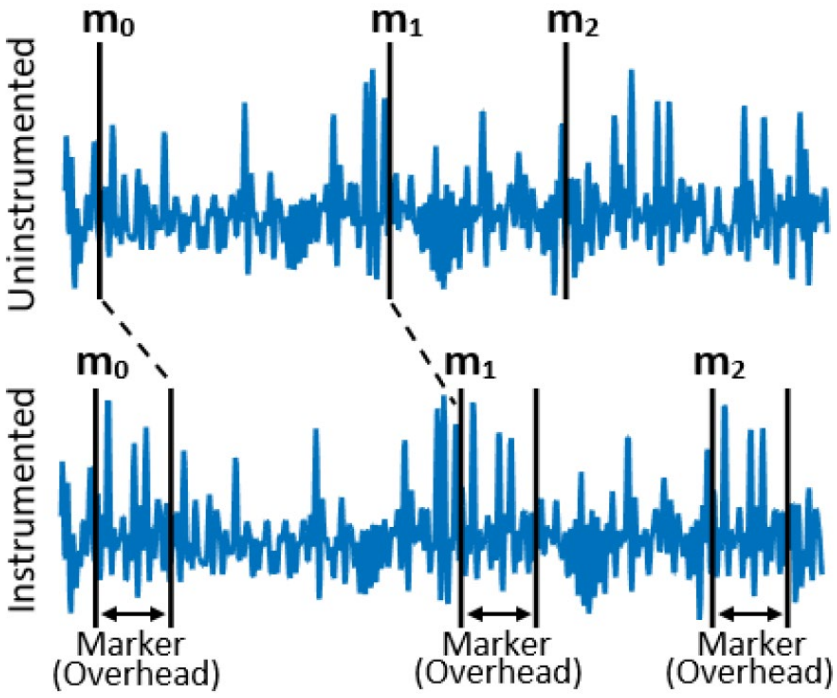


Figure 27 EM signals corresponding to the uninstrumented (top) and the instrumented (bottom) program executions. The dotted lines indicate the correspondence between the uninstrumented and the instrumented signal

Here, $e(l)$ is the Euclidean distance between the template $s_i(n)$ and the uninstrumented signal $y(n')$, and l indicates the shift from the initial estimated n'_i. Thus, $\hat{l}$ is the shift corresponding to the best match. Finally, we update initial estimated n'_i using the following equation

$$n'_i := n'_i + \hat{l} \tag{16}$$

This iterative process is depicted in Algorithm 1.

Input: $x(n)$, $y(n')$, $N = \{n_1, n_2, \ldots, n_k\}$
Output: $N' = \{n'_1, n'_2, \ldots, n'_k\}$
initialization: set $n'_0 = 0$;
for $i := 1$ ***to*** k **do**
    Estimate n'_i (Eq. 5);
    Find best match (Eq. 6);
    Update n'_i (Eq. 7);
end

Algorithm 1: Virtual marker annotation.

3.5.4 Program Execution Monitoring

To reconstruct the program execution path, TESLA compares the device's EM emanation with the (uninstrumented) training signals and predicts the control-flow execution path. The path prediction involves two steps. In step 1, we match the monitored signal with the training signals

to establish a signal correspondence. In step 2, we exploit this signal correspondence to predict program execution path by using the training signal annotations (i.e., the virtual markers). We discuss these steps with further details in the following paragraphs.

Signal Matching: To establish signal correspondence, we match fixed-length windows from the monitored signal against the training signals, and then adjust the window-size according to the signal similarities. The signal matching process is demonstrated in Figure 26. First, we extract a fixed-length initial window W of size L from the monitored signal. We then slide W across all training signals to find the best (i.e., the least Euclidean distance) match. This establishes a window-to-window signal correspondence (shown with a dashed arrow in Figure 26). Next, we compare the samples that follow these windows. In Figure 26, the initial window and its subsequent samples are overlaid on the matched window and its subsequent samples using red dots. We then iteratively extend the signal correspondence as long as the overlaid monitored signal is similar to the underlying training signal. Specifically, in each iteration, we compare the D subsequent samples and compute the sample-to-sample squared difference. If the mean squared difference is below a predefined threshold θ, we update the matched window size: L = L + D, and keep comparing the next D samples. Otherwise, we terminate the window extension process. We then again extract the next unmatched window from the monitored signal, match it across all training signals, and adjust the window-size. This process goes on until we establish signal correspondence for the entire monitored signal.

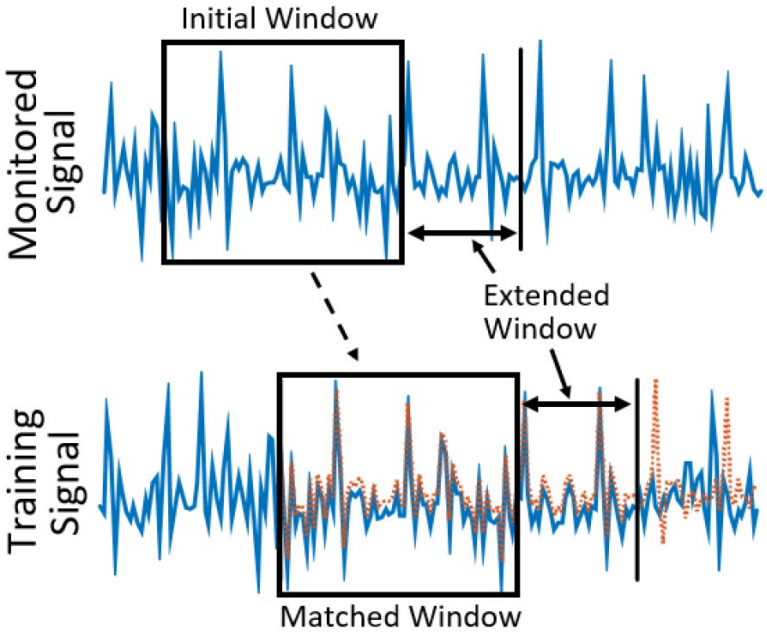


Figure 28 Signal matching process: dashed arrow indicates the correspondence between fixed-length windows in monitored and training signals. Window size is extended based on signal similarities, up to the point where the training signal (blue line) starts to deviate significantly from the monitored signal (overlaid red dots).

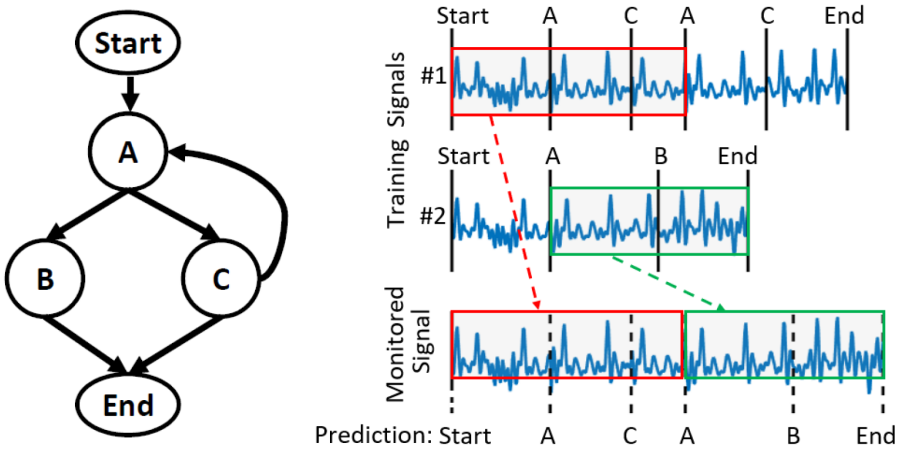
This approach for signal matching is computationally more efficient than that of multiscale signal matching, in which multiple windows of different sizes are simultaneously matched against the training signals. In contrast, we initiate the search using a small fixed-sized window, and then gradually extend the window size. Furthermore, the time complexity for the window search is directly proportional to the window size L. Thus, smaller window leads to faster search. However, if the window is too small, the match becomes unreliable. Therefore, in our experiments, we

choose L = 64. In addition, smaller values for D enable finer adjustment of the window size. However, too small a value for D may lead to early termination of the window extension due to a few noisy samples. In our experiments, we use D = 8.

Path Reconstruction: We next exploit the correspondence between the monitored and the training signals to reconstruct the execution path. Figure 27 demonstrates the path reconstruction process with a simplified example. On the left (Figure 27a), we have the program CFG where the nodes represent the markers and the edges represent the marker-to-marker subpaths. The training signals and the monitored signal are shown on the right (Figure 27b). The (virtual) markers are annotated on the training signals with vertical black lines, and indicate that training signal 1 corresponds the program path $Start - A - C - A - C - End$, while training signal 2 corresponds to $Start - A - B - End$. Note that, for this simple CFG, these two training executions are sufficient to provide coverage for all marker-to-marker subpaths (i.e., edges on the graph). However, most applications often require a large number of executions (e.g., hundreds or even thousands) for high code coverage.

Furthermore, we indicate the correspondence between the monitored and the training signals using color-matched windows and dashed arrows. For instance, the red windows in the training and the monitored signals demonstrate similar signal patterns, and so do the green windows. This signal correspondence enables us to reconstruct the monitored signal by concatenating matched-windows (e.g., red and green windows) from different training signals.

More importantly, the signal similarity or correspondence implies that the matched windows correspond to the same program subpath. Thus, we reconstruct the program execution path for the monitored signal by concatenating the program subpaths corresponding to the matched training windows. For instance, the red window (in training signal 1) corresponds to the program subpath $Start - A - C - A$, and the green window (in training signal 2) corresponds to the program subpath $A - B - End$. Therefore, we concatenate these subpaths to reconstruct the execution path. In Figure 27, the reconstructed execution path ($Start - A - C - A - B - End$) is indicated with dashed vertical lines.



(a) Program control-flow graph (b) Program execution path prediction using signal correspondence.

Figure 29 Execution path reconstruction exploiting signal correspondence between training and test signals

3.6 Task 6 Single Instruction Tracking

To enable instruction-level tracking when noise is present, we propose a new framework: PITEM (Permutations-based Instruction Tracking via Electromagnetic Side-channel Signal Analysis) [24]. This framework consists of two major steps: 1) identifying groups of instructions that are referred to as instruction types that have similar EM signatures, 2) tracking all possible orderings, i.e. permutations, of these instruction types and therefore, monitoring program flow at instruction type granularity. By generating all possible permutations of the instruction types, we generate instruction sequences systematically and the EM signatures for the sequences address the pipeline effect to a great extend as they represent the overall EM emanations for longer periods of time. Also, this method is not limited to devices with lower clock frequencies. Therefore, this framework is applicable in program activity tracking applications even for devices with complex processor architectures and higher clock frequencies.

Furthermore, this framework can be used in finer granularity malware detection applications. By using the permutations as reference signals, modifications that are made at instruction level can be resolved. In applications where the program is expected to execute one of the allowable instruction sequences, this technique can determine any unexpected instruction executions and resolve at what point the modification has been made. Note that, unlike EDDIE [10] and Remote [12], this method does not rely on per-iteration time of loops and it can detect the changes when the ordering of the instructions are changed. The malware detection performance of

PITEM is not tested on a real-life malware application, but the ability to resolve different orderings of instructions indicates its potential to finer-granularity malware detection application extension.

To show the feasibility of the proposed method, we perform testing on two devices with different architectures and operating clock frequencies. These devices are Intel's DE1 Altera FPGA Board with Altera NIOS-II (soft) processor [25], and A13-OLinuXino with ARM Cortex A8 processor [26], operating at 50 MHz and 1 GHz clock frequencies, respectively. The results are reported for different experimental setups. We note that single execution of the permutations of the instruction types can be detected with as high as 92.8% accuracy. As the number of successive executions of the permutations increase, the detection accuracy also increases to as high as 100%. We also test the performance of the proposed method with different signal-to-noise ratio (SNR) levels. We note that the system performance is stable for SNR levels higher than 15 dB. Finally, we test the limits of the system by tracking permutations of instructions from the same instruction type. As expected, the detection performance for single execution of the permutation is very low, but it increases significantly (to as high as 92.4% and 98%) when the permutation is repeated.

3.6.1 Determining Instruction Types by Using EM Side-Channels

The proposed methodology is based on analyzing EM side-channel signals. As mentioned earlier, EM side-channels are created as a by-product of fast switching currents flowing through transistors during program execution. Therefore, the execution of certain instructions generates

distinct EM signatures. In this section, we describe the procedure to identify these instructions and call them instruction types. Since different architectures are implemented differently on micro-architecture level, these instruction types differ for different architectures. Fig. 28 presents an outline of the procedure step by step. These steps are explained in detail in the following sections.

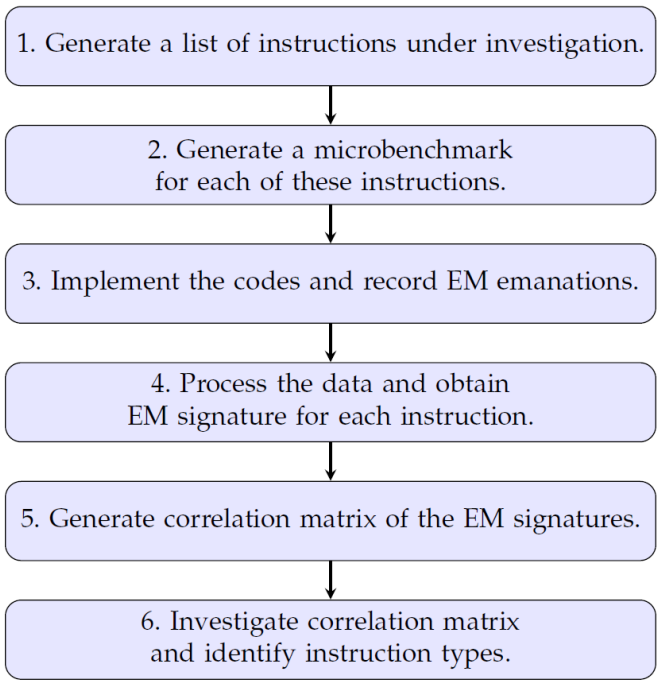


Figure 30 Flowchart of determining instruction types.

3.6.2 Generating List of Instructions Under Investigation

This step includes examining the available instruction set for the given processor and selecting the desired and applicable instructions to investigate. The applicability of the instructions is based on the micro-architecture of the processor.

3.6.3 Generating Micro-benchmarks for Instructions

After instruction selection, we generate a microbenchmark for each instruction whose pseudo-code is given in Fig. 29. One should note that this work uses an instrumented measurement setup where an input/output (I/O) pin is set to high voltage before the code under observation and reset to low voltage after the code under observation finishes. Therefore, the input/output pin signal is used to find the starting and ending points of the region of interest.

For most processors, the EM signature difference caused by a single instruction does not generate a distinct variation, therefore, the instruction under interest is repeated N times to magnify the EM signature. Note that this repetition is realistic because the pesudo-code structure is only used in determining instruction types step and it is not utilized in any testing scheme. The starting and ending markers are preceded and succeeded by two empty loops, respectively. This for-loop structure allows for a fair comparison since they make sure that the same instructions are pipelined before and after all instructions under interests.

```
while true
   for
      %empty for loop
   end

   % Set I/O Pin to High

   for N times
      %Instruction Under Interest
   end

   % Reset I/O Pin to Low

   for
      %empty for loop
   end
end
```

Figure 31 Pseudo-code for instruction type detection setup

3.6.4 Implementing the Codes and Recording EM Emanations

After microbenchmark implementation, EM emanation measurements are performed. The measurement includes two synchronized channels: 1) EM emanation signal, 2) I/O pin signal. For better localization, a near-field antenna with proper antenna gain and size should be utilized. The choice of the antenna size is based on their ability to capture relevant EM emanations from the processor and reject the interference from other parts of the device.

3.6.5 Data Processing to Obtain EM Signatures

In this step, the recorded EM emanation and I/O signals are processed to obtain EM signatures of each instruction. Fig. 30 presents an overview of data processing flow. First, the input I/O signal is filtered with a moving median filter to overcome overshooting. Then, the signal is normalized to account for possible DC offset. Next, the amplitude values are quantized to their binary representations by using a 3 dB threshold. The binary stream of data is smoothed by removing outliers and the starting and ending points are determined.

The input EM signal contains unintentional EM emanations hat are amplitude modulated (AM) to the periodic signals present on the board [22]. Among these modulations, the one around the first harmonic of the operating clock frequency is the strongest and the most informative. Therefore, the input EM signal is firstly down-converted with clock frequency, and then, low-pass filtered to reject interference from other modulating periodic signals and reduce the measurement noise at higher frequencies. Since the modulation around the clock frequency is not necessarily conjugate symmetric in the frequency domain, the resulting signal can be complex-valued. The amplitude of this complex-valued signal contains the shape information whereas the phase carries relative time shift information. Since our objective is to determine the shape, we proceed the data processing by only keeping the magnitude. Finally, the processed EM signal is cropped into chunks by using the cropping points obtained from input I/O signal, and the EM signatures are generated. Note that these signatures represent the EM waveforms generated from the repetition of instructions N times.

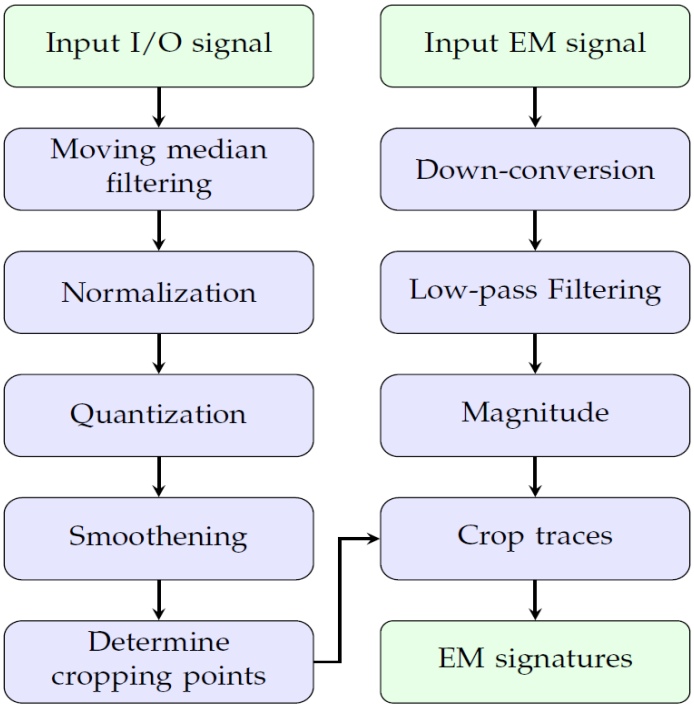


Figure 32 Flowchart of data processing

3.6.6 Generating Correlation Matrix

In this part, we generate a correlation matrix that represents the similarity between the generated EM signatures. The degree of (dis)similarity between two time domain signals can be measured in several ways: L1 norm, L2 norm, and cross-correlation etc. The power consumed by the devices fluctuates during run-time and this creates a DC offset in the measured EM emanations. Since L1 norm and L2 norm measure the sum of point-wise distances, it is an error-prone measure in the presence of DC offsets. Therefore, these distance measures are not suitable without normalization. On the other hand, cross-correlation measures the similarity of the waveforms,

which is a suitable similarity metric for our purposes. Note that the execution of some instructions take longer than the others and the corresponding EM signatures are longer in length. To account for different EM signature lengths, while correlating two signals, the correlation is performed by sweeping the longer waveform with the shorter one and the highest correlation is denoted as the cross-correlation of these two waveforms. Cross-correlation is performed for all EM signature pairs to generate the correlation matrix.

3.6.7 Identifying Instruction Type

The objective of this step is to find the groups of instructions that have similar EM signatures. We refer to these distinct groups that have similar EM signatures as instruction types. The correlation matrix of the EM signatures shows how much the signatures are correlating with each other. A subjective method to find the instruction types from the correlation matrix is visual inspection. However, this approach is prone to misclassifications due to its subjective nature. As an objective method, we propose to utilize hierarchical (agglomerative) clustering. This clustering technique is a bottom-up algorithm that starts with treating each sample as a separate cluster and merges these clusters pair-wise until all samples are merged into a single cluster [27]. As the clusters are successively merged, a cluster tree (dendrogram), which is sequence of clusterings that partition the dataset, is generated [28]. Unlike other clustering algorithms such as K-means, this method does not require a random centroid initialization or a prior cluster number. This is especially useful in our application since we do not have an apriori knowledge on the number of

instruction types. We can decide for the number of clusters by observing the dendrogram. The main disadvantage of this algorithm is its large time complexity ($O(N^2\ log(N))$), and space complexity ($O(N^2)$). Since the number of instructions under investigation is typically not a large number, the dataset size is relatively small, therefore, the time and space costs are affordable. The output of the correlation matrix presents the similarity measure between the signatures. However, hierarchical clustering is based on the (dis)similarities between samples. Therefore, by using [29], we convert the cross-correlation values, ρ, to distance values, d, as follows,

$$d = \sqrt{2(1-\rho)}. \tag{17}$$

3.6.8 Detecting Permutations of Instruction Types

This section explains a systematic way of tracking the execution of instruction types. In previous sections, instruction types are identified by repeating the same instruction for several times and clustering them based on the similarity of their EM signatures. Next step is to detect these instruction types in a testing scenario. The most straightforward way to do so is to find the EM signature of the instruction types when they are executed once instead of N times, and use these signatures as dictionary while testing. However, this approach has two major drawbacks:

1. Most processors implement pipeline architecture where the execution of the instructions is divided into different stages. This allows for overlapping executions of consecutive instructions at different stages. Since all these stages utilize transistor switching activity

during their operation, all stages behave as an EM emanation source. Therefore, the measured EM emanation by the antenna is a combination of the EM waves radiated from different stages. Due to the lack of a complex model to decouple these combinations, it is not possible to isolate the EM signature of a given instruction.

2. The variation created by the execution of a single instruction is generally not very strong. Hence, the EM signatures for different instructions are very similar to each other, which consequently leads to the inability of tracking these instructions.

Previous work in literature suggests to generate EM signatures for instruction sequences rather than single instructions and reports high self-correlation and low cross-correlation values for several instruction sequences to show the applicability of the proposed system [19]. However, the instruction sequences used in this work are relatively long and the choice of these instruction sequences is not systematic.

Instructions in a program appear in different orders. We propose to generate the sequences in a systematic way by generating all possible orderings of the instruction types. In other words, we propose to generate EM signatures for all permutations of the instruction types and track these permutations. Note that this approach addresses the aforementioned problems for the following reasons.

1. By generating EM signatures for the permutations, we observe the overall effect of the permutation block. Certain interactions caused by different orderings of these

instructions and the impact of the pipeline are embedded into the EM signature. Although it is not possible to isolate the impact generated by each of these instructions, we obtain an EM signature that covers their aggregate impact.

2. By using permutations instead of single instructions, the EM emanation variation becomes stronger and the length of the EM signature waveform gets larger. Hence, we obtain a more informative signal that enhances the testing accuracy.

One should keep in mind that inclusion of permutations is very helpful to address the pipeline effect but there are a few issues that cannot be addressed with this scheme:

- The EM signature still experiences the pipeline impact in the beginning and at the end due to the instructions that come before and after the permutation, respectively.
- For some processors, the pipeline length might be longer than the length of the permutation, and this limits the capability of the permutation to represent the emanation coming from all pipeline stages.

Finally, note that the number of the permutations is given by K!, where K is the number of clusters or instruction types that are obtained in Section 2.6.7. For large K, the number of permutations becomes a large number which leads to a costly measurement and large memory usage. Therefore, the choice of K from the cluster tree should be made carefully.

The steps of detecting permutations of *instruction types* are explained step-by-step as follows.

3.6.9 Picking an Instruction to Represent Each Instruction Type

Since the instructions from the same instruction type have similar EM signatures, one instruction from each type is chosen to represent their types. Instruction types are labelled with the first K capital letters of the alphabet.

3.6.10 Generating Microbenchmarks for Permutations

As mentioned earlier, with K clusters, we need to generate K! benchmarks that include all permutations. For example, if there are 4 identified instruction types (A, B, C and D), the permutations should include: ABCD, ABDC, …, DCBA.

Execution of most programs and embedded systems go through loops during the operation. These loops include execution of the same instruction sequences several times. Furthermore, program spends most of the execution time in functions that are called many times successively. Considering this repetition-based nature of the most programs, we propose to investigate the

impact of repetition of instruction blocks on the tracking performance. In particular, we create EM signatures for different repetitions of the same instruction block. For ease of reference, we use the notation $(ABCD)_N$, where (ABCD) is the investigated permutation block, and N is the number of permutation block repetition. Note that the ultimate goal is to track permutation blocks with N = 1. However, due to the repetitive structure of code implementations, N could be different than 1. For example, a certain permutation might appear within a loop that is repeated several times and this apriori knowledge can be used to improve tracking performance. The pseudo-code for the new microbenchmark structure is shown in Fig. 31.

```
while true
  for
    %empty for loop
  end

  % Set I/O Pin to High

  for N times
   -First instruction in permutation order,
   -Second instruction in permutation order,
         .
         .
         .
   -Kth instruction in permutation order.
  end

  % Reset I/O Pin to Low

  for
    %empty for loop
  end
end
```

Figure 33 Pseudo-code for permutation detection setup.

3.6.11 Implementing Code and Recording EM Emanations

After implementing the pseudo-codes, EM emanations from the device is measured. Two measurements are taken for each microbenchmark at different times. The waveforms obtained in the first measurement are used for training, whereas the second measurement waveforms are used for testing.

3.6.12 Training: Generating Templates for Each Permutation

In this part, we generate templates for each permutation and these templates are used in the testing phase for prediction. A general overview of training can be found in the left hand part of Fig. 32. Several EM signature traces are obtained for each permutation from Measurement 1 recordings. These traces from the same permutation are aligned using cross-correlation. Then, they are cropped so that all of them have the same length. Finally, the EM template for the corresponding permutation is generated by using the point-wise average of the aligned and cropped EM signature traces.

3.6.13 Testing: Predicting Instruction Sequences Using Templates

A general overview of testing can be found in the right hand part of Fig. 32. Testing traces are labeled with their permutation order. The prediction step implements a matched filter-like structure where the filters are the normalized versions of EM templates. We use such a structure because matched filter is the optimum receiver in terms of maximizing the signal-to-noise ratio (SNR) when the received signal is corrupted by additive random noise. Another advantage of such a structure is that the matched filter finds the best offset between the received signal and the templates on its own without requiring synchronization. After correlating the testing trace with all filters, the permutation of the trace is predicted as the template whose corresponding filter gives the highest correlation.

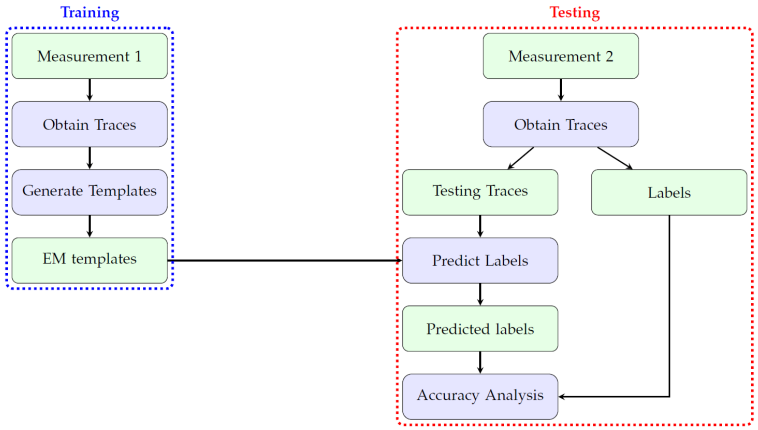
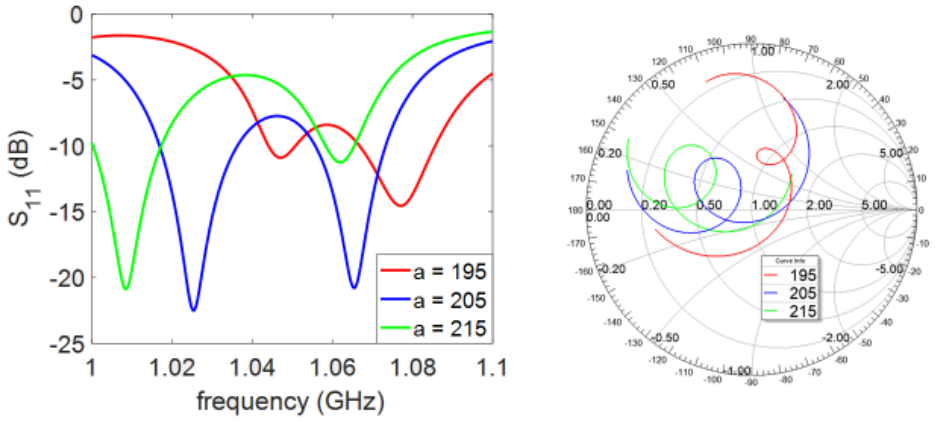


Figure 34 Overview of training and testing

4.0 RESULTS AND DISCUSSION

4.1 Task 1.1 - Results

Figure 34 (a) shows the reflection coefficient vs. frequency with lower disc radius, *a,* as parameter. With an increase in radius, the resonances shift to lower values. Furthermore, Figure 34 (b) shows less coupling between two resonances due to smaller loop sizes in the smith chart, which increases with the disc radius. When the disc radius is 205 mm, the impedance is matched for the band. Slot length, *l,* for this case is 113 mm, which is also the selected length for the fabrication.



(a) (b)

Figure 35 (a) Reflection coefficient vs. frequency and (b) Impedance loci variation with lower disc radius a as parameter

The simulated radiation patterns in the E & H-plane for four frequencies in the band of interest are shown in Figure 35 (a) and (b). The peak gain is above 18 dBi in the whole frequency band with a maximum value of 19.1 dBi at 1.03 GHz. The maximum cross polarization level in H-plane is ~ 20 dB below the main lobe in the entire frequency band. We observe in the simulations that with the increase in frequency, the H-plane sidelobe increases from -10.2 dB at 1.01 GHz to -7.7 dB at 1.04 GHz. This is because at the higher frequencies of the band the array spacing becomes larger and hence results in increased SLL. In the E-plane, the beam is shifted 2° from the maximum at 1.03 GHz.

The antenna geometry shown in the Figure 2 was designed, fabricated and tested. The center frequency of the designed antenna is 1.03 GHz. A square aluminum sheet of dimension 1.04 m was used as a ground plane. The individual discs have the radius of 20.5 cm with the slot length and width of 11.3 cm and 1 cm respectively. Each of them is fabricated using aluminum sheet of thickness 2 mm. The center disc is suspended at 5 mm above the ground plane while the other four are at 10 mm above the ground plane. The center disc is directly fed by a 50 Ohm coaxial probe, which is placed at 50 mm away from the center. The fabricated antenna picture is shown in Figure 36.

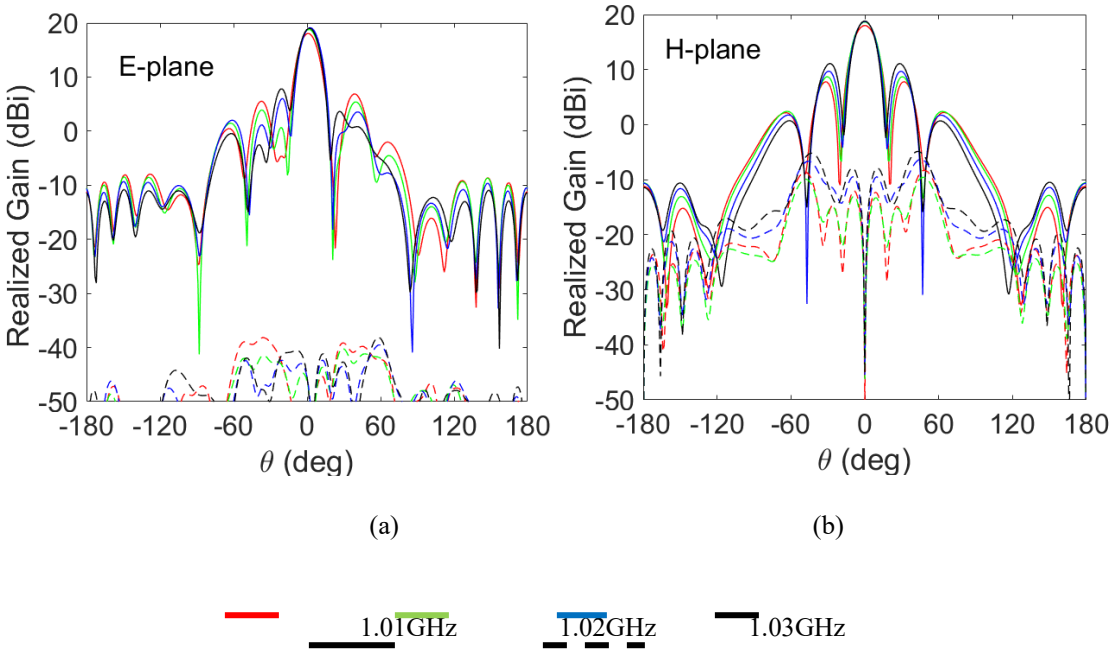


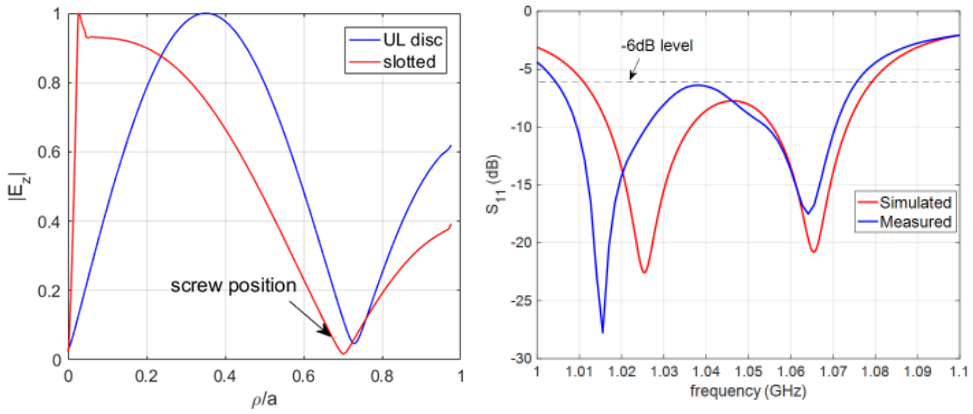
Figure 36 Radiation pattern over the band for the antenna geometry shown in Fig. 2 at (a) E-plane, (b) H-plane

4.1.1 Antenna Fabrication and Measurements

Each disc is suspended using four Teflon screws. Modal electric field distribution of the TM_{12} mode is used to determine the position of screws. To explain this, Figure 37 shows the simulated electric field $|E_z|$ inside the cavity vs. normalized radius, for a single unloaded and slot loaded disc. The $|E_z|$ of unloaded (UL) disc follows the first order Bessel function $J_1(k\rho)$. For UL case, the electric field null is at ~ 0.7a. We have observed that slot loading does not have significant effect on the position of electric field null as shown in Figure 37 (a). Compared to the fundamental mode, this property is an added advantage of TM_{12} mode since nulls in electric field allow us to suspend the patch on the air and hence eliminate the need for the substrate.

(a) (b)

Figure 37 Fabricated antenna (a) front view (b) side view

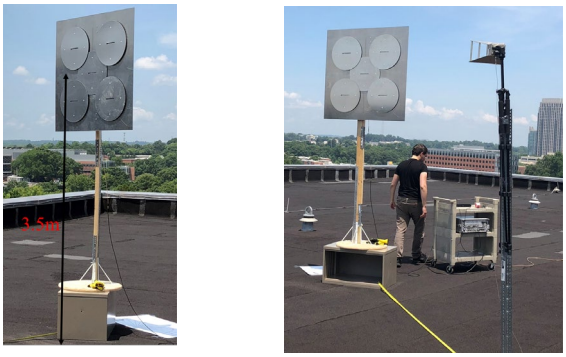


(a) (b)

Figure 38 (a) Simulated cavity electric field vs normalized radius ($\boldsymbol{\rho/a}$) for unloaded and slotted disc operating in TM12 mode (b) Comparison of simulated and measured S11 as a function of frequency

Figure 37 (b) shows the simulated and measured reflection coefficient for the antenna shown. The difference between the measured and the simulated resonant frequencies is less than 1%. The measured $S_{11} \leq -6$ dB bandwidth is 6.7% or 70 MHz. It covers the required bandwidth for the side channel EM detection (shown later in section 3.5). Figure 38 (a) & (b) shows the

mounted antenna picture and the measurement set up to measure the near field and far field patterns of the proposed antenna. The proposed antenna is used as a receiving antenna while the transmitting antenna is a standard broadband double ridge waveguide horn shown in the Figure 38 (b). A digital protractor was used to measure the angle of rotation. The antenna patterns both near field and far field were measured at the roof top of Tech Square Research Building at Georgia Institute of Technology. The measurements were done for 3m, 5m (near field) and 15 m (far field) distance. The antennas were mounted at the height of 3.5 m above the ground. In the far field measurements, to reduce the specular ground reflections from the transmitting horn, the absorbers were used in the middle region of the measurement set up.



(a) (b)

Figure 39 Pictures of antenna measurements (a) mounted antenna (b) measurement setup

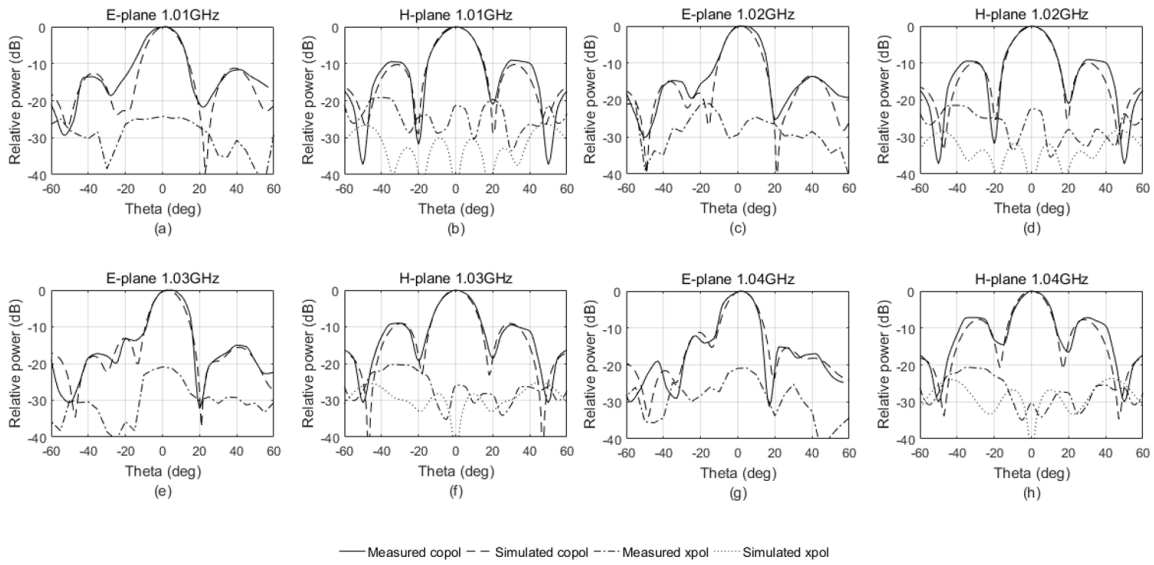


Figure 40 Simulated and measured radiation patterns in E and H-plane (a) & (b) 1.01GHz, (c) & (d) 1.02GHz, (e) & (f) 1.03GHz, (g) & (h) 1.04GHz (i) Comparison of simulated and measured realized gain as a function of frequency

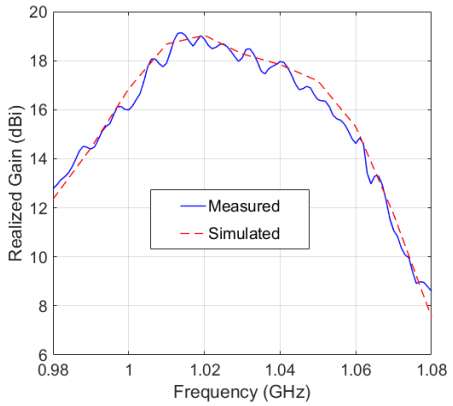


Figure 41 Comparison of simulated and measured realized gain as a function of frequency

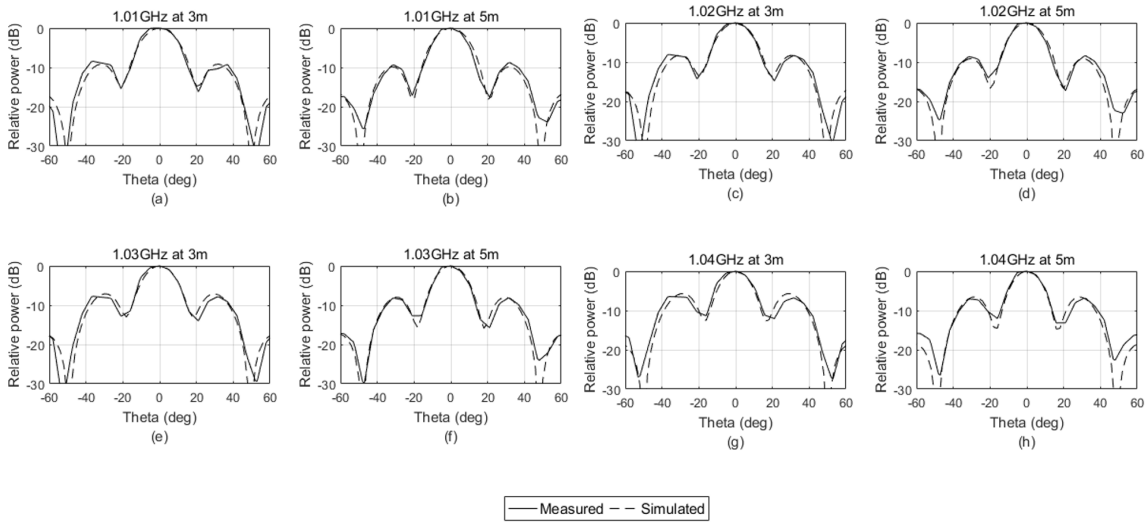


Figure 42 Near field relative power patterns at 3m and 5m distances from the antenna aperture, (a) & (b) 1.01GHz, (c) & (d) 1.02GHz, (e) & (f) 1.03GHz, (g) & (h) 1.04GHz

Figure 39 (a) - (h) shows the measured E & H-plane radiation patterns of the antenna, for the various frequencies in the band. The measured radiation patterns match well with the simulated ones. In the measured E-plane pattern at 1.03 GHz, the beam is shifted by 3° as compared to 2° in the simulations. The measured cross-polarization is less than -21 dB and -19 dB, for the entire band in the E and H-plane respectively. In Fig. 38, the simulated cross-polarization in the E-plane are less than -40 dB also shown in Figure 34. Figure 40 shows the simulated and measured realized gain as a function of frequency for the fabricated antenna. For planar directive antennas, the peak gain and gain pattern measurements are conveniently conducted in the antenna measurement ranges. However, since we do not have access to antenna measurement ranges, we have verified the peak gain of the antenna by using the conventional gain transfer method [30], using standard horn. The measured gain matches well with the simulated one. Peak measured value is 19.2 dBi

as compared to 19.1dBi in the simulations. The measured value is higher due to the ripples in the gain measurements, which are ~ 1.2 dB and are caused by the standing wave patterns in front of the aperture due to reflections.

Figure 41 shows the near field patterns of the antenna at 3 and 5m distances from the antenna aperture. Both the distances are in the radiating near field region of the antenna. The measured power pattern matches well with the simulated patterns. The measured maximum sidelobe level at 3 and 5m are -6.1dB and -6.6 dB respectively. It is observed that between 1.01 to 1.04 GHz, the maximum sidelobe level changes by ~ 3 dB, for both 3 m and 5 m distances. The antenna is used to receive the fields from the board processor at those distances as presented in the next section.

4.1.2 SNR Measurements and Malware Detection

The proposed antenna was used to measure the radiated emissions from the various embedded systems and Internet-of-Things (IoT) boards, at various distances under two conditions: direct Line of Sight (LoS) and Non-Line of Sight (NLoS). These IoT boards typically consist of an ARM processor, a Flash memory, and a set of peripherals (e.g., WiFi modules, etc.). IoT boards are typically used for controlling a variety of tasks in factory lines, hospitals, critical infrastructures, etc. Recently, there have been a growing interest in attacking these devices since both the number and importance of them are growing rapidly. Monitoring these devices using the EM side channel signals generated by them is one of the ways to improve the security of IoTs

against cyber-attacks. Collecting stronger EM signals will improve the accuracy of the malware detector and that is the main goal of designing our proposed antenna.

4.1.3 Line of Sight (LoS) Measurements

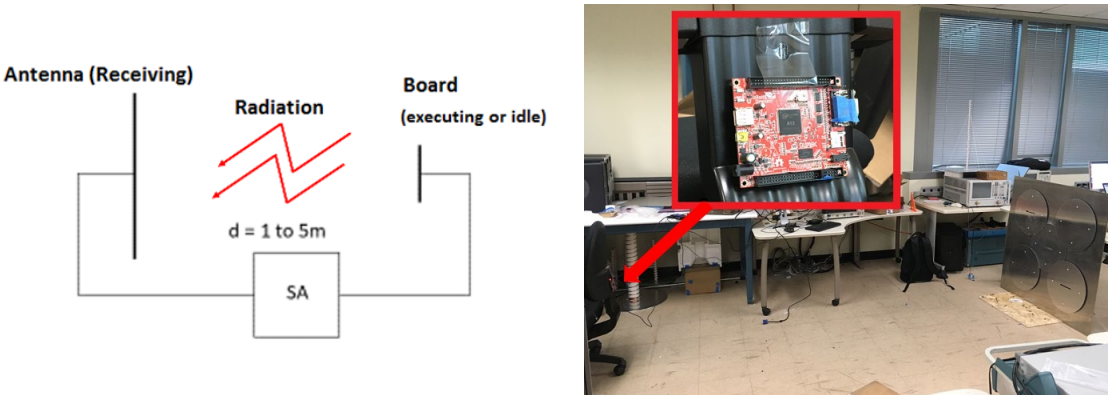
Here, we will first describe the direct LoS measurements for the IoT board shown in Figure 42 in detail. Figure 42 (a) shows a diagram of the measurement setup and Figure 42 (b) shows the photo of the measurement setup where the proposed antenna is measuring the EM signal from an IoT device named Olimex [26] which has an ARM processor and runs a Linux operating system. The signal power measurements, using a spectrum analyzer (Agilent N9020A), were conducted at various distances between 1-5 m from the device. For each distance, two measurements were collected and the corresponding Signal to Noise Ratio (SNR) was calculated. Since it is not straightforward to estimate SNR for emanations from the electronic devices, we have conducted additional experiments to estimate SNR as described below.

In the first set of measurements the objective is to estimate total emanated power, S, received when the board is on and running the program activity of interest. In the second set of measurements, the objective is to estimate the noise spectral power, N, received when the board is on but there is no application running (idle mode). The noise power here includes thermal ($N_{thermal}$) noise as well as emanations coming from the board itself (N_{board}) that are not related to the program activity. SNR is then calculated as:

$$SNR\ (dB)\ =\ S\ (dB)\ -\ N\ (dB) \tag{18}$$

$$\text{SNR} = \frac{P_{r_{executing}}}{P_{r_{idle}} + N_{thermal}} \tag{19}$$

where, $P_{r_{executing}} \propto \frac{P_t}{r^2}$ is the power received when the processor is executing the code, while P_t is the power at the input. $P_{r_{idle}}$ is the power received when the processor is turned on but not executing a code. This part carries no useful information, and acts as a source of noise. $N_{thermal}$ is the thermal noise, independent of distance.



(a) (b)

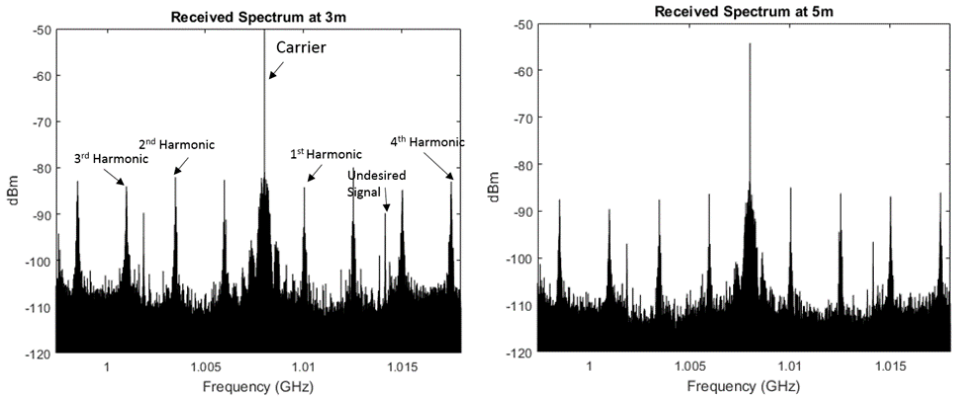
Figure 43 SNR Measurements for an IoT (Olimex) board: (a) Block diagram of set up (b) Set up picture that shows the antenna (on the right side) and the board (on the left side).

The proposed antenna is used to receive electromagnetic radiation coming from the board's processor. The objective is to find the possible malicious activities by analyzing the program execution through EM emanations. The main idea behind the malware detection method is that since there is a correlation between the program activities and the generated EM signals, executing

a certain application will generate unique and distinguishable signatures in the EM signal. Thus, by collecting these EM signals for each application and extracting the signatures, a reference model for each application can be built. Then during monitoring, if an attacker changes the application's code, this will result in generating different EM signals that no longer match with the model and hence can be detected. Further details can be found in [10], [11].

The signature extraction is based on the premise that a program spends most of its time executing some repetitive code (e.g., loops) which results in prominent peaks appearing in the spectrum separated by $\Delta f = {}^{1}/_{T}$, where T is the duration of a single loop iteration. In addition of base-band signal where these loops can be observed, they can also be observed as a modulated signal around the processor clock frequency (in our case 1 GHz), which is the signal we are observing. Measured power spectrum at the distances of 3 m and 5 m are shown in Figure 43 (a) and (b) respectively. From Figure 43 (a), we can observe that the strong spectral lines are amplitude modulated by a clock frequency (which acts as a carrier) of 1.008 GHz, which is significantly stronger than everything else. Each of the labeled harmonics are approximately 1.95 MHz apart from one another, which indicates that each iteration of the loop in the code takes about 514 ns. Since the board has many activities going on at once, it creates some other signals that are not related to the code that is being run on the processor. An example of this is marked as undesired signal in Figure 43 (a).

Figure 44 (a) shows the measured SNR for various distances in comparison with the SNR obtained by a theoretical model defined in (19). The theoretical fit agrees well with the measured SNR.



(a) (b)

Figure 44 Measured signal power while code is executing at various distances (a) 3 m and (b) 5 m

To explain the measured SNR, with the theoretical model, the noise observed in the measurements is assumed to be created by two sources: thermal noise and the noise generated by the board itself. Since the processor is not intended to function as a transmitter, only a part of the total radiation coming out of the board carries meaningful information. This undesired part of the radiation lowers the quality of the signal. Since this part of the signal is radiated from the board, it gets weaker by a factor of r^2 , whereas the thermal noise is constant, as pointed out in (19). For

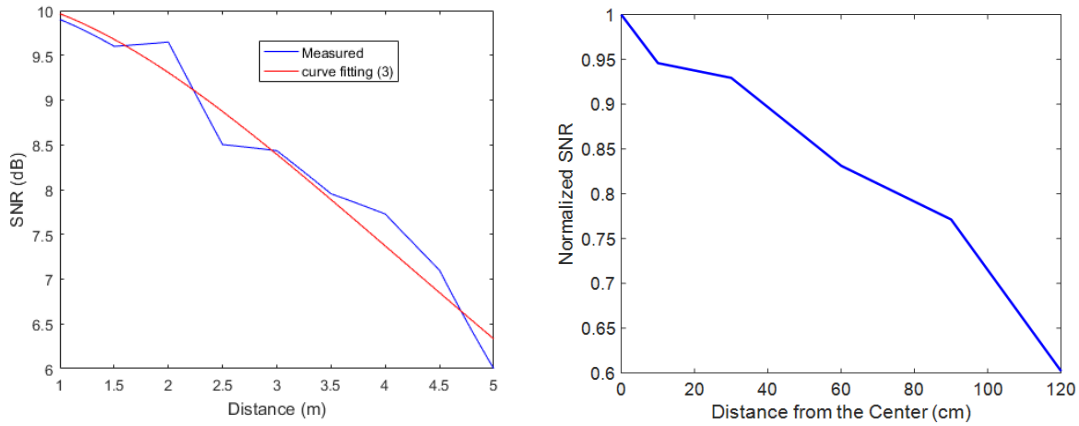
this reason; at smaller distances $P_{r_{idle}}$ is more significant, at larger distances $N_{thermal}$ is more significant, and at intermediate distances the SNR trend is neither constant nor r^2.

SNR fit is given as:

$$SNR_{fit} = \frac{\frac{a}{r^2}}{\frac{b}{r^2}+c} \tag{20}$$

$$SNR_{fit}^{-1} = \frac{b}{a} + \frac{c}{a}r^2 \tag{21}$$

It can be seen that SNR_{fit}^{-1} is a linear function of r^2 and the data points were fitted using linear least squares method. The multiplicative inverse was taken of the resulting line, which fitted the data very well.



(a) (b)

Figure 45 (a) Measured SNR vs. distance in comparison with the theoretical model fit (b) Measured normalized SNR vs offset distance from the LoS (SNR = 1 corresponds to LoS)

4.1.4 Non-LoS Measurements

As mentioned earlier, the proposed antenna is designed so that it can be hanged on the wall and received the EM signals from electronic devices that are active in a room. In this scenario, not all the monitored devices would be in the LoS but the antenna should still be able to monitor them. In order to use the EM signals for malware detection, the spectral peaks (as shown in Figure 43) should at least be 1 dB higher than the noise floor. In other words, in order to be able to monitor non-LoS devices, the receiving signal should have peaks with at least 1dB higher than the noise floor.

To evaluate the effectiveness of our design, we repeat the measurement in Figure 42 this time with moving the board toward up-down and/or left-right directions from the center of the antenna with the step of 10 cm. All measurements are done while the center of the board is 3 m away from the center of the antenna. For each step (i.e., different distances from the center of antenna while being 3m away from it), we measure the SNR for the receiving EM signal. Figure 43 (b) shows the results for the weakest peak in the test application.

As shown in the Figure 43 (b), the antenna can receive EM signals with only 30% decrease in SNR while being 1 m away from the center of the antenna. However, our measurements show that beyond 1m, the SNR decreases dramatically. This is due to directive beam in both E and H-plane.

4.1.5 Malware Detection

Finally, to illustrate how well the proposed antenna works in the system for malware detection, we use the antenna to receive EM signals while we are running several standard embedded systems applications such SHA, Djikstra's path-finding algorithm, QSort, CRC32, and FFT from a standard benchmark called *MiBench.* We also implement two real attacks: one a Distributed Denial-of-Service (DDoS) attack, and the other a Ransomware attack. We run the applications first 25 times without having any attack on them (benign), and 25 times with the DDoS attack, and 25 times with the Ransomware attack. We then used an algorithm proposed in [10] to analyze the receiving EM signals and label each run as either "benign" or "malicious". We

then calculate False Positive Rate as the number of runs that were incorrectly labeled as "malicious" divided by the total number of runs. Similarly, True Positive is defined as the number of runs that are correctly labeled as "malicious".

Our results show that for all the applications while measuring from 3 m, 4 m, and 5 m distances, we can perfectly find all the instances of the malware while achieving 0% false positive rate which confirms that our designed antenna is suitable for receiving EM signals from such devices from >3 m distance. Note that the results in [10] were reported while measuring from 5 cm distance from the board and collected by a probe.

4.2 Task 1.2 – Results for Automated Discovery of Sub-Channels

4.2.1 Experimental Setup

We have evaluated the algorithm by testing it on spectra from a desktop, a laptop, and a smartphone system described in Table II. The signals are recorded using the spectrum analyzer (Agilent MXA N9020A). The desktop and laptop measurements are collected with a magnetic loop antenna (AOR LA400) at a distance of 30 cm as shown on the left of Figure 44. To receive weaker signals from smartphones, EM emanations were recorded using a small loop probe with 20 turns and a 4 mm radius positioned on top of the cellphone as shown on the right of Figure 44. The spectra were measured from 0 to 4 MHz with a resolution bandwidth of 10 Hz.

Table III Description of measured devices

Type	Device	Processor
Laptop	Lenovo	Intel Core 2 Duo
Phone	LG P705	Snapdragon S1
Desktop	Dell	Intel i7

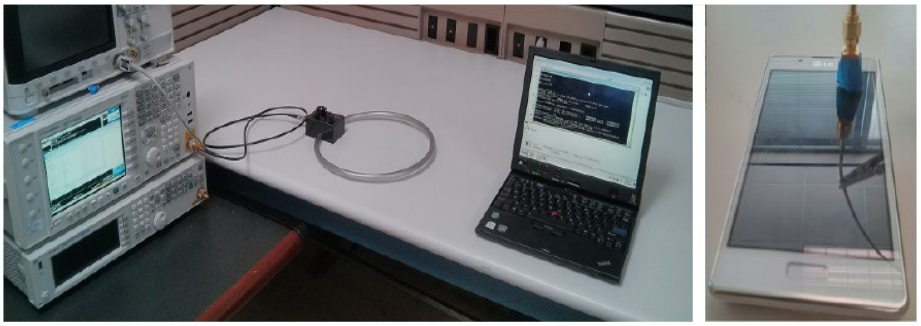


Figure 46 Measurement setup for laptop or desktop (left) and measurement setup for cell-phone (right)

The benchmarks are run at several different alternation frequencies $f_{alt} = \{23000, 29000, 37000, 53000\}$ Hz with duty cycles $d = \{20, 40, 50, 60, 80\}\%$. The alternation frequencies were chosen to ensure sufficient separation between sidebands of modulated signals, i.e. separation between f_{alt1}, f_{alt2}, etc. and their harmonics has to be sufficient to prevent overlapping. For example, if f_{alt1} = 23 kHz is chosen, frequencies in the vicinity of the harmonics of f_{alt1} should be avoided. Aside from this consideration, the choice of f_{alt} is arbitrary.

We have found that four alternation frequencies are sufficient in the algorithm to identify carrier frequencies. To identify if the modulation is AM or FM, we need all five duty cycles.

The benchmarks were run on the laptop and desktop systems single-threaded Windows 7 32-bit user mode console applications, and as normal Android applications on the smartphone. When possible all unrelated programs and activities were disabled, CPU frequency scaling was disabled, and screens were turned off. We measured two alternation activities. The first activity alternated between a load from DRAM memory and a load from the on-chip L1 cache, which we abbreviate as LDM/LDL1. This alternation is useful in exposing modulated carriers related to memory activity. The second activity alternated between loads from the on-chip L2 and L1 caches, which we abbreviate as LDL2/LDL1. This activity exposes carriers modulated by on-chip activity. We tried other instruction pairs (e.g., arithmetic, memory stores, etc.) and found that that all known modulated carriers could be found using just these two activities.

4.2.2 Experimental Results

We tested three devices described in Table I, with two measurements per device (one for LDM/LDL1 and one for LDL2/LDL1). Table II summarizes carrier frequencies found using our algorithm, type of modulation, signal to noise ratio (SNR) of the received carrier, and the confidence level that the found carrier is correctly identified from a laptop. Here, we define SNR as a difference in decibels between $M_{true}(f; d_j)$ and $M_{false}(f; d_j)$, as defined in equation (6). Our algorithm has found one FM carrier and it's two harmonics with confidence level above 99%. We

can also observe that SNR for all three FM modulated frequencies is above 10 dB which indicates that these carriers are strong and will carry signal to some distance away from the laptop. Our algorithm has also found one AM modulated carrier but the observed SNR is only 4 dB, which indicates that this is a weak carrier. Please note that our algorithm finds all carriers independently and then we check for possible harmonic relationship among found frequencies and if found, we report the harmonic order.

Table IV Carrier frequencies found in laptop

Carrier Frequency [Hz]	Harmonic No.	SNR [dB]	Type of Modulation	Confidence Level
383010	1	16	FM (Δf=2275 Hz)	99.8%
765949	2	12	FM (Δf=4700 Hz)	99.9%
1148959	3	10	FM (Δf=7225 Hz)	99.8%
448071	1	4	AM	99.1%

Table III summarizes carrier frequencies found using our algorithm, type of modulation, signal to noise ratio (SNR) of the received carrier, and the confidence level that the found carrier is correctly identified from a cell phone. Here, our algorithm has found one AM carrier and its second harmonic with confidence level above 99%. The SNR for these two frequencies is above 20 dB, i.e., they are excellent candidates to carry signal outside of the cellphone. Our algorithm has also found two FM modulated carriers, but the observed SNR is only 1 dB, which indicates that these are weak carriers.

Table V Carrier frequencies found in a cell phone

Carrier Frequency [Hz]	Harmonic No.	SNR [dB]	Type of Modulation	Confidence Level
110543	1	1	FM (Δf=3100 Hz)	98.6%
1599990	1	30	AM	100%
3200000	2	22	AM	99.98%
3257391	1	1	FM (Δf=96002 Hz)	99.98%

Finally, Table IV summarizes carrier frequencies found using our algorithm, type of modulation, signal to noise ratio (SNR) of the received carrier, and the confidence level that the found carrier is correctly identified from a desktop. Here, our algorithm has found one AM carrier and its 11 harmonics with confidence level above 99%. The SNR for first seven harmonics is above 10 dB, while SNR for other five harmonics is above 5 dB. Furthermore, we have found one more AM carrier and its seven harmonics all with SNR above 10 dB. Finally, we have found one FM carrier with SNR of 5 dB. To verify the accuracy of the algorithm, we have visually inspected all spectra and confirmed that carriers found by the algorithm exist in the spectrum. From the results, it can be observed that there are only 2 or 3 fundamental frequencies and the rest are their harmonics. The fundamental frequencies that were reported are all attributable to voltage regulator and memory refresh activity on the measured system. For example, in Figure IV we can observe that the two strongest sources are voltage regulator (315 kHz) and memory refresh (software activity in the system at 511 kHz). The voltage regulator emanations can be reduced by better shielding of coils, and the memory refresh can be eliminated by creating different scheduling pattern for memory refresh. Alternatively, program code can be changed to avoid power-fluctuations and memory activity that depends on sensitive information. Please note that carrier frequencies can be found at higher frequencies as well (here we have tested only up to 4 MHz).

They are typically above 500 MHz and belong to processor or memory clock. While our algorithm can find these frequencies as well, information about processor and memory clocks is readily available. Finding carrier frequencies at lower frequency range is more challenging because there is much more noise-like activity in the spectrum and it is difficult to identify information caring signals.

Automatic identification of potential carriers in the system has several benefits. From the security prospective, it allows us to quickly identify frequencies of interest for observing RF emanations, it allows prediction of distances from which we can expect to receive good quality signal (based on observed SNR), and the type of demodulation needed to correctly receive signals. From the system designer prospective, finding carrier frequencies helps us identify leaky circuits. For example, the unintentional FM and AM carriers found for a desktop and laptop were caused by voltage regulators and memory refresh commands. For a cell phone, several carriers were found to be caused by voltage regulators. The remainder of the carriers found on the cell phone were traced to particular IC packages or modules and were likely caused by either voltage regulators or an unknown periodic memory activity. However, smartphones integrate many system components into System on Chip (SoC) modules and often use Package on Package (PoP) technology to integrate both the processor and memory into the same package and little information is publicly available describing these components. More information would be needed to definitively determine the circuits and mechanisms modulating these carriers.

Table VI Carrier frequencies found in a desktop

Carrier Frequency [Hz]	Harmonic No.	SNR [dB]	Type of Modulation	Confidence Level
315488	1	28	AM	99.8%
631006	2	28	AM	99.99%
946654	3	22	AM	99.7%
1262312	4	21	AM	99.8%
1566849	5	19	AM	99.9%
1893447	6	18	AM	99.8%
2209415	7	13	AM	99.9%
2840661	9	5	AM	99.8%
3156239	10	6	AM	99.8%
3471917	11	8	AM	99.9%
3787705	12	6	AM	99.8%
451581	1	5	FM (Δf=550 Hz)	99.92%
511653	1	17	AM	99.96%
1023306	2	13	AM	99.97%
1534938	3	24	AM	100%
2046601	4	25	AM	100%
2558214	5	23	AM	100%
3069877	6	20	AM	100%
3581530	7	11	AM	99.99%

4.3 Task 2.1 – Results for Spectral Profiling

In this section, we evaluate our framework using three set of experiments to show the effectiveness of REMOTE to detect different types of attack on variety of devices. In the first set of experiments, we use two real-world cyber-physical system (CPS). The first CPS we use is an embedded medical device called Syringe-Pump which is a representative of a medical cyber-physical system. The second system is a PID controller that is used for controlling the temperature of a soldering Iron. This type of system could also be used to control the temperature in other settings, such as a building or an industrial process, and thus is representative of a large class of industrial CPS/IoT systems.

For the second set of experiments, we use five applications from an embedded benchmark suite called MiBench [16] running on an IoT/embedded device, which are a representative of the computation that is needed in that market (e.g., automotive, industrial systems, etc.)

Finally, for the third part of our evaluations, we chose a robotic arm (LewanSoul LeArm 6DOF) [31], which is a representative of commonly-used CPS existing in the market.

4.3.1 Experimental Setup

The measurement setup is shown in Fig. 45. Depending on the distance, either a hand-made magnetic coil or a horn antenna is used to receive EM signals (no amplifier is used). For all measurements, we use a cheap (<$30) software defined radio (SDR) receiver (RTL-SDRv3) to record the signal. Using this radio, the entire cost for the near-field measurement setup (including the radio and a hand-made coil) is only around $35, and for the far-field measurement setup is around $100-200 (depending on the antenna). Further cost advantages can be gained if REMOTE is used in settings where multiple similar devices (with similar vulnerabilities) are used, so a single (or a few) devices can be monitored by REMOTE (especially in far-field scenario), with random changes to which specific devices are monitored at any given time. Fig. 46 shows the entire setup including the monitored device (Syringe-Pump in this figure) and REMOTE. Note that all of our measurements were collected in the presence of other sources of the electromagnetic interface (EMI) including an active LCD that was intentionally placed about 15 cm behind the board. A set

of TCL scripts are used to control the monitored system and the SDR (to record the signal). The entire REMOTE algorithm is implemented on a PC using Matlab2017-b.

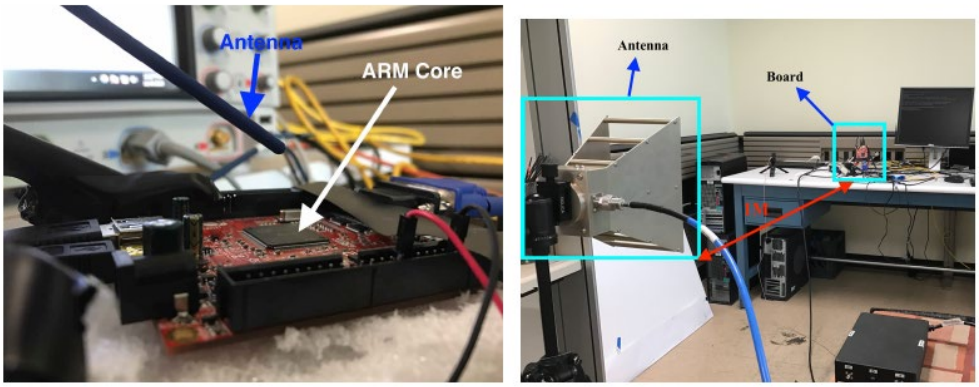


Figure 47 The near-field setup (left) consists of a small EM probe or a hand-made magnetic probe (not shown) placed 5 cm above the system's processor. A horn antenna placed 1 m away from the board for far-field measurements (right)

In all cases, a software-defined radio smaller and lighter than most portable USB hard drives, is used to record the signal.

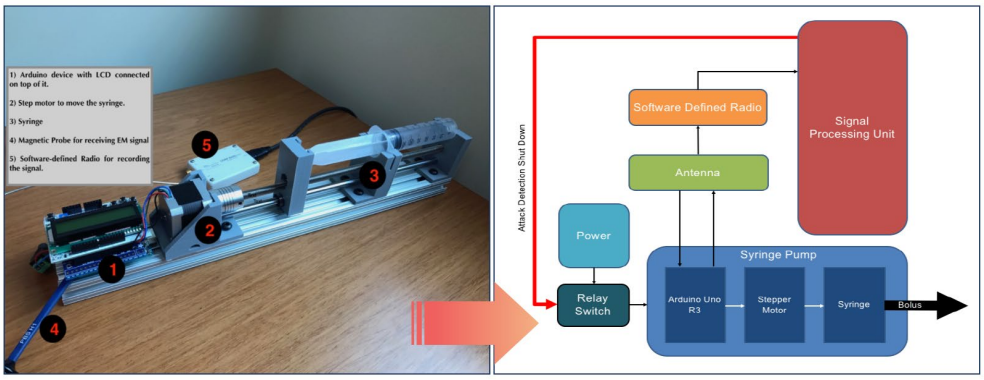


Figure 48 Syringe Pump (left) and REMOTE framework (right). In our setup, the signal processing unit is implemented on a separate PC.

4.3.2 File-less Attacks on Cyber-Physical-Systems

the monitored device (Syringe-Pump in this figure) and REMOTE. Note that all of our measurements were collected in the presence of other sources of the electromagnetic interface (EMI) including an active LCD that was intentionally placed about 15 cm behind the board. A set of TCL scripts are used to control the monitored system and the SDR (to record the signal). The entire REMOTE algorithm is implemented on a PC using Matlab2017-b.

The first part of our evaluations presents the results for two real-world CPS which are implemented on four different devices (shown in Table V). To attack these devices, we implement two end-to-end file-less attacks namely a code-reuse attack and an APT attack (advanced-persistent-threat). The first attack we implement in this paper is a Code Reuse [32], [33] attack on a medical CPS called Syringe-Pump. Syringe-Pump is a medical device designed to dispense or

withdraw a precise amount of fluid, e.g., in hospitals for applying medication at frequent interval [34]. The device typically consists of a syringe filled with medicine, an actuator (e.g., stepper motor), and a control unit that takes commands (e.g., amount of fluid to dispense/withdraw) and produces controls for the stepper motor. The systems must provide a high degree of reliability and assurance (typically by using a simple MAC) since imprecise or unwanted dispensing of medication, or failure to administer medication when needed can cause significant damage to the patient's health. In our evaluation, we use the Open Source Syringe-Pump from [35].

Our code-reuse attack involves overflowing the input buffer in reading the serial input function, which normally reads the input, sets a flag to indicate that new input is available, and returns. Exploiting this vulnerability, the return address in the stack is overwritten by a chain of gadget's addresses to launch an attack.

Since the security-critical part of this system is moving the syringe, a desirable goal for an attacker is being able to call the MoveSyringe() function, which is responsible for syringe movement, at an unwanted time while skipping the input checking part, Delay() function, which is responsible to check the authenticity of the command (otherwise the attacker needs to hack into the C&C server to send the commands which may not be a feasible task).

Table VII Boards used in this paper to evaluate REMOTE.

Device	Processor	Clock-rate	OS
Arduino Uno	ATMEGA-328p	16MHz	No
DE0-CV Altera FPGA	Nios-II softcore	50MHz	No
TS-7250	ARM9	200MHz	Debian
A13-OLinuXino	ARM Cortex A8	1GHz	Debian

We use ROPGadget [36] for finding the proper chain of gadgets to put the address of MoveSyringe() in a register and branching to that function (from the readInput() function to skip the checking part). After branching to MoveSyringe() and executing it, PC jumps back to the main function and resumes normal behaviour of the application.

Figure 48 shows a spectrogram of the Syringe-Pump application in (top) malware-free run, and (bottom) when the CR attack happens. As seen in the figure, the Syringe-Pump application has three distinct regions with clearly different EM signatures: printing debug info and reading inputs, a delay/checker function which checks the message authenticity (using a simple MAC), and an actual movement of the syringe. The major difference between these two figures is the reverse order of "Delay" and MoveSyringe() parts in malicious run (bottom). In normal behaviour, REMOTE expects to see readInput ! Delay! MoveSyirnge however, in CR attack, since the return address of the readInput function is overwritten by the adversary, the code immediately jumpsto MoveSyringe() and skips the "Delay" part, thus in the spectrogram, the third region

(MoveSyringe()) is seen before “Delay” (bottom), which violates the correct ordering of regions and will be reported as “malicious” by REMOTE.

Our evaluation uses one attack per run in 25 runs with REMOTE successfully detecting each of these attacks (see Table VI). We then performed 25 attack-free runs and found that REMOTE produced no false positives (see Table VI). To further evaluate our system, we performed 1000 malware-free runs and 1000 malicious run on one device (Arduino) for 24 hours. For these 2000 runs, REMOTE successfully found all the 1000 instances of malicious run and reported 997 out of 1000 malware-free runs as normal (i.e., only 3 out of 1000 false positive = 0.3%).

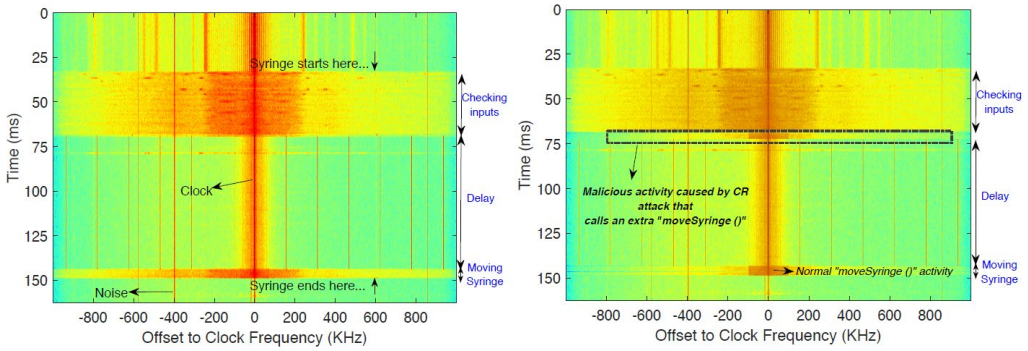


Figure 49 Spectrogram of the Syringe pump application in malware-free (left) and malware-afflicted (right) runs. Note that the differences in colors between the two spectrograms correspond to differences in signal magnitude which are caused by different

positioning of the antenna. Such variation is common in practice and has almost no effect on REMOTE's functionality because REMOTE was designed to be robust to such variation

Table VIII Accuracy of REMOTE for several different systems and attack scenarios using various boards and applications

Device (attack)	Syringe-pump (code-reuse attack)							
Board	*Arduino*		*Nios-II*		*TS-board*		*OLinuXino*	
Accuracy	True Pos.	False Pos.	True Pos.	False Pos.	True Pos.	False Pos.	True Pos.	False Pos.
	>99.9%	<0.3%	>99.9%	<0.1%	>99.9%	<0.1%	>99.9%	<0.1%

Device (attack)	Soldering-iron (APT attack)							
Board	*Arduino*		*Nios-II*		*TS-board*		*OLinuXino*	
Accuracy	True Pos.	False Pos.	True Pos.	False Pos.	True Pos.	False Pos.	True Pos.	False Pos.
	>99.9%	<0.1%	>99.9%	<0.1%	>99.9%	<0.1%	>99.9%	<0.1%

Device (attack)	Embedded/IoT (shellcode attack)									
App	*bitcount*		*basicmath*		*qsort*		*susan*		*fft*	
Accuracy	True Pos.	False Pos.	True Pos.	False Pos.	True Pos.	False Pos.	True Pos.	False Pos.	True Pos.	False Pos.
	>99.9%	<0.1%	>99.9%	<0.1%	>99.9%	<0.1%	100%	<0.1%	>99.9%	<0.1%

Device (attack)	Robotic-arm (firmware modification)	
Board	*LewanSoul LeArm 6DOF*	
Accuracy	True Pos.	False Pos.
	>98.2%	<0.2%

Note that, depending on the size of injection, the MoveSyringe() in Syringe-Pump could be very brief in time (e.g., around 3 ms as can be seen in Fig. 4-left), and we found that without correctly handling the interrupts on Olimex and TS platforms (which have an operating system), we would either get very high false positives (due to interrupts), or high false negatives (by using large N to ignore short-term activity). However, by adding training-time samples for interrupts, we can use small N, while having 0% false positives.

Furthermore, we also repeated our measurement for Syringe-Pump for both 50 cm and 1 m distances (using a 9 dBi horn antenna [37] connected to the SDR) and in both cases, we also get

perfect accuracy. It is also important to mention that the detection latency (i.e., the time attack starts until REMOTE detects it), for all four devices is <2 ms.

An alternative method for attacking Syringe-Pump is by changing the InjectionSize (i.e., Data-only attacks). This also can be done using a CR attack. REMOTE is able to protect Syringe-pump against such attack since changing the InjectionSize will change the duration (i.e., the number of SSs) of MoveSyringe(). Since REMOTE is checking the signal in the granularity of SS, it can count the SSs which belong to MoveSyringe() activity and compare it to the expected number of SSs. To check how well REMOTE can detect such an attack, we check the number of SSs for MoveSyringe() for all the 25 attack-free runs and compare it to the actual InjectionSize. In all the instances, REMOTE reports the correct number of SSs. Note that we are not detecting EM emanations (RF) signal produced by the motor movement but the change in the code execution when "data-only" attack is performed. i.e., we observe the signal at clock frequency of the board and observe software changes, while motor movement signature occurs at much lower frequencies.

However, if the change is less than one SS or if the expected InjectionSize is unknown, REMOTE is not able to detect the change. Overall, there is a trade-off between the size of SS and REMOTE's ability to detect small changes.

Thus to improve the effectiveness of the system, either a higher sampling-rate setup can be used (smaller SS hence smaller detection granularity) or REMOTE can be combined with other existing methods (e.g., Data Confidentiality and Integrity (DCI) methods [38]) to protect the

system against different types of data-only attacks. Finally, it is important to mention that However, as shown in this work (for this attack and other attacks in this section), meaningful attacks typically have much larger signature (i.e., order of milliseconds) than the current detection limit in REMOTE (200 microseconds).

The second attack is an advanced-persistent-threat (APT) attack on an industrial CPS (called Soldering-iron). A well-known example of such attack for CPS is Stuxnet. Soldering-iron is an industrial CPS that allows users to specify a desired temperature for the iron and maintains it at that temperature using a proportional-integral-derivative (PID) controller. This type of controller could also be used to control the temperature in other settings, such as a building or an industrial process, and thus is representative of a large class of industrial CPS. This application is significantly larger than the Syringe-Pump - with 70,000 instructions in its code and 1,020 static control-flow edges [35].

The application starts by initializing all the components (e.g., PID controller, Iron, etc.). It then begins to control the Iron's temperature: it checks all the inputs (e.g., knob, push buttons, etc.) and then based on them decides to decrease or increase the temperature, prints new debug information on its display, etc. and then repeats this ad infinitum. The security-critical function is where the temperature of the iron is set keepTemp(). This function uses an iterative process (a PID controller) to change or keep the temperature of the iron. The critical variable is temp hist – it

holds the last two temperatures of the iron and is used to calculate the difference between the current temperature of the iron and these two last temperatures.

To implement a Stuxnet-like malware on this application, we assume that the attacker can reprogram the device. The attacker's goal is to change a critical value under some conditions, which in turn can cause damage to the overall system. A possible modification to the code is shown in Example 1 (lines 8-10), where based on one or several conditions (e.g., in our evaluation it checks the model of the device that is stored in memory), the temperature history can be changed. The key insight is that the added instructions will cause the spectral spikes during execution of the main loop to be shifted to lower frequencies (more time per iteration) as shown in Fig. 48 for the A13-OLinuXino device. To evaluate how well REMOTE can detect this type of attack, we use 7 runs in training, and use 25 runs without malware and 25 runs with malware to evaluate the monitoring algorithm. Our results show REMOTE can successfully detect all the instances of the attack (a 100% true positive rate) (see Table VI).

```
// The main loop
void loop() {
    int16_t old_pos = read(&rotEncoder); // finding
    the position of the control knob

    bool iron_on = isOn(&iron); // iron is the
    object for the soldering iron

    // adding malicious activity
  if (some_condition){
    iron.temp_hist[0] = maliciousVal0;
    iron.temp_hist[1] = maliciousVal1;
    // where these values can be read from a file,
    memory or they could be random
  } // end of malicious activity

   byte bStatus = intButtonStatus(&rotButton); //
    reading input button

   showScreen(pCurrentScreen);
   keepTemp(&iron);    }
```

Example 1. A code fragment from the main loop of the soldering iron application and a possible injected malicious activity.

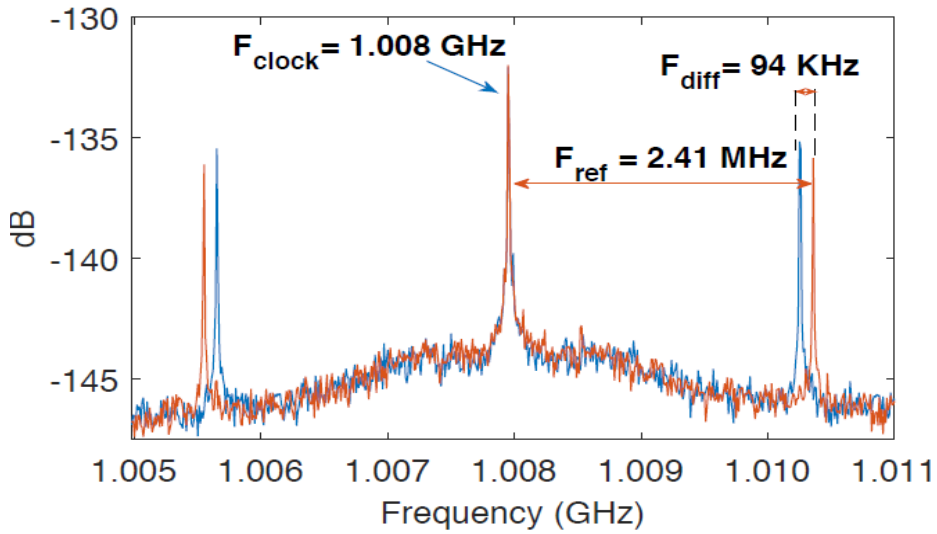


Figure 50 Adding malicious activity to the main loop of the Soldering-iron application (red: without malware, blue: with malware)

4.3.3 Shellcode Attack on IoTs

Another popular class of attacks on CPS/IoTs are shellcode attacks where the adversary executes a malicious application (payload) through exploiting a software vulnerability. It is called "shellcode" because it typically starts a command shell (e.g., by executing (/bin/sh) binary) from which the attacker can control the compromised machine, but any piece of code that performs a similar task can be called shellcode.

Once the attacker takes the control, she can execute any injected code such as a Denial-of-Service attack. In this section, we implement this attack by invoking a shell (/bin/sh) via a buffer overflow exploit. We then run two malicious payloads on the invoked shell: a DDoS bot, and a Ransomwmare. These attacks typically target devices with operating systems. In this work, we implement them on an IoT device with an ARM core (A13-OLinuXino), which is a representative of state-of-the-art IoTs.

The attacks are implemented on five representative programs from MiBench suite (bitcount, basicmath, qsort, susan, and fft). We chose these applications among all the MiBench applications (this benchmark is designed to represent typical behaviours of embedded system: e.g., Security, Telecomm., Network, etc.) mainly because bitcount is a good representative of the applications that have several different distinct regions (our HDBSCAN clustering found 9 for this application) and has lots of different activities including nested-loops, recursive functions, interacting with memory, etc. basicmath is chosen because it is a good representative of

unstable/weak activities since the activities in each region are very dependent on values (it is calculating different fundamental mathematics operations such as integration, square-root, etc.). We also chose qsort because it has lots of memory accesses, and picked susan and fft since they are good representatives of common and popular activities in embedded system domain (i.e., image processing and telecomm.). In all these application, first a buffer-overflow vulnerability is exploited, and using a shellcode, a shell with same privileges as the original application is invoked.

A malicious payload (i.e., DDoS or Ransomware) is then executed in this shell. For the DDoS, we port the C&C and the bots from the Mirai open source to run on our IoT. The DDoS payload execution begins right after the shell is invoked and ends after sending 100 SYN packets. The application then resumes its normal activity. We use a PC on the local network as the target of the DDoS attack (SYN flood), and we verify on that PC that the attack is taking place. As another payload, we also implement a simple Ransomware prototype payload that uses AES-128 with CBC mode to encrypt data. This encryption represents the bulk of the execution activity created by Ransomware.

As in previous cases, we use 7 runs for training and then use 25 runs without malware and 25 runs with each malware (i.e., DDos and Ransomware) for all five applications.

Our results (see Table VI) show REMOTE can successfully detect all the instances of the attack (a >99.9% true positive rate) while none of the malware-free runs incorrectly identified as malware (0% false positive rate). We found that invoking a shell itself is visually detectable on

our IoT device since it takes around 8 ms (about 32 SSs) and sending 100 SYN packets adds about 4 ms to that (see Fig. 49 (left) for DDoS and (right) for Ransomware).

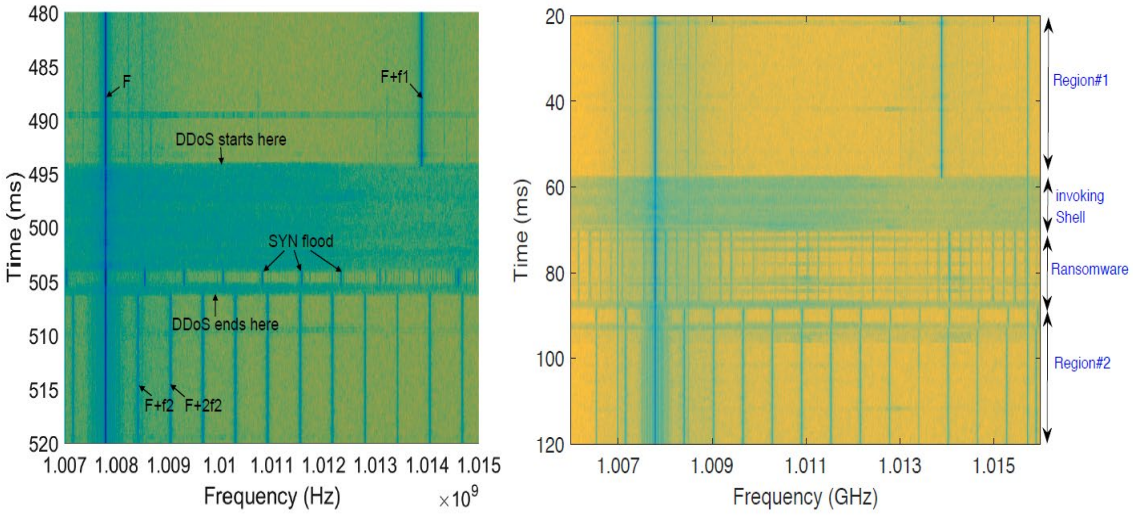


Figure 51 A run (left) where exploit, shellcode, and a 100-packet payload are injected into the execution between the original loops. A run (right) where exploit, shellcode, and a Ransomware payload are injected into the execution between the original loops.

4.3.4 APT Attack on Commercial CPS

The final system in our evaluation is a Robotic-arm. Robotic arm is often used for manufacturing and, typically, a critical component of any modern factory. It usually receives inputs/commands for a user and/or sensors and move objects based on these inputs. There is a growing concern in security of these CPSs since they are typically connected to the network and

are exposed to cyber-threats [39]. In this work, we use a commercial robotic arm (LewanSoul LeArm 6DOF [40]) which uses an Arduino board as a controller and a Bluetooth module to receive command. For this system, we implement an APT attack (firmware modification), where we assume that the reference libraries (e.g., library for Servo, Serial, etc.) are compromised (this can be also considered as a zero-day vulnerability). Note that, we assume that REMOTE s training contains the "unmodified" version of these library (baseline reference data). In this attack, we modify a subroutine (writeMicroseconds()) in Arduinos Servo library [41] by adding an extra if/else condition to change the speed of Servo motor randomly and reprogram the system with this compromised library, assuming that the adversary is interested in causing a malfunction in arms movement in real-time occasionally.

We use 7 runs for training and then use 1000 runs without and 1000 runs with the firmware modification. Our results (see Table VI) show REMOTE can successfully detect the instances of the attack with very high accuracy (>98.2% true positive rate) while only less than 0.2% of the malware-free runs incorrectly identified as malware.

4.3.5 Further Evaluation of Robustness – Interrupts and System Activity

Among the platforms we tested, the longest-duration system activity "inserted" (via an interrupt) into the application activity tends to take a few milliseconds, and it appears to be associated with display management/update because disabling lightdm [42], the display manager, eliminates these interrupts (but other kinds of interrupts still occur).

In contrast, in bare-metal devices interrupts (when there are any) tend to be around a microsecond in duration. Figure 50 shows the (perfect) ROC curve (solid blue line) for SyringePump on Olimex (and Debian Linux OS) when using REMOTE. We then prevented REMOTE from forming interrupt-activity clusters during training, and used the EDDIE's scheme, and that has resulted in a severely degraded ROC curve (red dashed line) where many false positives are detected when 4 consecutive clusters are found to be "unknown" (N = 4), and where increasing N reduces the false positives but also the true positives. This confirms that our approach of addressing system activity directly in REMOTE is significantly contributing to REMOTE's ability to detect malware while not reporting false positives due to system activity.

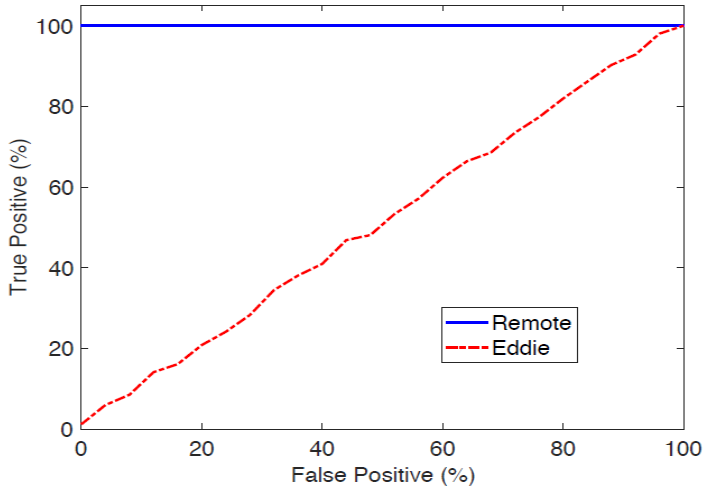


Figure 52 Accuracy of REMOTE with its mechanism for addressing interrupt activity (solid blue line) and EDDIE [10] (red dashed line). The results are for the SyringePump software running on the Olimex board

4.3.6 Further Evaluation of Robustness – Hardware Platforms and Distance

As mentioned in Section 2.3, packaging and other limitations may require the EM signal to be received from some distance, which significantly weakens the signal. To evaluate the impact of distance on REMOTE, we receive the signal from distances of 5 cm, 50 cm, and 1 m away from each of the tested devices. To limit the amount of data that is recorded, we use only two representative programs from MiBench suite (bitcount and basicmath), and only two representative malware behaviours - one that adds a relatively small number of instructions inside a loop (Stuxnet-like), and another where similar malicious activity is done all-at-once outside of loops (DDoS-like).

For each device and each application, we use 25 malware-free runs and 25 runs for each of the two malware activities (75x3 runs for each of the platforms) to obtain the false negative (malware activity not reported in a malware affected run) and false-positive rates (malware reported in a malware-free run) achieved by REMOTE. Our results show perfect accuracy (i.e., 0% false negatives and 0% false positives) for all devices and all three distances. However, if we prevent REMOTE from using total non-clock power when comparing SSs and use the scheme in EDDIE and/or Syndrome, on the TS board (which has the weakest signal among the boards tested) for 50 cm and 1 m distances we only observe 80% (at 50 cm) and 55% (at 1 m) true positive rates once we adjust other parameters to achieve 0% false positives (see Figure 51). This confirms that

when signals are weak, comparisons based on spectral peaks alone are insufficient and other signal features (such as non-clock power used in REMOTE) must also be considered.

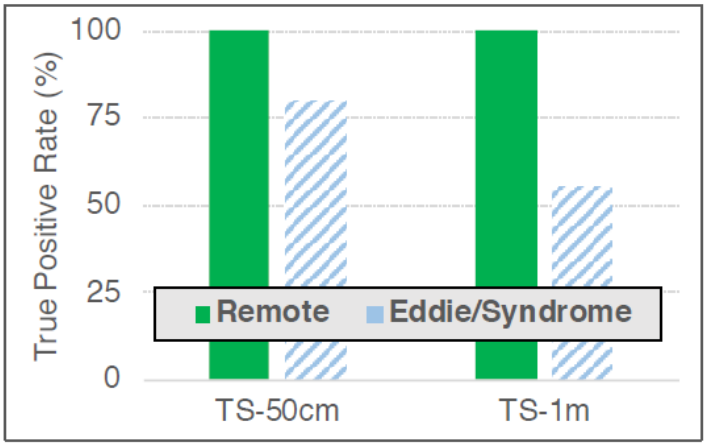


Figure 53 True positive rate (with 0% false positives) of REMOTE with its non-clock-power feature when comparing SSs (dark blue) and EDDIE /SYNDROME [10] (light red). The results are for basicmath running on the TS board

4.3.7 Further Evaluation of Robustness – Manufacturing Variations

To study the effect of manufacturing variations on the EM signals and REMOTE accuracy, i.e., to determine if training is needed for each type of device or for each physical instance of a device, we use 30 physical instances of the Cyclone V DE0-CV Terrasic FPGA development board (chosen primarily because we have 30 such boards), to train REMOTE on one board (randomly

selected) and use that training to monitor each of the other 30 instances, with 20 runs of bitcount on each instance, both with and without malware.

Our results show that REMOTE's accuracy remains at 100% true positives and 0% false positives throughout this experiment. However, when we prevent REMOTE from frequency-adjusting the SSs used in comparisons, we still find no degradation for 17 of the boards, but for 13 the false positive rate increases to nearly 100%. Further analysis shows that the clock frequencies of the boards vary, with 17 of them (including the one trained-on) were within the frequency-tolerance (parameter D) of the matching, whereas the other 13 were outside the tolerance, causing none of their peaks to vote for the cluster the signal actually should belong to. If D is then adjusted to avoid false positives, the true positive rate is severely degraded.

Figure 52 shows one such scenario where we trained on board number 3, and test on board number 4. The figure shows the ROC curve for board number 4 when frequency-adjusting is active and inactive. We also repeated this experiment for 10 Olimex boards (we do not have 30 of those), with very similar results with and without REMOTE's frequency adjustment.

These results confirm the need for frequency adjustment in REMOTE if training and monitoring do not use the same physical instance of a device.

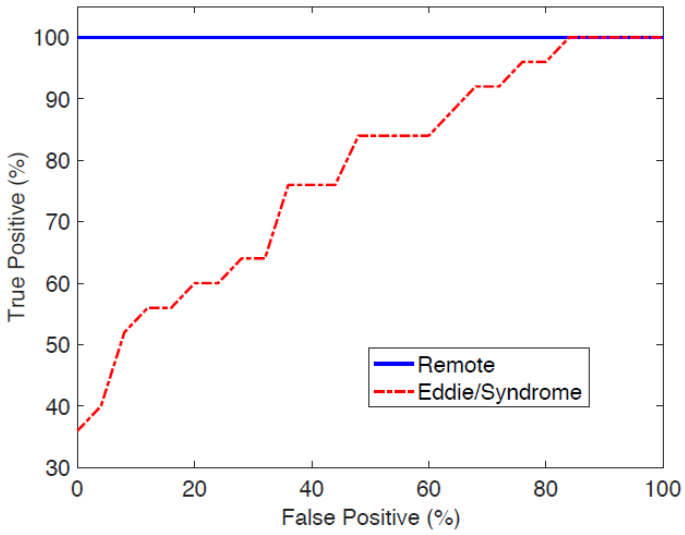


Figure 54 Accuracy for REMOTE with frequency-adjusting, vs. Eddie/Syndrome for FPGA board running bitcount

4.3.8 Further Evaluation of Robustness – Variations Over Time

We record the signals at one-hour intervals, over a period of 24 hours, while keeping the FPGA board and the receiver active throughout the experiment, to observe how the emanated signals vary over time as device temperature (and room temperature) and external radio interference such as WiFi and cellular signals change during the day and due to the day/night transition. The set of measurements collected each hour consists of 60 bitcount runs, 20 without malware and 20 times with each of the two types of malware. The training data for all REMOTE analyses in this experiment was recorded just after the device (FPGA board) and the receiver (SDR) were turned on.

We observed no deviation from REMOTE's accuracy throughout this experiment (solid blue line in Figure 53). We then prevent REMOTE from clock-adjusting the frequencies and repeat the experiments (on the same signal recordings), and find that the detection accuracy is dramatically degraded between hours 4 through 13 and hours 23 and 24 (dashed red in Figure 53). Further analysis shows that the clock frequency has shifted during these hours, coinciding with use of business-hours and off-hours thermostat setting for the room4, likely because temperature affected the board's crystal oscillator whose signal is the basis for generating the processor's clock frequency.

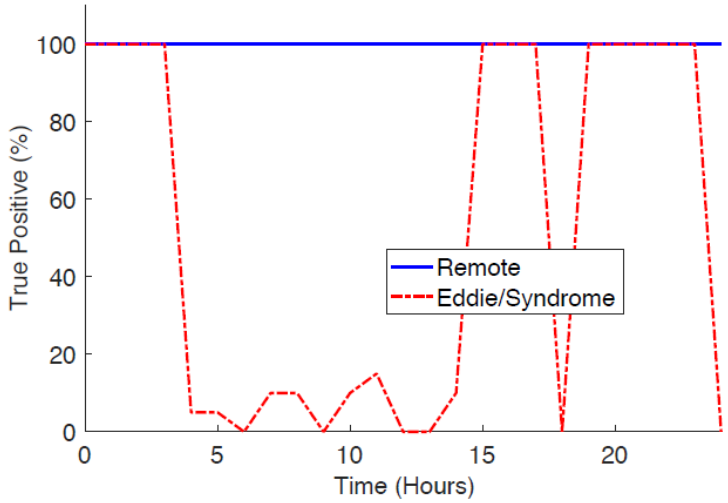


Figure 55 Performance of REMOTE with its clock-frequency adjustment feature vs. Eddie/Syndrome

4.3.9 Further Evaluation of Robustness – Multi-tasking/Time-sharing

In our final set of experiments, we apply REMOTE in the runs where Ransomware is executed as a separate process, without changing the application. The OLinuXino board only has one core, so its Debian Linux OS context switches between the two processes until the Ransomware payload completes. Figure 54 shows the spectrogram in one such execution. In the first part of the spectrogram only the application is running. At some point (millisecond 812 in this spectrogram), the Ransomware process is started, and the context-switching in (approximately) 10 ms time-slices can clearly be seen beyond this point in the spectrogram.

The spectrum of the malware process is clearly different from the spectrum produced by the application at this point in its execution, so we expect REMOTE to detect this malware execution scenario easily.

To evaluate REMOTE accuracy for this scenario, we use 25 runs, and in each run, start the Ransomware process at a different point in the run. The results of this experiment are that REMOTE successfully detects all these runs even with the tolerance threshold that produces no false positives for malware-free executions. It should be noted here that in this set of runs, according to our threat model, the IoT system is running only one valid application. To successfully handle scenarios in which the system context-switches between multiple valid applications, REMOTE must be extended to identify when context switches are occurring and to keep track and validate spectral samples with the knowledge of which application(s) they might

belong and where the "current" point is in each of those applications. Although we believe such an extension to REMOTE is possible, it will likely require significant effort to figure out, implement, and evaluate, so we leave it for future work.

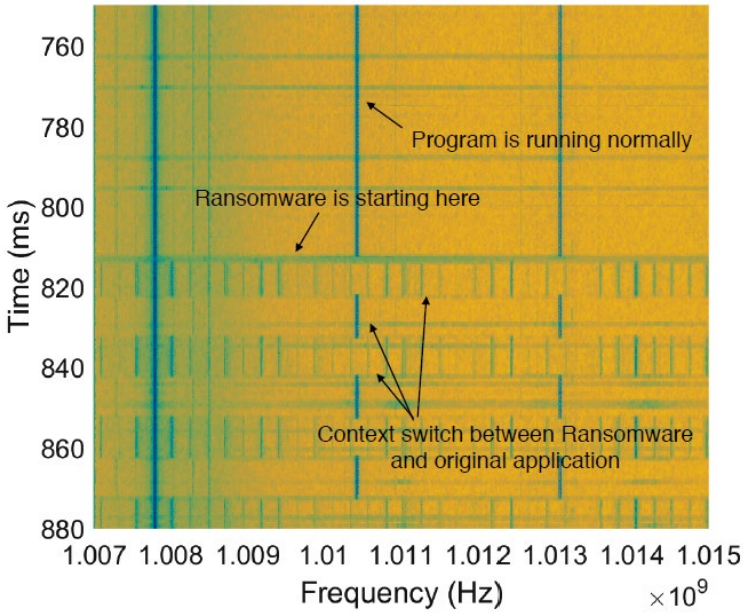


Figure 56 Spectrogram of context-switching between the unmodified Bitcount application and the Ransomware process

4.4 Task 2.2 – Results for Multi-Core Spectral Profiling

In this section, we provide experimental setups and results for the proposed profiling scheme. We perform experiments on 3 different devices with different number of cores. Table VII

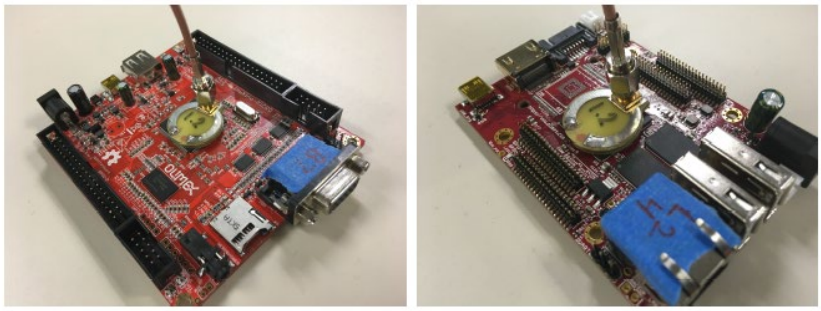
demonstrates the name of the devices and the number of cores they have. The goal of this section is to illustrate that the proposed methodology works for multiple devices, programs and cores.

4.4.1. Experimental Setup

The experimental setup for all devices are given in Figure 55. For the experiments, we used a lab-made magnetic probe and a spectrum analyzer [43] to measure the emanated signals. We first present our results when only a single core then multiple cores are active.

Table IX Devices and corresponding core frequencies and numbers

Device	# of Cores	Clock Frequency (GHz)
A13-OLinuXino-Micro [55]	1	1
A20-OLinuXino-LIME2 [56]	2	1
Alcatel Ideal [57]	4	1.1



(a) A13-OLinuXino-Micro. (b) A20-OLinuXino-LIME2.

(c) Alcatel Ideal.

Figure 57 Experimental setups

4.4.2 Program Profiling When Only One of the Cores is Active

In this section, we provide experimental results for the proposed model when a single core is active. We first collect training signals, and then train our neural network by utilizing Algorithm 1. This is followed by generating state transition diagrams by applying Algorithm 2. The profiling results while Bit_count is executed in a single core for Alcatel phone are given in Figure 56. In this figure, we consider two scenarios: when the system is benign and when the program has malware. The decision stream for the benign system is given in Figure 56a. Please observe that the plot also contains a state with the number "8" which does not comply with the Markov Model given in Figure 17. We use this extra state to represent that the output of the CNN is not above threshold and the Markov Model could not assign the current output to any possible states. Since there exists only a single path for each state (which can be executed only if the execution of the previous state is completed), we observe a stepwise structure which indicates the program is running properly. However, if we consider Figure 56b, we observe an anomaly between State -2 and State-3. Since this anomaly lasts longer than t_M second, we alert malware. Here, t_M is set to 2 ms which represents the sensitivity of the program.

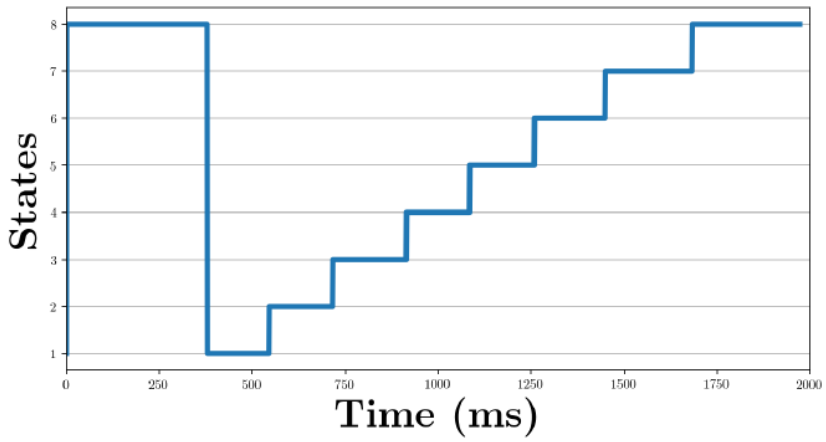
Please note that the proposed framework does not only work for state transitions which have only a single direction as given in Figure 17, but also for any Markov Model that has more branching operations. As an example, we consider a SAVAT program [9] which generates four different states during its execution. Compared to Bit_count and Basicmath, the Markov model of the program has more paths, as given in Figure 57. This program has infinitely many paths since

the state ‘4’ has a connection to the state ‘1’. The biggest advantage of the proposed methodology is that we do not need to collect all possible training signals, which is impossible for this case.

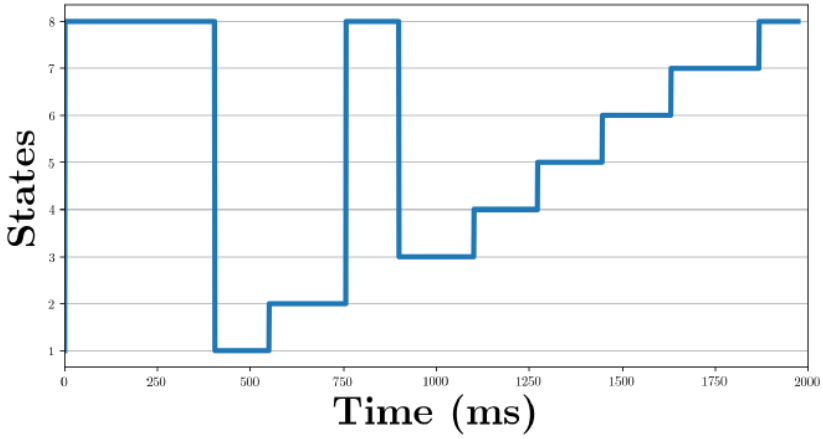
The training signals are collected only from a few executions of the code, irrespective of the path as long as the measurement has samples from each state. Since the states are chosen such that they produce similar signals whenever they are executed, having a couple of execution is enough to train our CNN model. After training the CNN model, the program can be monitored applying Algorithm 2. An example of the state transitions of the program is given in Figure 58. The actual state transitions for the experiment is “1-2-4-1-2-3-4”. We observe that the transitions are perfectly followed by the proposed method. Therefore, the proposed methodology can monitor systems even with a more complex Markov Model.

We have done experiments on all of the devices with these programs. We obtain 0% false negative rate (claiming no malware although the program has malware) for all of

the devices. However, the false positive rate is 0.1%, 0.2% and 0.3% for OLinuXino-A13, OLinuXino-A20 and Alcatel, respectively. These ratios are calculated by dividing inaccurate “idle” samples to total number of samples (> 103). Please note that having a false positive only increases the maintenance cost of the system. However, having a false negative can cause serious problems since this inaccurate classification can cause information leakages or irreparable damages on the system.



(a) The states while profiling the system when `Bit_count` is running.



(b) The states while profiling the system when `Bit_count` with a malware is running.

Figure 58 State transition diagrams while only a single core is active

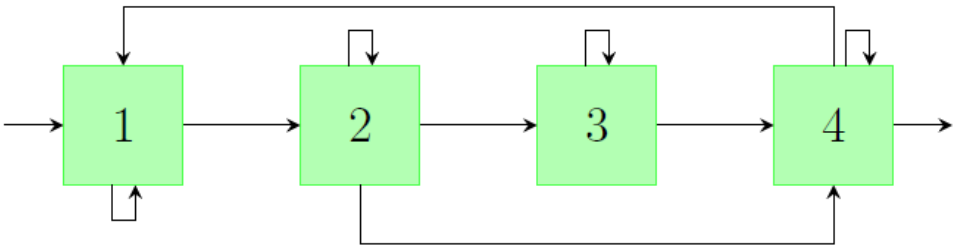


Figure 59 *Proof-of-concept implementation:* State transitions of a program written by combining SAVAT with different alternation frequencies

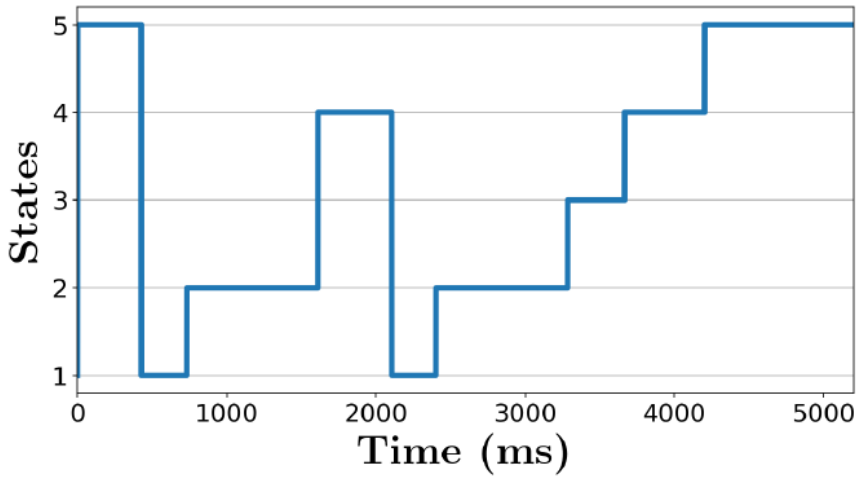


Figure 60 State transitions while profiling SAVAT based program

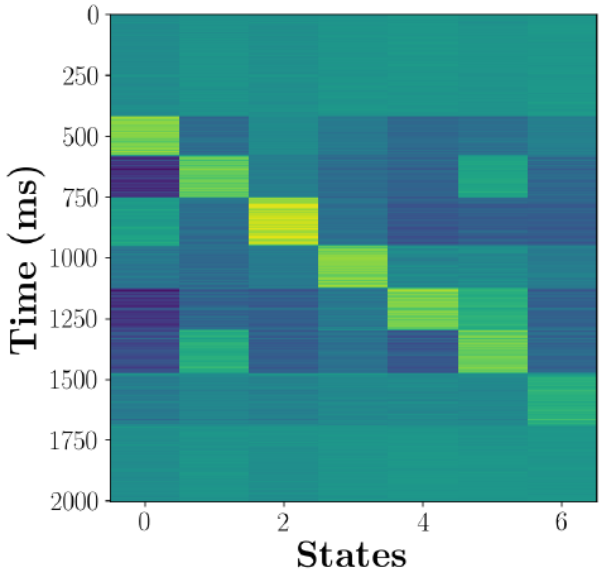
4.4.3 Program Profiling When Multiple Cores Are Active

In this section, we consider the scenario when multiple cores are active. For a better explanation, we first provide examples when two cores are running Basicmath and Bit_count, but provide results for other cases as well. We investigate 1) the results when both programs are benign and 2) why the cooperation between the neural network and the Markov Model is important. In that respect, we run Basicmath and Bit_count on the Alcatel phone such that these programs are executed at different cores. The spectrogram of the received signal is already given in Figure 16. We provide the values of the first parallel unit of the proposed neural network and corresponding state-transition flow obtained by employing Algorithm 2. The measurement started before program execution, and monitored until all program executions were done. In these "idle" regions, the proposed model does not report any malware because it is aware that the cores are not executing the programs. In other words, these "idle" states appear at the beginning and end of the program no matter what, and in the middle of the program if there is malware. Considering Figure 59a, we observe two main problems that can cause inaccurate monitoring of a program:

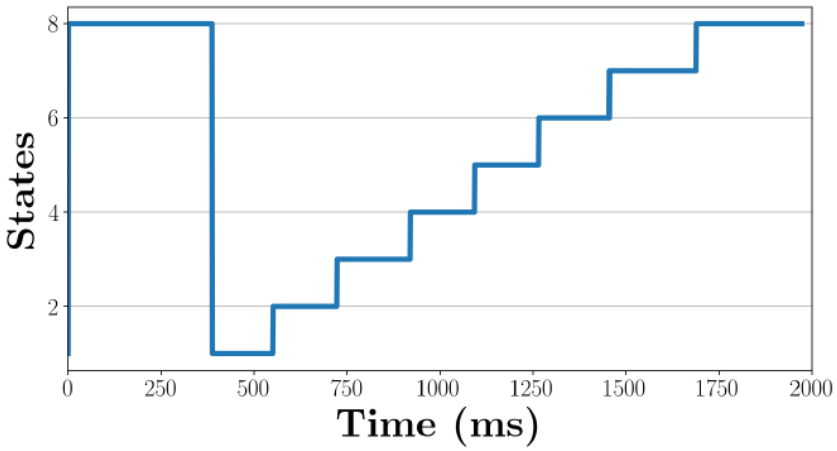
- Multiple neurons are fired at the same time.
- A neuron other than the expected one is fired.

These problems can cause confusion or increase in false negative/positive rate if the Markov Model is disregarded. The Markov part of the proposed framework ensures that the profiling methodology can overcome these problems. These inaccurate transitions are prevented via the thresholds employed by Algorithm 2 and limitations on the transitions imposed by the

proposed Markov Model. In Figure 60, the same plots are drawn for Basicmath, i.e. the second parallel unit of the neural network. Similarly, the same problems are observed with the Bit_count experiment, yet perfect monitoring is obtained with the proposed model.

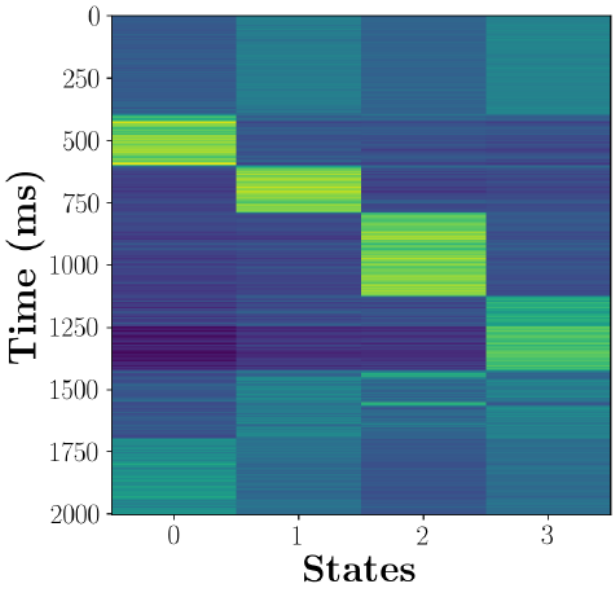


(a) The color becomes brighter as the output of the CNN gets larger. Likewise, the color gets darker as the corresponding neuron at the output layer gets smaller.

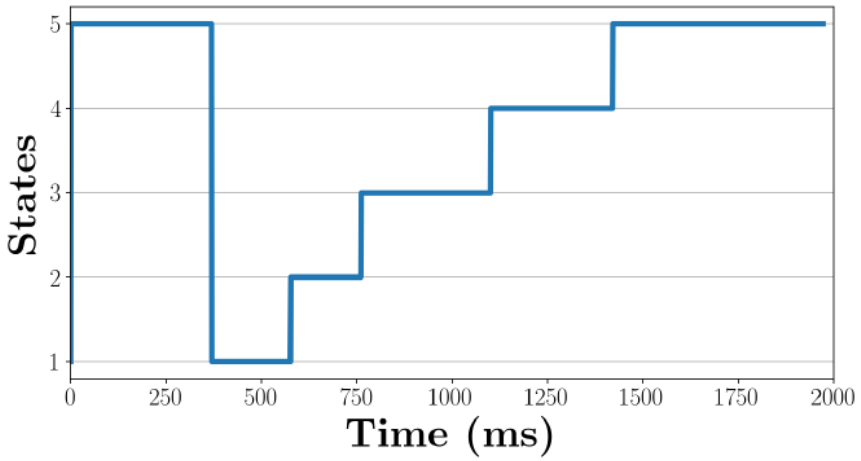


(b)

Figure 61 Profiling based on the CNN and Markov Model for Bit_count



(a)



(b)

Figure 62 Profiling based on the CNN and Markov Model for Basicmath

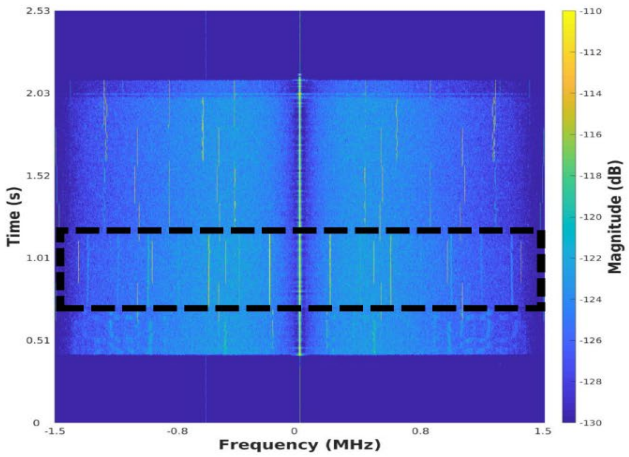
The main observations regarding the experiment can be listed as follows:

- Although the training is performed by utilizing single-core measurements, the proposed methodology can track and profile the program when multiple cores are active.
- The neural network realizes the components emanated due to other profiled programs and does not cause false positives even tough multiple cores are active.

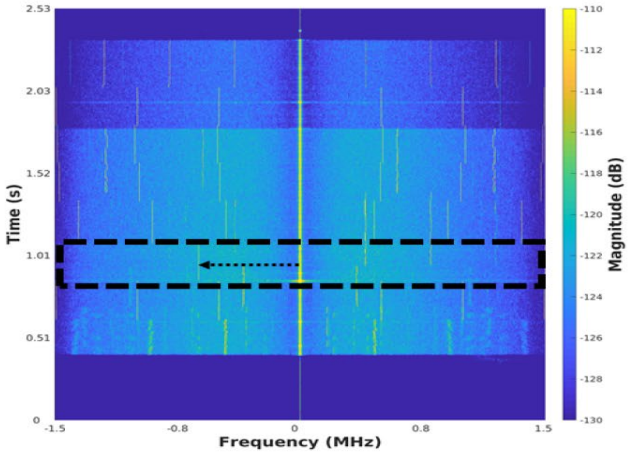
As the second step, we investigate the behaviour of the model when malware is injected to one of the programs. In that respect, the spectrograms of the programs in Figure 61 are examples of when Basicmath and Bit_count have malware, respectively. The regions corresponding to malware are indicated with a dotted rectangular. For the experiment where Basicmath has malware, we observe that there is frequency shift toward lower frequencies (toward the centre frequency) due to injection of extra code lines. This causes inner-loops to last longer to execute and some frequency components fade away. However, for the experiment where Bit_count has malware, we observe a shift away from centre frequency (to higher frequencies) (see dotted region of Figure 61b). This could be the result of inserting a new code which lasts less time, or the deletion of some code lines, etc.

In Figure 62, we provide the output streams of the Algorithm 2 for both parallel units when Basicmath has malware. In Figure 62a, the profiling results are given for the first parallel unit, whereas Figure 62b provides the results for the second parallel unit of the proposed model. As expected, the algorithm does not alert while profiling Bit_count as opposed to the other parallel

unit since the malware is injected into Basicmath benchmark. As another example, Figure 63 demonstrates the output stream of Algorithm 2 when Bit_count has malware. The same comments as before can be applied for this example. This time, the first parallel unit alerts the malware while the second parallel unit reports a benign program since the malware is injected to Bit_count.

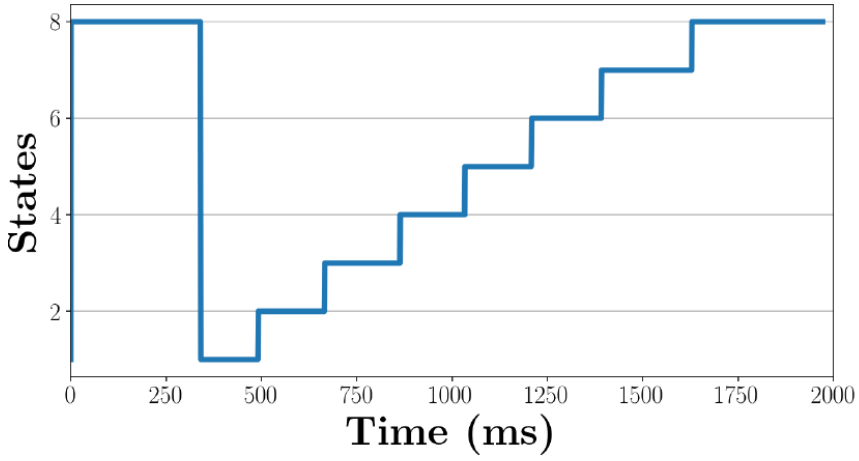


(a) When a malware is injected into `Basicmath`.

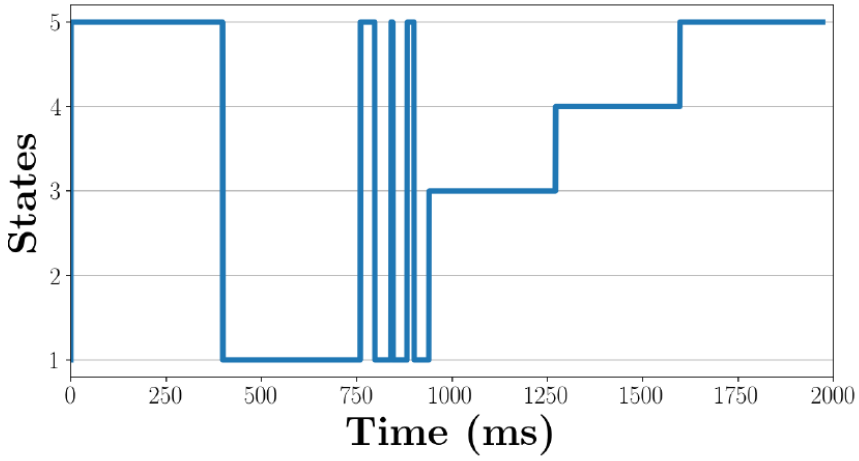


(b) When a malware is injected into `Bit_count`.

Figure 63 Hot regions when one of the programs has a malware

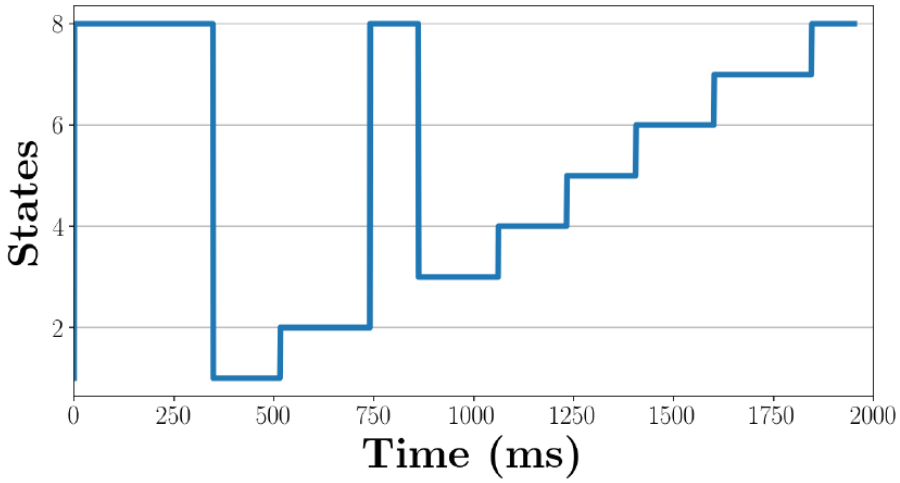


(a) First parallel unit.

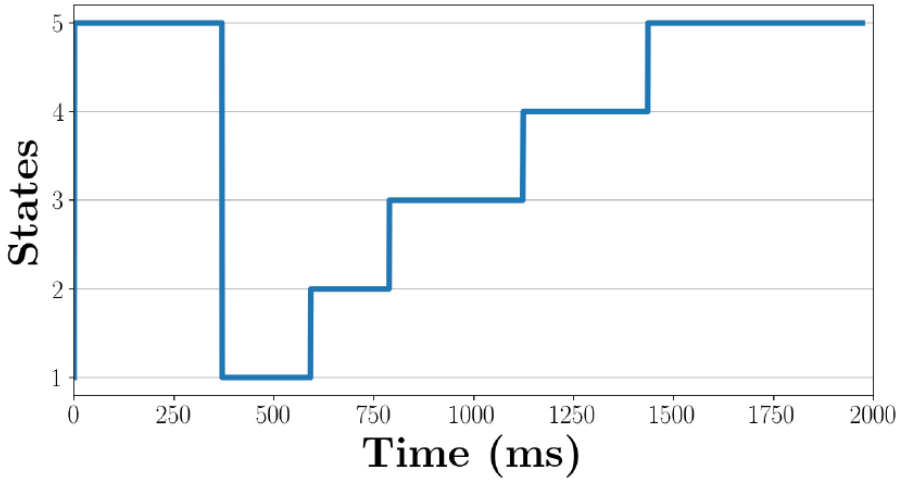


(b) Second parallel unit.

Figure 64 Profiling based on the CNN and Markov Model when Basicmath has malware



(a) First parallel unit.



(b) Second parallel unit.

Figure 65 Profiling based on the CNN and Markov Model when Bit_count has malware

As the final example, we investigate the behaviour of the model when the same algorithm is executed at both cores. This is the worst case scenario since both programs produce the same frequency components. We observe that the proposed model can still detect the malware because the distortion of the malware on the spectrum causes information loss for the CNN to extract features, as shown in Figure 64. This time, we are not able to reveal which program has the malware. However, after being aware of malware existence in one of the programs, we run these programs individually by activating only a single core of the device. Based on the outcomes of the single-core-experiments, we successfully identify which program has the malware. However, if the malware does not distort the spectrum severely and if the initialization of the same program on both cores is exactly same, it is possible for the framework to miss the malware. However, for the paper, we assume these two conditions do not occur at the same time.

- Main observations about these profiling experiments can be listed as follows:
- As long as the malware does not affect the whole spectrum, the proposed methodology can predict which of the programs has the malware. If it does affect the whole spectrum, it alerts an anomaly.
- The system can keep monitoring even after the part related to malware is executed.
- Combining Markov and CNN Models enables accurate prediction of the location of malware within the program.

After these results, we can provide examples of multi-core devices running different programs. Table VIII contains false positive (FP) and false negative (FN) rates for different number of active cores. Here, FP and FN represents false positive and negative rates, respectively. We observe that false negative is 0% for all cases, and false positive rate is less than 2%. These results show that the proposed methodology is a very powerful tool for monitoring multi-core devices.

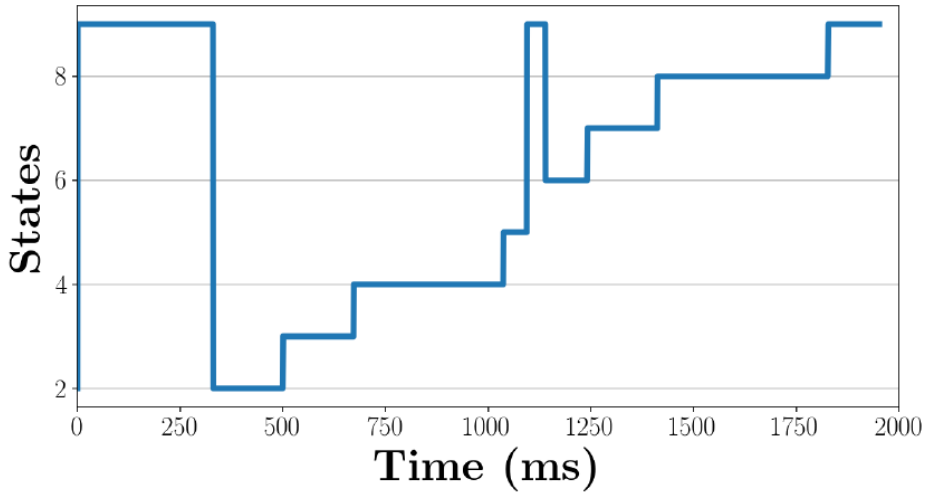


Figure 66 The states while profiling the system when two Bit counts are running and one of them has a malware

Table X The performance of the framework for different devices when multiple cores are active (%)

	# of Active Cores							
	1		2		3		4	
	FP	FN	FP	FN	FP	FN	FP	FN
A13-OLinuXino-Micro	0.1	0	-	-	-	-	-	-
A20-OLinuXino-LIME2	0.2	0	0.6	0	-	-	-	-
Alcatel Ideal	0.3	0	0.5	0	1.2	0	2	0

4.5 Tasks 3-5 – Results for Basic Block Tracking

We evaluate TESLA by monitoring two different devices executing three different benchmark applications. The evaluation matrix, the benchmark applications, and the experimental results are discussed in the following sections.

4.5.1. Evaluation Matrix

To evaluate TESLA, we compute the edit distance between the actual execution path and the reconstructed execution path. Specifically, we use Levenshtein distance [44] that computes the minimum number edits (insertions, deletions or substitutions) required to change reconstructed marker sequence to the actual marker sequence. We then compute the path reconstruction accuracy as using the following equation.

$$\text{Accuracy} = 1 - \frac{\text{Edit Distance}}{\text{Length of Actual Marker Sequence}} \tag{22}$$

We further compare the actual and the reconstructed timestamps. Specifically, we compute and report the absolute timing difference between the actual and the reconstructed markers. Note that, the edits are excluded from this comparison, as there are no timestamps for the edited (e.g., inserted or deleted) markers.

4.5.2 Benchmark Applications

We selected 3 benchmark applications (Print Tokens, Replace, and Schedule) from the SIR repository [45]. These applications are commonly used to evaluate techniques that analyze program execution. Table IX provides the size matrix for the benchmark applications.

Table XI Benchmark applications statistics

Benchmark	LOC	Basic Blocks
Print Tokens	464	178
Replace	495	245
Schedule	579	175

Moreover, these applications have many inputs, each taking a unique execution path through the CFG. We used disjoint sets of inputs for training and testing. For each application, we randomly selected 500 inputs for training, and 100 for testing. Table X summarizes training-testing split.

Table XII Training and testing executions

Benchmark	Training	Testing
Print Tokens	500	100
Replace	500	100
Schedule	500	100

We evaluate TESLA by executing these applications on two different devices: 1) FPGA device and 2) IoT device.

4.5.3 FPGA Device Monitoring

First, we monitored an Altera DE-1 prototype (Cyclone II FPGA) board. This device has a 50 MHz NIOS II soft processor. We placed a magnetic probe near the device to collect the EM side-channel signal. We then used an Agilent MXA N9020A spectrum analyzer to observe and demodulate the EM emanations. The demodulated signal is next passed through an anti-aliasing filter with 5 MHz bandwidth. Finally, we sampled the filtered signal at 12.8 MHz sampling rate, and analyzed the digitized signal using TESLA.

Table XI summarizes the mean accuracy. We observe that TESLA achieves excellent accuracy for monitoring all three benchmark applications, with roughly 99% accuracy for Print Tokens and Replace, and near-perfect accuracy for Schedule. We also report the mean timing difference of the predicted timestamps in Table XII. For Print Tokens and Schedule the mean timing difference is less than 1 sample. However, for Replace, the mean timing difference is roughly 4 samples.

Note that, at the experimental sampling rate (12.8 MHz), 1 sample is equivalent to 78:125 ns. Thus, all timing estimations are very precise.

Table XIII Mean accuracy for FPGA

Benchmark	Path Prediction Accuracy
Print Tokens	98.7%
Replace	99.1%
Schedule	99.8%

Table XIV Mean timing difference for FPGA

Benchmark	Mean Timing Difference
Print Tokens	0.98 samples
Replace	4.13 samples
Schedule	0.88 samples

Monitoring from Distance: We further evaluate TESLA by monitoring the FPGA device from 1 m distance using a panel antenna. Figure 65 shows the experimental setup. We summarize the mean accuracy in Table V. TESLA achieves better than 95% accuracy on all three benchmarks. In fact, for Print Tokens and Schedule, TESLA achieves roughly 99% accuracy.

Table XV Mean accuracy at 1 m

Benchmark	Path Prediction Accuracy
Print Tokens	98.68%
Replace	95.14%
Schedule	99.40%

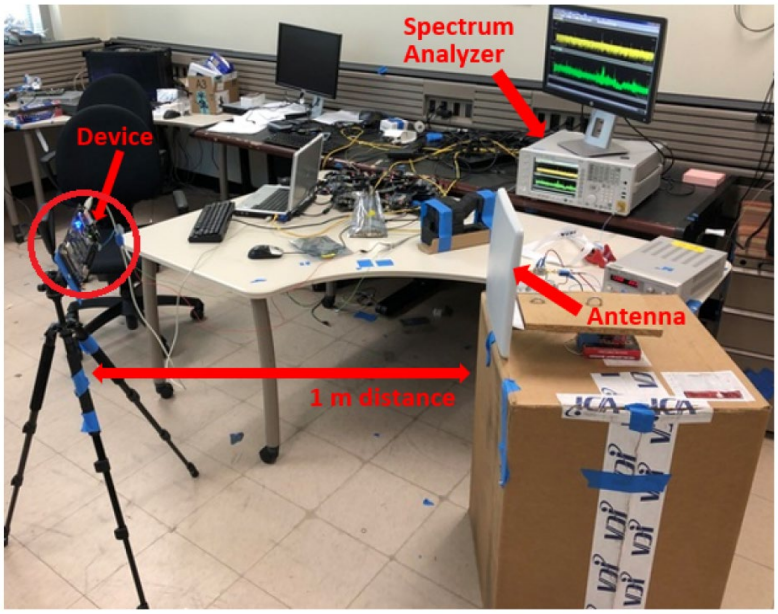


Figure 67 Experimental setup: monitoring from 1 m distance

Table XIV shows the mean timing difference for the predicted marker timestamps. While the timing differences are slightly higher than that of with probe, predicted timestamps are still very precise, and within a few samples.

While TESLA demonstrates excellent performance from 1 m distance, we notice slight degradation in accuracy compared to that of with probe (i.e., at 1 cm distance). This degradation is due to the lower SNR at distance, and can be improved by using high-gain antennas and/or low-noise amplifiers.

Table XVI Mean timing difference at 1 m

Benchmark	Mean Timing Difference
Print Tokens	3.78 samples
Replace	5.56 samples
Schedule	1.32 samples

4.5.4 IoT Device Monitoring

We demonstrate the robustness of TESLA by monitoring an A13-OLinuXino IoT development board. This device has a 1 GHz Cortex A8 ARM processor [26]. Unlike the FPGA device, A13-OLinuXino runs on a Debian Linux operating system. We collected the EM side-channel signal by placing a magnetic probe near the microprocessor. The signal was recorded and demodulated using a spectrum analyzer (Agilent MXA N9020A). We then digitized the signal by

passing it through an anti-aliasing filter with 20 MHz bandwidth, and sampling at 51.2 MHz sampling rate.

Table XVII Mean accuracy for IoT device

Benchmark	Path Prediction Accuracy
Print Tokens	94.15%
Replace	96.85%
Schedule	95.91%

Table XV shows the accuracy of TESLA for monitoring the IoT device. TESLA demonstrates high accuracy on all three benchmark applications; 94.15% on Print Tokens, 96.85% on Replace, and 95.91% on Schedule. Note that, TESLA achieves even higher accuracy (roughly 99%) for monitoring FPGA device. However, A13-OLinuXino has a much faster processor (1 GHz compared to FPGA's 50 MHz), which makes fine-grained execution monitoring more challenging. Furthermore, the operating system on A13-OLinuXino leads to more variations between training and testing executions. This, in turn, can cause performance degradation. As such, TESLA's performance on monitoring the IoT device is impressive.

Furthermore, the timing differences reported in Table XVI demonstrate that TESLA predicted timestamps are also quite precise. The mean timing difference for all three benchmark

applications is within 10 samples. Note that, in our experiments (at 51.2 MHz sampling rate), each sample is equivalent to 19:5 ns.

Table XVIII Mean timing difference for IoT device

Benchmark	Mean Timing Difference
Print Tokens	7.68 samples
Replace	10.33 samples
Schedule	6.92 samples

Monitoring from Distance: We also evaluate TESLA by monitoring the IoT device from distance. For this, we placed a slot antenna at 1 m distance from the device.

Table XVII shows that TESLA achieves roughly 90% mean accuracy on all three benchmarks. Furthermore, Table XVIII reports the mean timing difference. We also notice some performance degradation at distance. This is due to lower SNR that affects the signal matching adversely.

Table XIX Mean accuracy at 1 m

Benchmark	Path Prediction Accuracy
Print Tokens	89.87%
Replace	90.79%
Schedule	90.40%

Table XX Mean timing difference at 1 m

Benchmark	Mean Timing Difference
Print Tokens	11.40 samples
Replace	33.66 samples
Schedule	7.53 samples

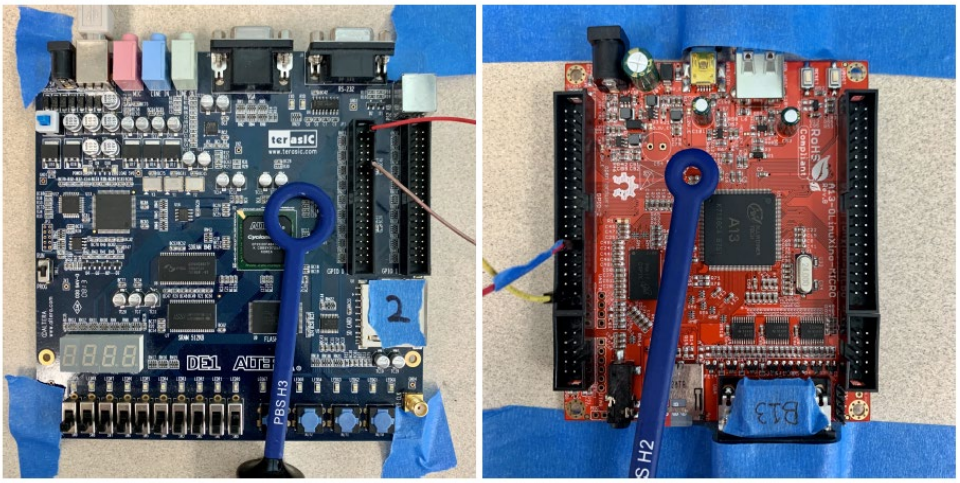
4.6 Task6 – Results for Single Instruction Tracking

In this section, we explain our experimental setup and provide the results for instruction type determination and permutation tracking of instruction types.

4.6.1 Experimental Setup

To demonstrate the feasibility of the proposed methodology, we experiment on two devices. The first device is Intel's DE1 Altera FPGA Board that has Altera NIOS-II (soft) processor. This processor is a general purpose RISC (reduced instruction set computer) processor that implements Nios-II architecture with 6 pipeline stages. The operating clock frequency is 50 MHz and this board does not have a present operating system. The second device is A13-OLinuXino, which is a low-cost embedded Linux mini-computer that has ARM Cortex A8 processor that operates at 1 GHz clock frequency. The processor implements ARMv7-A architecture and is an in-order, dual-issue, superscalar microprocessor with 13-stage main integer pipeline. In the remainder of the text, we refer to the first and second devices as the DE1 device, and the A13 device, respectively.

To record the EM emanations, we use Aaronia's H3 nearfield magnetic probe for the DE1 device and H2 near-field magnetic probe for the A13 device. These probes are chosen so that the resonance frequency of the probes are aligned with the operating clock frequencies of the devices. We locate the probes on the pin edges of the processors as shown in Figure 66. Also note that two GPIO pins of these devices are utilized to record the marker signal. Measurements are obtained with Keysight's DSOS804A high-definition oscilloscope at 10 GHz sampling rate.



(a) DE1 Device. (b) A13 Device.

Figure 68 Experimental setup used for EM emanation recordings

4.6.2 Instruction Type Determination Results

To record the EM emanations, we use Aaronia's H3 nearfield magnetic probe for the DE1 device and H2 near-field magnetic probe for the A13 device. These probes are chosen so that the resonance frequency of the probes are aligned with the operating clock frequencies of the devices. We locate the probes on the pin edges of the processors as shown in Figure 66. Also note that two GPIO pins of these devices are utilized to record the marker signal. Measurements are obtained with Keysight's DSOS804A high-definition oscilloscope at 10 GHz sampling rate.

After inspecting the instruction set for both devices, we select 17 instructions per each device for investigation that are listed in Table XIX with their corresponding abbreviations. Note

that these are frequently used instructions including mathematical operations such as addition and multiplication; logical operations such as AND, OR; and memory access instructions such as load and store, etc.

After selecting the instructions, we generate micro benchmarks for each of them that include N = 10 and N = 100 times repetitions for the DE1 and A13 devices, respectively. The reason for the higher N value of the A13 device micro benchmarks is the higher operating clock frequency of the device. Next, we record the EM emanations and obtain the EM signatures for different instructions. Figures 67 and 68 present examples of the obtained EM signatures of different instructions for the DE1 and A13 devices, respectively. Note that each subfigure plots several EM emanation traces obtained for different executions of the given instruction. One can easily observe that these traces are highly aligned indicating that the EM signatures of the instruction sequences are relatively similar for successive executions.

In Figure 67, we observe that some instructions such as LDW and MULI have significantly different EM signatures that differ both in length and shape, whereas some instructions such as ADD and SUB have very similar EM signatures in both length and shape. Similar conclusions can be obtained for Figure 68, as well. One should note that these conclusions validate our initial assumption that some instructions have similar EM signatures while some have significantly different EM signatures. Therefore, our methodology also provides the capability of decreasing

the entropy of the instructions. We also realize that the instructions with similar EM signatures are those that have similar physical implementations such as addition and subtraction.

Table XXI Investigated instructions

DE1 Device			A13 Device		
	Assembly Code	Description		Assembly Code	Description
ADDI	`addi r20, r20,171`	Add immediate value to register	**ADDI**	`add r3, r3, #1`	Add immediate value to register
ADD	`add r20, r20, r16`	Add two register values	**ADD**	`add r3, r3, r3`	Add two register values
SUBI	`subi r20, r20,173`	Subtract immediate value from register value	**SUBI**	`sub r3, r3, #1`	Subtract immediate value from register value
SUB	`sub r20, r20, r16`	Subtract two register values	**SUB**	`sub r3, r3, r3`	Subtract two register value
LDW	`ldw r20, 0(r21)`	Load 32-bit word from memory to register	**LDR**	`ldr r3, [r1]`	Load a word from memory to register
STW	`stw r20, 0(r21)`	Store word from register to memory	**STR**	`str r3, [r1]`	Store a word from register to memory
MUL	`mul r20, r20, r16`	Multiply two register values	**MUL**	`mul r3, r3, r3`	Multiply two register values
MULI	`muli r20, r20, 173`	Multiply immediate value with register value	**SMULL**	`smull r3, r2, r3, r3`	Multiply two 32-bit signed values
DIV	`div r20, r20, r20`	Divide two register values	**UMULL**	`umull r3, r2, r3, r3`	Multiply two 32-bit unsigned values
OR	`or r20, r20, r16`	Bitwise logical OR operation of register values	**ORR**	`orr r3, r3, r2`	Bitwise logical OR operation of register values
ORI	`ori r20, r20, 173`	Bitwise logical OR operation of register and immediate values	**ORRI**	`orr r3, r3, #1`	Bitwise logical OR operation of register and immediate values
MOVI	`movi r20, 173`	Move immediate value to register	**MOVI**	`mov r3, #1`	Move immediate value to register
MOV	`movi r20, r16`	Move register value to register	**ANDI**	`and r3, r3, #1`	Bitwise logical AND operation of register and immediate values
AND	`and r20, r20, r20`	Bitwise logical AND operation of register values	**AND**	`and r3, r3, r2`	Bitwise logical AND operation of register values
CMPEQ	`cmpeq r20, r20, r16`	Compare if register values are equal	**CMP**	`cmp r3, r0`	Subtract register values to compare
XOR	`xor r20, r20, r16`	Bitwise logical XOR operation of register values	**CMPI**	`cmpi r3, #1`	Subtract immediate value from register value to compare
XORI	`xori r20, r20, 173`	Bitwise logical XOR operation of register and immediate values	**NOP**	`nop`	No operation

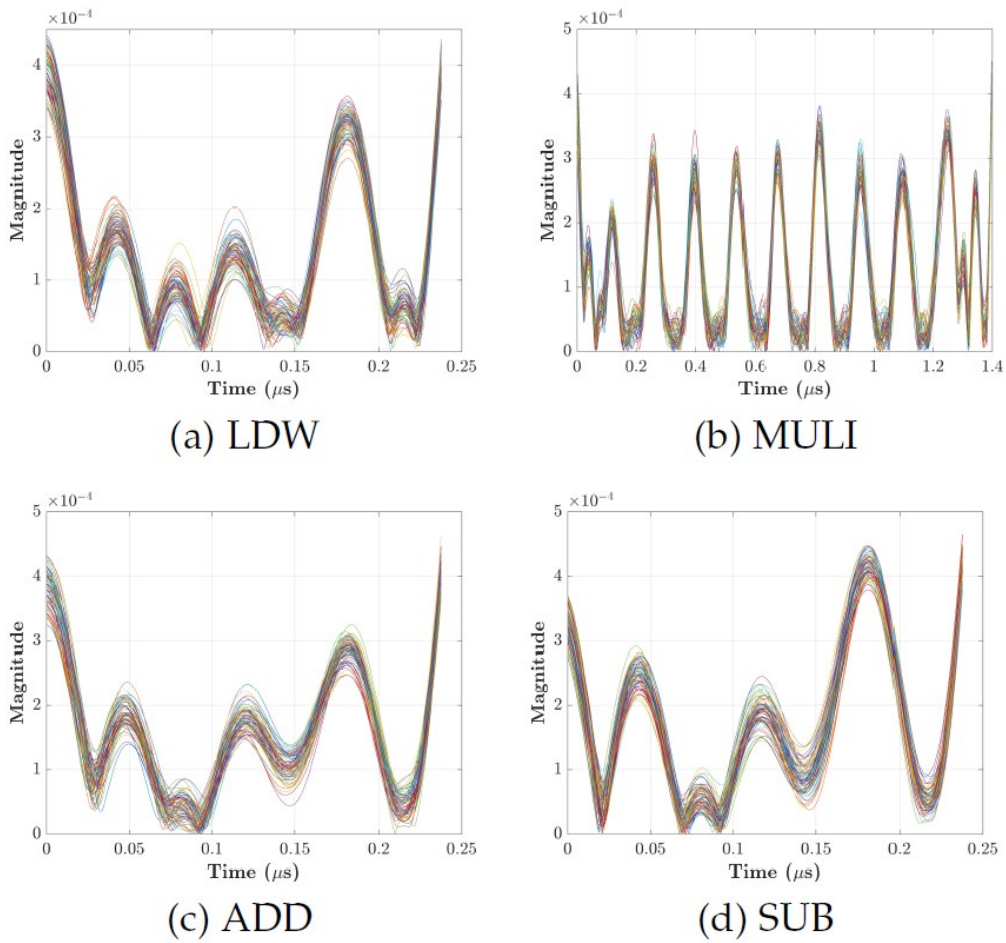


Figure 69 Obtained EM signatures of several instructions for the DE1device

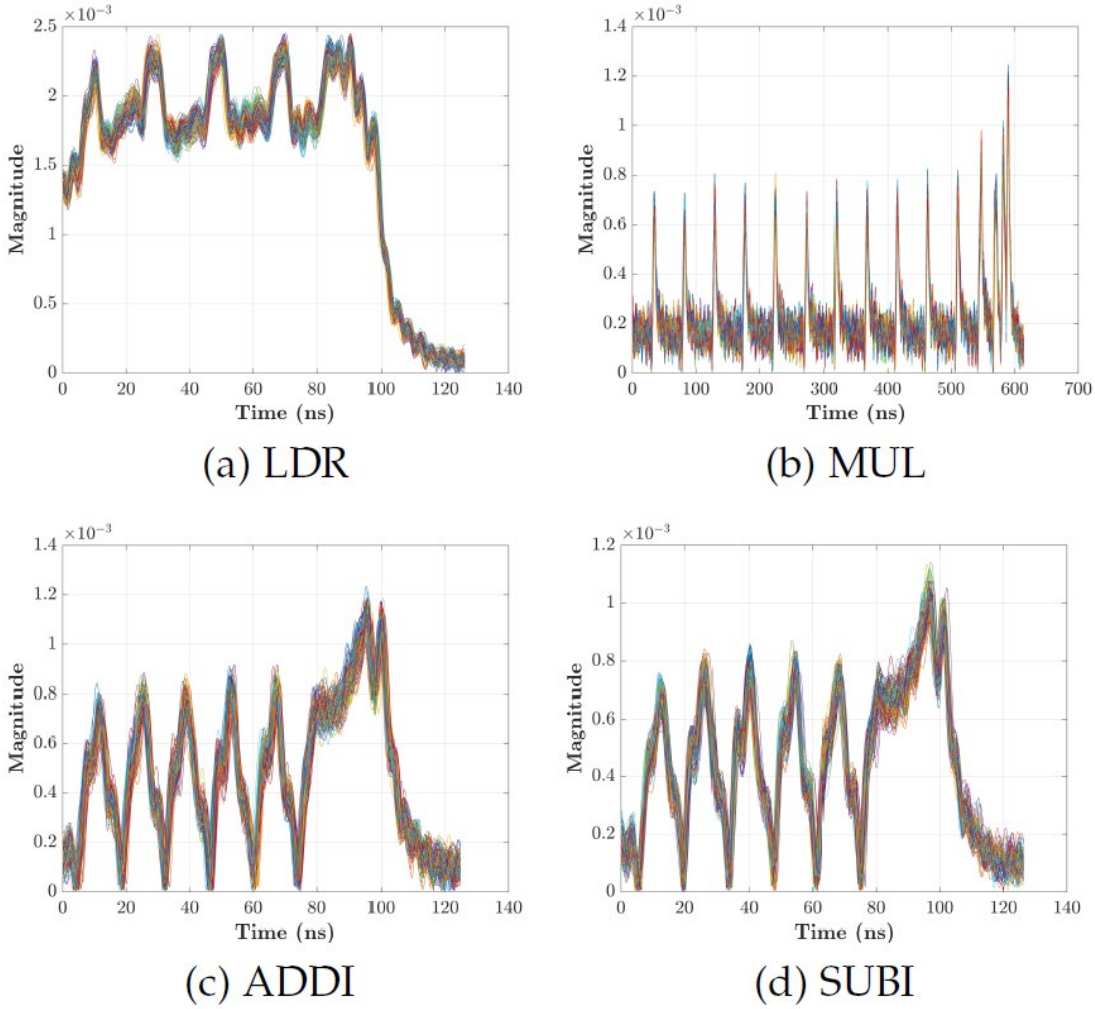


Figure 70 Obtained EM signatures of several instructions for the A13device

As can be seen in Figure 68, the EM signature of LDW is significantly different than MUL, ADDI and SUBI. This difference reflects the significance of the additional memory system

pipeline present in the A13 device for memory access instructions. Although the EM signatures are obtained by repeating the same instruction N times, we are not observing the repetition of the same pattern N times in the EM signature. We note that the beginning and ending parts of the signatures are significantly different than the middle parts, where we can observe the same kind of pattern repetition. This observation emphasizes the significance of the pipeline effect caused by the instructions that come before and after the instruction sequence. From these observations, we conclude that the EM signatures are indicative of the pipeline structure. These observations also prove that each stage of a pipeline emits EM signals while executing instructions.

After obtaining the EM signatures, we generate the correlation matrices for the DE1 and A13 devices as shown in Figure 69. In Figure 69, brighter colors indicate higher correlation, hence higher similarity. Then, we convert this correlation matrix to a distance matrix and cluster these instructions with average-link hierarchical clustering. The resulting dendrograms are presented in Figure 70. Note that the bottom part of the dendrograms start with all instructions, and as it goes to the top, these instructions are merged pair-wise so that they are in the same cluster until all instructions are merged at the top. By investigating the correlation matrices and dendrograms, we set the number of clusters, K, to be 4 in both cases because for both cases when K is set to 4, resulting clusters include instructions that are similar in operation such as MULI and MUL. The clusters are indicated with different branch colors. Red, blue, green and black colors represent A, B, C and D type clusters, respectively.

For the DE1 device, A-type instructions include memory-access operations (LDR, STR) as well as the clock-cycle arithmetic and logic operations that use register sources (ADD, SUB, AND, CMPEQ, MOV, OR), there are also two exceptions (MOVI, ORI) that use immediate values. B-type instructions are the arithmetic and logic operations that use immediate values (ADDI, SUBI and XORI). C-type instructions are the multiplication operations (MULI and MUL) and D-type instructions are the division operation (DIV).

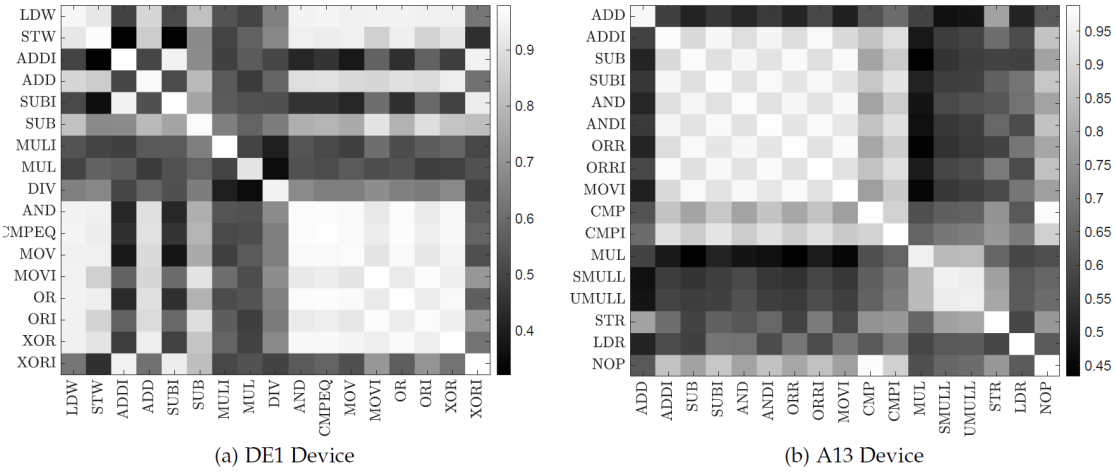


Figure 71 Correlation matrices for DE1 and A13 devices

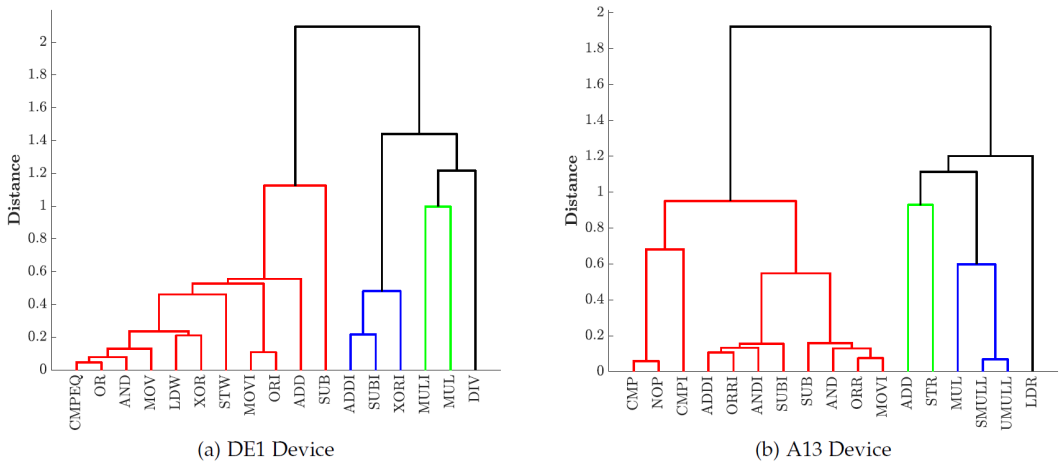


Figure 72 Dendrogram of the instructions obtained with hierarchical clustering for DE1 and A13 devices. Different colors represent the clusters

For the A13 device, A-type instructions include all clock-cycle arithmetic and logic operations that are either using register values or immediate values. One exception is the ADD instruction that uses register values, which is clustered as a C-type instruction along with the store instruction (STR). B-type instructions are the multiplication operations (MUL, SMULL, UMULL), and D-type instructions are the load operation (LDR).

As it has been discussed earlier, the clusters reflect the structure of the pipeline. For example, load and store operations include the memory address location calculations which are addition and subtraction operations. Therefore, these instructions have the same type as addition

and subtraction for the DE1 device. However, A13 implements an external memory-access pipeline, which results in separate clusters for load and store operations. Similarly, we note that different variants of multiplication operations are clustered in the same group for both devices.

4.6.3 Permutation Tracking Results

After determining the instruction types, we pick one instruction from each type to represent the entire type. These instructions are LDW, ADDI, MULI and DIV for the DE1 device, and ADD, MUL, STR and LDR for the A13 device representing A, B, C and D in the given order. Using these instructions, we generate micro benchmarks, record EM emanations and generate EM signatures for different N values.

Figure 71 and Figure 72 present example EM signatures for different permutations and N values obtained from the DE1 and A13 devices. For N = 1, the plotted EM signatures of the permutations $(ABCD)_1$ and $(DCBA)_1$ are visually different from each other for both devices. However, we note that, when N = 1, the EM signatures for the A13 device has a large variance among different executions, whereas this variance is much smaller for N = 10 case of the same device. In Figure 71, when N = 10, we can observe a pattern that is repeated 10 times, but this repetition is not present in Figure 72 when N = 10. This difference can be explained by the difference of the pipeline lengths: DE1's pipeline length (6 stages) is shorter than A13's pipeline length (13 stages). Therefore, a permutation block of length 4 instructions is more capable including the contributions from the pipeline stages of the DE1 device and get the EM signature

into a steady state, whereas the EM signature for A13 does not reach the steady state with 10 repetitions.

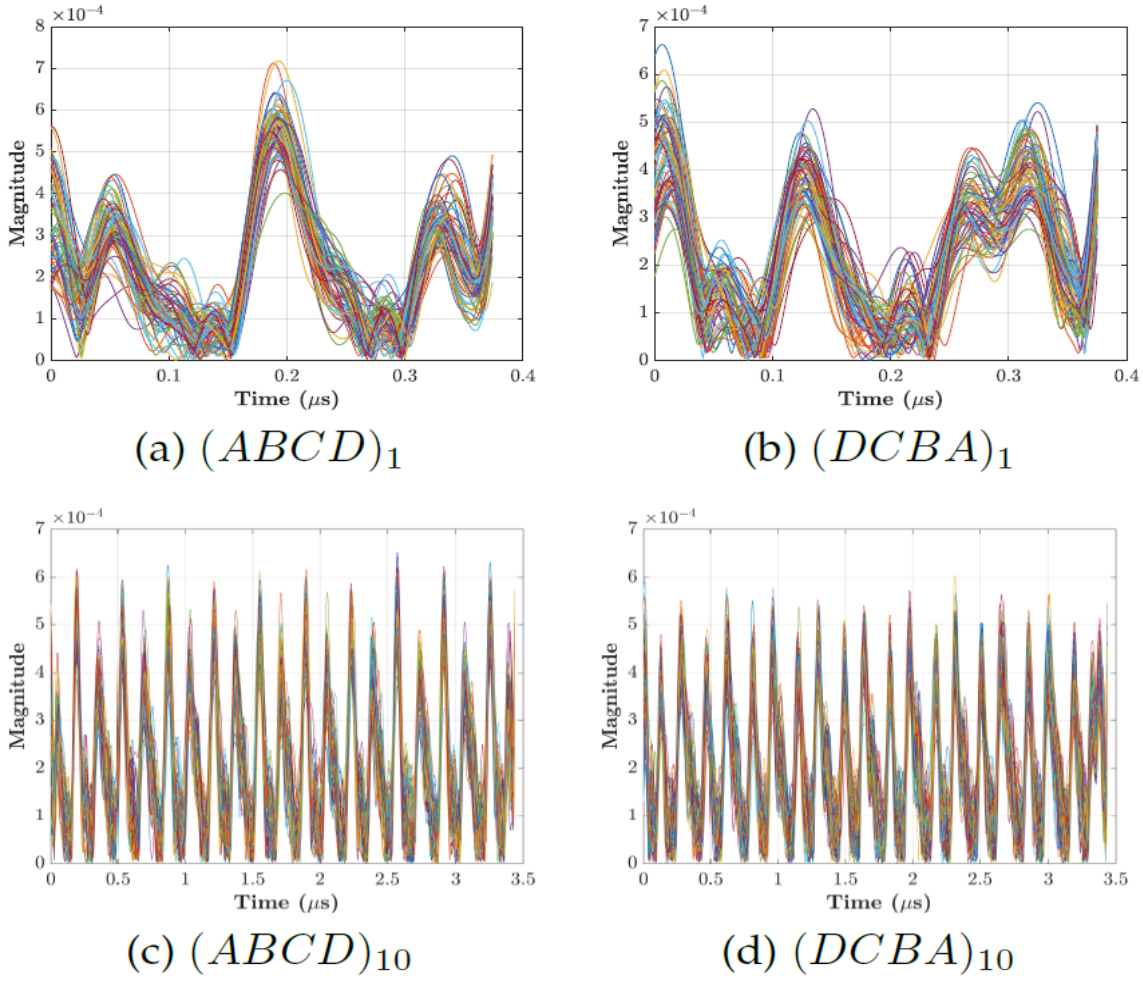


Figure 73 Sample EM signatures of permutations with different N values for the DE1 device

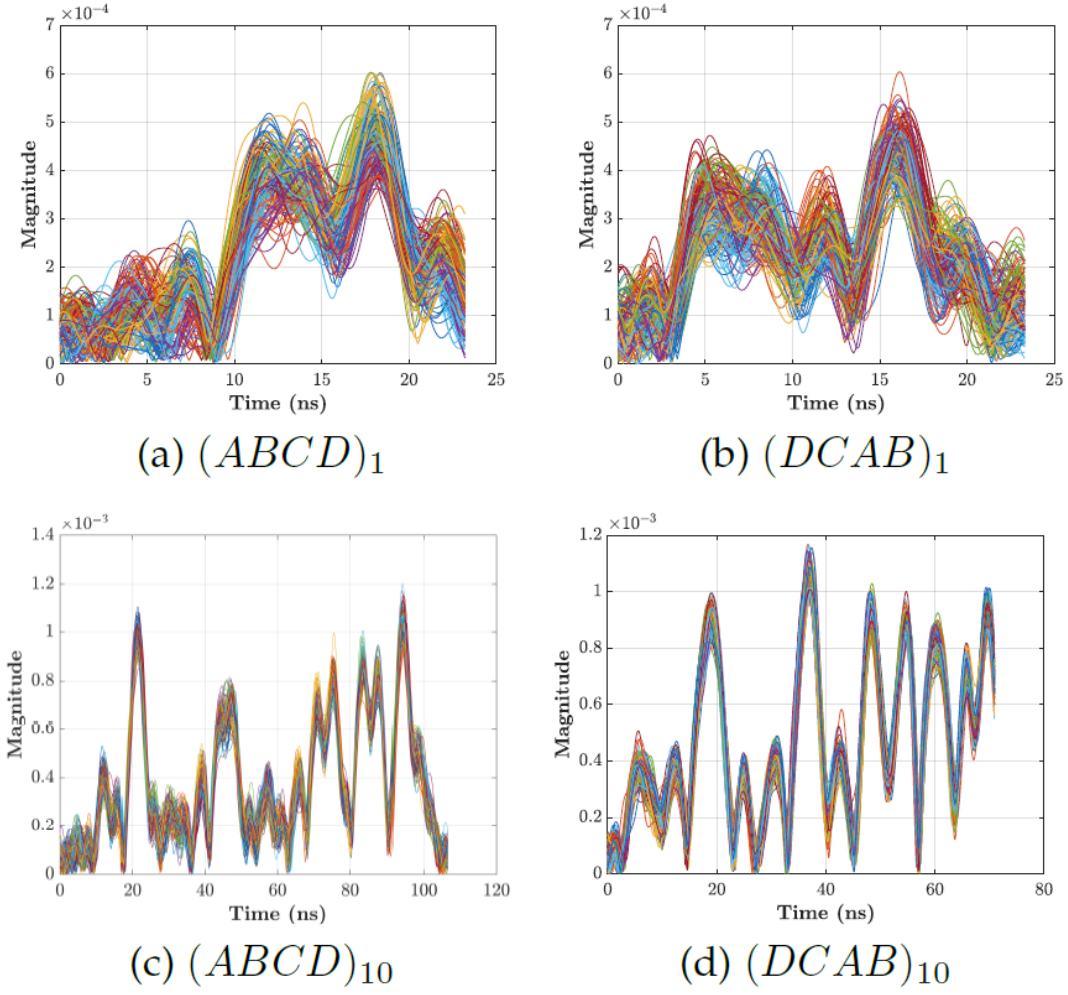


Figure 74 Sample EM signatures of permutations with different N values for the A13 device

Note that for both devices, we cannot visually identify the locations of the A, B, C, and D instruction types from the EM signatures. As discussed earlier, this is because single execution of the instruction does not create significant variation in the signature and due to the pipeline, the variation generated by execution of single instruction is distributed to different stages of the pipeline. To test this, while executing the permutation only once (N = 1), we repeat each instruction within the block 10 and 100 times for the DE1 and A13 devices, respectively. The EM signatures obtained with this repetition are shown in Figure 73 and Figure 74. In Figure 73, we see that the instruction blocks A and B that appear in the beginning for ABCD permutation can be identified at the end of DCAB permutation. Note that identifying C and D precisely is still not possible. In Figure 74, we can identify all instruction type blocks clearly as indicated in the figure. These results show that, although single execution of an instruction does not generate an identifiable waveform pattern, several consecutive executions can result in distinct waveforms. Please note that this is just an observation and cannot be directly used for testing purposes because enforcing the repetition of the same instruction within the blocks is not very realistic for many programs and, therefore, is not practically applicable.

We generate templates for each permutation using Measurement 1, and predict the permutation order of the snippets obtained from Measurement 2. We use 50 snippets per each permutation resulting in a total of 1200 testing snippets. To observe the impact of the utilized low-pass filter bandwidth on the accuracy and find the optimum bandwidth, we report the accuracy for different bandwidth values as shown in Figure 75. Note that the accuracy is calculated as the ratio

of number of the correctly classified permutations to the total number of testing traces. We note that the highest accuracy is obtained at 20 MHz for the DE1 device and 400 MHz for the A13 device. These are the optimum bandwidths that coincide with $2f_c=5$ where f_c is the clock frequency that corresponds to 50 MHz for DE1, and 1 GHz for A13. This is an interesting observation that the more bandwidth does not necessarily translate into more information and higher accuracy, and the optimum bandwidth is scaled with the clock frequency. Apart from the increase in the noise energy, the major reason behind the accuracy drop beyond the optimum bandwidth is the aliasing effect coming from the neighbouring harmonics of the clock frequency.

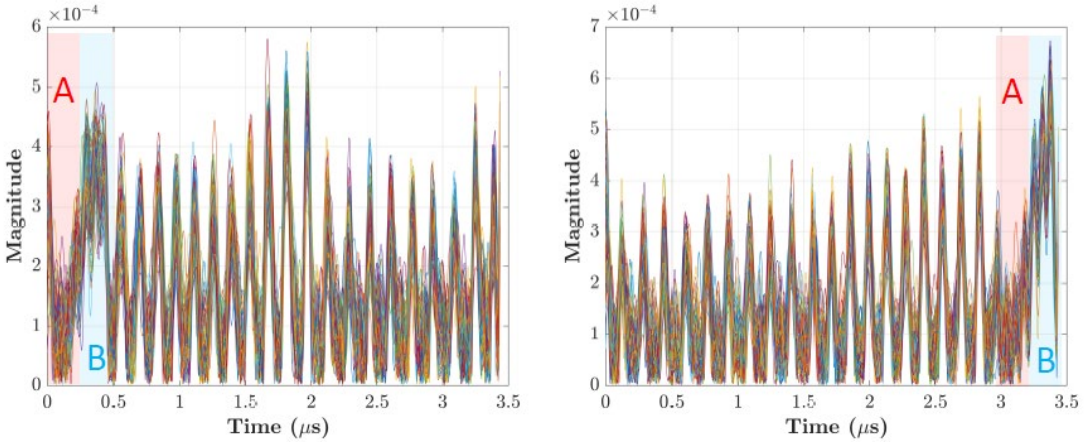


Figure 75 Sample EM signatures when instruction types are repeated 10 times within the permutation block for the DE1 device

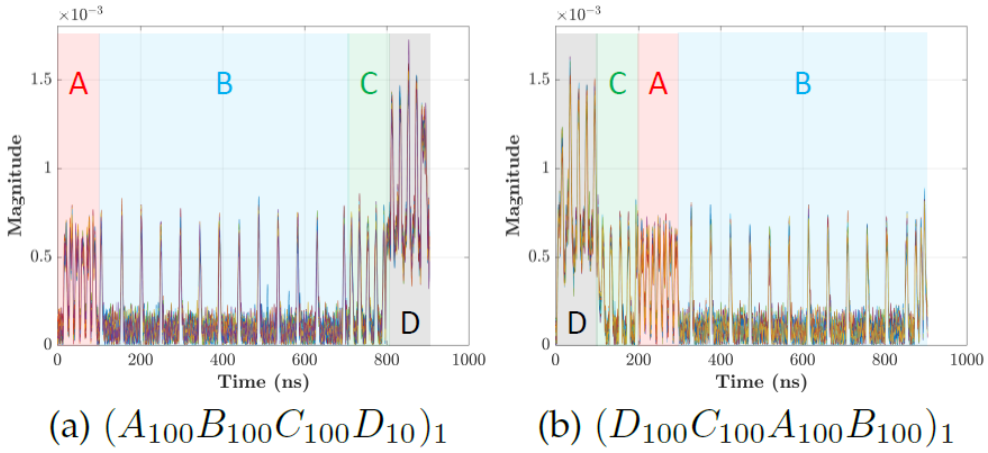


Figure 76 Sample EM signatures when instruction types are repeated 100 times within the permutation block for the A13 device

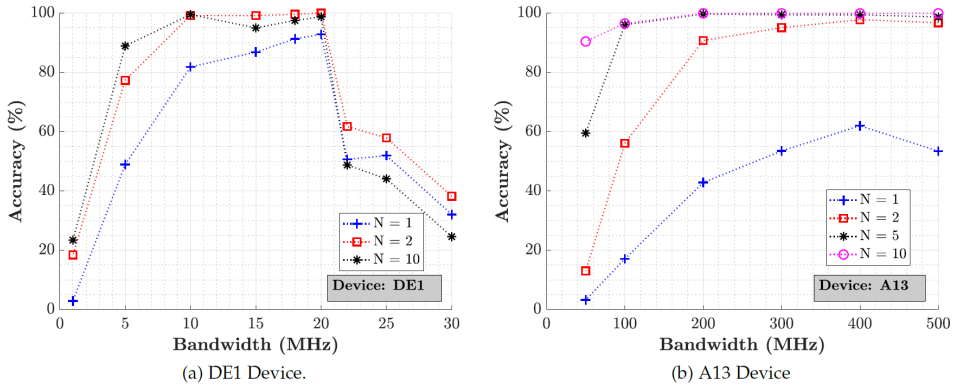
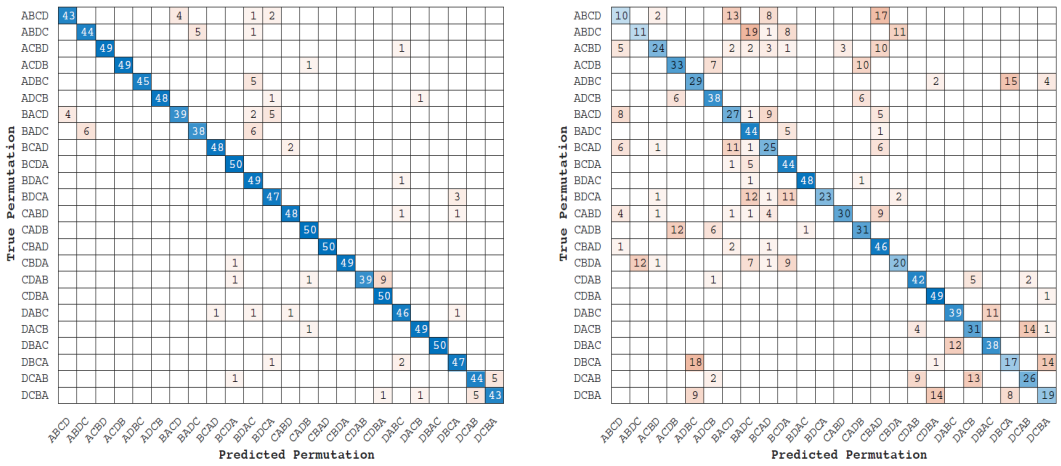


Figure 77 Impact of *N* value on accuracy

In Figure 75, we observe that the optimum accuracies when N = 1 are 92.8% and 62% for DE1 and A13, respectively. For higher values of N, the accuracy significantly improves. For N =

2, accuracy gets as high as 100% for DE1 and 97.8% for A13. To investigate N = 1 case, we provide the confusion matrices in Figure 76. Note that both matrices are mostly diagonally dominant. When we investigate the correlation matrix for the A13 device, we observe that some permutations are more vulnerable to misclassification such as ABCD and ABDC, whereas some permutations are almost perfectly classified such as BDAC and CDAB. This shows that the ordering of the instructions changes the strength of the EM signature variation.

To demonstrate the improvement with larger N, we provide the correlation matrix for the A13 device when N = 2 in Figure 77. Note that a single increment in N significantly improves the accuracy from 62% to 97.8%. Here we would like to emphasize the trade-off between the number of repetitions and the accuracy. Including repeated versions of the permutation blocks is significantly increasing the detection accuracy in the expense of limiting the applicability of the proposed method. Therefore, if the system or program under investigation has repetitive nature, this method can be modified for better performance.



(a) DE1 Device. (b) A13 Device

Figure 78 Confusion charts for *N*=1

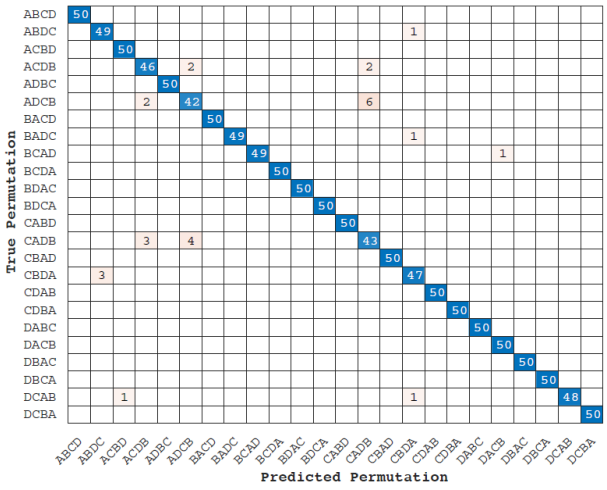


Figure 79 Confusion chart for N = 2, A13 Device

4.6.4 Further Evaluation of Robustness

In this section, we test our methodology's robustness and extensions. Firstly, we evaluate the performance for different SNR levels. Next, we extend the tracking ability to instructions rather than instruction types. The results are explained in the following sections.

Performance for different SNR levels: To evaluate the performance for different SNR levels, we assume that the measurement channel is corrupted by additive white Gaussian noise (AWGN). The calculation of SNR requires the signal power without any noise. Since EM side-channels are unintentionally generated, we cannot control or measure the signal power by isolating the noise. Therefore, we obtain measurements at locations with strongest EM emanations and use the power of the signals obtained from these measurements as the referenced signal power, therefore assuming an SNR level of infinity. After obtaining these traces, we introduce additive white Gaussian noise with different noise powers to the testing traces.

Let **x** be a vector representing a testing trace with a length of L samples, and xi be the sample values of this trace such that $\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \dots & x_L \end{bmatrix}$. To add the noise, we first calculate PS, the signal power of the vector, as

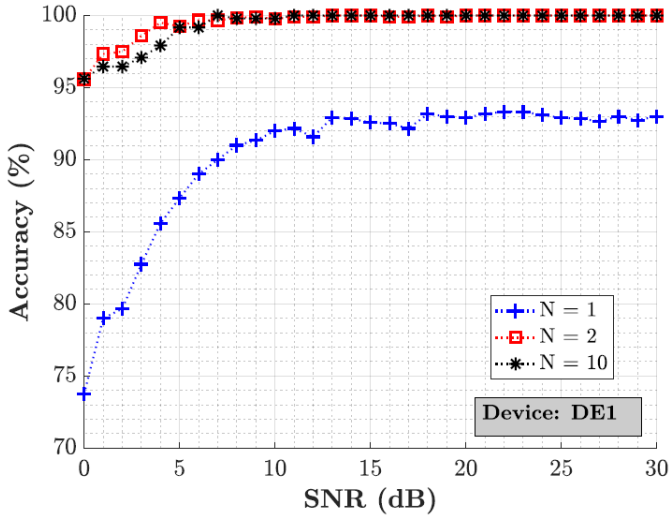
$$P_s = \frac{1}{L}\sum_{i=1}^{L} |x_i|^2. \tag{23}$$

Then we add AWGN to each sample of x and obtain the new signal *y* that is corrupted by AWGN. The elements of **y**, y_i, are obtained as

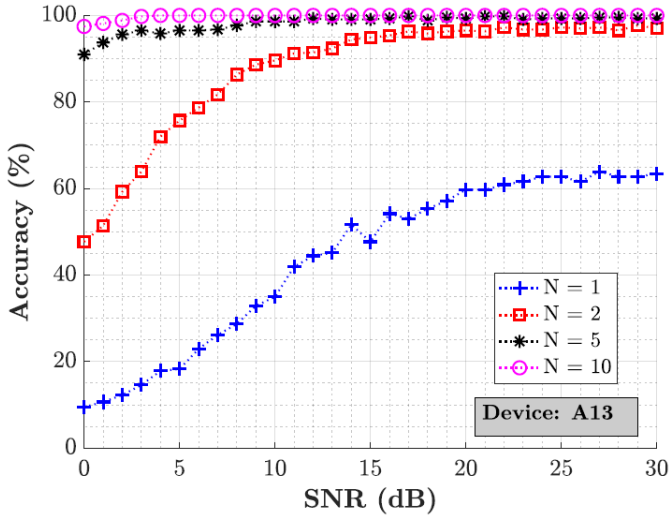
$$y_i = x_i + z_i \quad \text{for} \quad 1 \leq i \leq L, \tag{24}$$

Where $z_i \sim \mathcal{N}(0, \frac{P_s}{SNR_{lin}})$ is independent and identically distributed realizations of a random variable that has a zero mean Gaussian (normal) distribution with $\frac{P_s}{SNR_{lin}}$ variance, and SNR_{lin} represents SNR in linear scale.

One should note that we add AWGN to all testing traces, but the EM templates are kept as originals. Then, we perform the same testing procedure for different SNR values with the optimum low-pass filter bandwidths. The results are shown in Figure 78, where we observe that the accuracy decreases for low SNR levels, as expected. We note that the performance converges for SNR values higher than 15 dB. Also, we observe that the permutations with larger N are generally more resistant to AWGN.



(a) DE1 Device.



(b) A13 Device.

Figure 80 Impact of SNR on accuracy for permutations of different instructions for different N values

4.6.5 Permutations of Instructions from the Same Instruction Type

In this section, our objective is to extend the applicability of the permutation tracking from tracking different instruction types to tracking instructions of the same instruction type. To do so, we pick 4 instructions from instruction type A for both devices. For the DE1, we pick ADD, AND, MOV and CMPEQ; for the A13, we pick ADD, AND, MOV and CMP. Note that these instructions use the same destination registers, register sources and immediate values, therefore we minimize the variation that might be caused by data values or register differences. Using these 4 instructions, we repeat the experiments and report the accuracies in Figure 79. Note that, the accuracies for N = 1 are very low because these instructions are from the same type and their permutations have very similar shapes. This low accuracy verifies that the instructions are clustered correctly. On the other hand, we note that if we can afford to repeat the permutation sequence for several times, in other words as we increase N, the accuracy successively increases. In fact, for N = 100, accuracy for the DE1 and A13 devices get as high as 92.4% and 98%, which are relatively high accuracies considering the similarity of the instruction signatures. To be able to determine the differences between the instructions from the same cluster, we need more observations.

This points to the trade-off between better instruction resolution and number of observations. This also shows that higher frequency bandwidth does not necessarily provide more information to increase the accuracy rate.

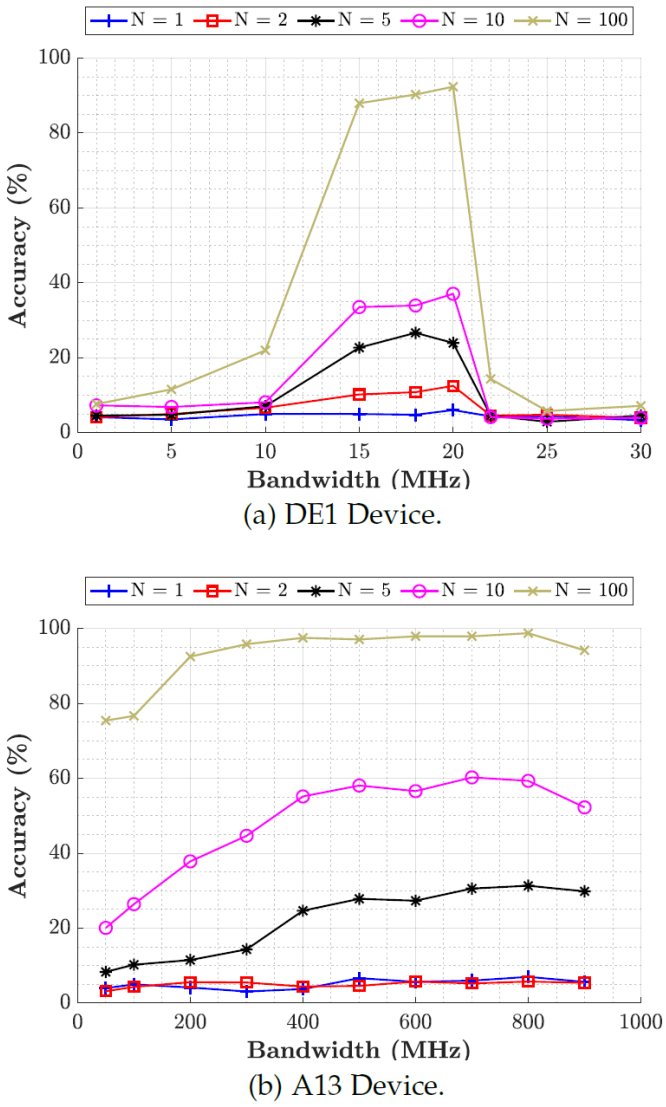


Figure 81 Impact of N value on accuracy for permutations of instructions from the same instruction type.

5.0 CONCLUSIONS

In this report, we have described a system called CAMELIA that monitors computation in an EMSD (or phone/laptop/server) device by leveraging the involuntary electromagnetic (EM) emanations from the monitored device. CAMELIA does not require changes to the monitored device or its software, and its monitoring ability remains intact even after a complete compromise of the monitored system. CAMELIA collects signals using purpose-designed antennas, then pre-processes the signals and separates them into sub-channels that carry information about different aspects of the system's state. CAMELIA uses models of valid software behavior and software/system/hardware interactions to form hypothesis about the sequence of execution and software/system/hardware events in the monitored system, then updates these hypotheses by matching the expected signals for each hypothesis to the observed signals. This allows CAMELIA to maintain high accuracy and fidelity even when monitoring large codes, and even in the presence of interrupts, input/output activity, cache misses, branch miss-predictions, and other events that change emanated EM signals significantly in a way that is seemingly random but that CAMELIA can account for end even use to improve monitoring.

To manage the tradeoff between fidelity, computational cost of modeling, and timeliness of reporting, CAMELIA operates at three levels of fidelity. More precisely, CAMELIA can (1) discover loop/module-level anomalies immediately, (2) detect basic-block-level control flow violations and anomalies at the granularity of several instructions very rapidly (after one or few dynamic instances of the violation are observed) and (3) uncover anomalous execution/event

patterns and even “below noise level” problems (e.g., when a valid instruction is replaced by a similar instructions) after enough dynamic instances are observed.

The system is divided into six tasks as shown in Figure 1. In Task 1.1, a high gain planar slotted circular disc antenna, designed for receiving EM emanations modulated around processor clock was presented. The antenna was designed around 1 GHz for a 70 MHz bandwidth, using higher order mode $TM12$ mode, which had a larger electrical size than the fundamental mode. This was done to reduce the number of elements. The antenna was designed in stacked configuration which permits the use of EM coupling as an excitation and hence feed lines were avoided. The antenna was fabricated using aluminum circular slotted discs, which are suspended in air using Teflon screws. It was shown that the electric field null property of $TM12$ mode allows the use of screws to suspend the discs above the ground plane. The signal detection at the distances greater than 3 m were demonstrated by direct LoS SNR measurements from an IoT board. For each distance, SNR was calculated by subtracting the detectable signal power, when board activity is on, with the noise power when there is no activity. Finally, the antenna was used to collect EM signals from an IoT board while being >3m away from the board. The results show that using this antenna, an IoT board can be monitored from >3m with excellent accuracy. Furthermore, the antenna is cost effective and can be treated as a sub array for larger array for going further distances in EM emanations measurements.

In Task 1.2 An algorithm for finding carriers of frequency-modulated (FM) and amplitude-modulated (AM) electromagnetic (EM) emanations from computer systems was described. Computer systems create EM emanations across the RF spectrum making it difficult, error-prone, and time-consuming to find the relatively few emanations that expose sensitive information. One of the most common and simplest mechanisms for information leakage occurs when the amplitude or a frequency of an existing strong signal (e.g. a processor or memory clock) is amplitude or frequency modulated by a system activity. If the system activity can be linked to sensitive information, this results in information leakage. We have presented an algorithm for automatically finding these AM and FM modulated signals, demonstrated the algorithm's performance on several different types of processors and systems (desktop, laptop, and smart phone), and compared the results to an exhaustive manual search. We have also verified that all signals identified by the algorithm can be traced to plausible unintentional modulation mechanisms to illustrate that these signals can potentially cause information leakage. This algorithm is an important tool for system designers to quickly identify circuits that are leaking sensitive information.

In Task 2.1, we have proposed (REMOTE), a new robust framework to detect malware by externally observing EM signals emitted by an electronic device. REMOTE does not require any resources or infrastructure on, or any modifications to, the monitored system itself, which makes it especially suitable for malware detection on embedded and/or cyber-physical systems where hardware resources may be limited and performance and energy overheads introduced by other

monitoring approaches may be unacceptable. REMOTE can identify malicious code injection into a known application that is running on a device in real-time and with a low detection latency.

To develop a robust framework, we systematically explored practical concerns through experiments and analysis. First, to demonstrate the usability of REMOTE in real-world scenarios, we ported several real-world cyber-physical- systems each with a meaningful attack, to different platforms. Our results showed that for all of the programs on each of the platforms, REMOTE successfully detected the instances of attacks with high accuracy and almost no false positives. We then systematically evaluated the robustness of REMOTE to interrupts and other system activity, to signal variation among different physical instances of the same device, to changes in antenna distance, and to changes over time. By selectively disabling the robustness-oriented features of REMOTE, we also demonstrated that these features are indeed contributing to its robustness.

Using these measurements and analysis, we showed REMOTE has several advantages over state-of-the-art external malware detection frameworks and it is a promising candidate for protecting resource-constrained devices (e.g., CPS, IoT, PLC, etc.) when implementing an internal malware detector is infeasible.

In Task 2.2, we have proposed a new scheme that combines CNN and Markov models to monitor multi-core systems to detect anomalies and malware. In the proposed methodology, the Markov Model describes the dependencies among hot paths of a given program whereas the neural network estimates the likelihood of Markov states at a specific instance. The structure is designed

such that the training phase of the neural net does not consider the dependency among its inputs. However, in the testing phase, both the CNN and the Markov models are combined to check whether the transition between states are allowed. Here, states are considered as the hot spots of the program and transitions are allowed only if the program has such a path. The inputs to the neural network are the emanated EM signals that are unintentionally generated while executing a program. Since there is no instrumentation planted to the monitored system, the proposed method introduces no overhead, therefore, it is observer-effect-free.

We tested the proposed methodology on various devices with different numbers of active cores. We obtained no false negative, in other words, the method always alerts users whenever an anomaly exists in the system. Compared to other zero-overhead profiling methods, the proposed framework can identify which program has anomalies while multi-cores are active. Therefore, the proposed methodology can be used to secure systems or to monitor anomalies even on multi-core devices since the deviations from the actual behavior of a program can be exposed with the proposed scheme.

Tasks 3-5 describe TESLA – a new approach for program execution tracing via EM side-channel signals. TESLA is completely non-invasive and does not impose any overhead on the monitored system. TESLA is especially useful for monitoring resource-constrained embedded devices for tasks such as program debugging and anomalous/malicious program activity detection.

Experimental evaluations reveal that TESLA can provide highly accurate program traces for benchmark applications running on different embedded devices, even from 1 m away.

Finally, Task 6 describes PITEM, a new approach for instruction-level tracking using EM side channel. PITEM is a tool that can be used for fine-grain malware detection with an ability to locate the malware injection precisely. It first identifies groups of instructions, instruction types that have similar EM signatures using hierarchical clustering. After identifying instruction types, during training phase, it generates templates for all possible permutations of these instruction types. In the testing phase, we obtain testing traces and predict the best matching template with a correlation based, matched-filter-like predictor. The proposed methodology is tested using two different devices, one FPGA-based processor and one ARM-based IoT device. The results are reported for different repetitions of the permutation blocks. For single execution of the permutation block, we obtain 92.8% and 62% accuracies for the two different testing devices. We observe that with only two repetitions of the permutation blocks, these accuracies significantly increase to 100% and 97.8%. Then, we evaluate the performance of the detection system under AWGN. We note that the performance of the system is stable for > 15 dB SNR and the performance gradually decreases for lower values of SNR. We also note that repeated permutation blocks are more resilient to AWGN. Finally, we perform detection of the permutations from the same instruction type. Although the detection accuracy is low for single execution of the instructions within the block, the accuracy increases significantly when the instructions are repeated.

6.0 REFERENCES

[1] Prateek Juyal, Sinan Adibelli, Alenka Zajic "A Directive Antenna Based on Conducting Disc for Detecting Unintentional EM Emissions at Larger Distances," IEEE Transactions on Antennas and Propagation, vol.66, pp. 6751-6761, December 2018..

[2] Jelena Dinkić, Dragan Olcan, Antonije Djordjević, and Alenka G. Zajic," High-Gain Quad Array of Nonuniform Helical Antennas," International Journal of Antennas and Propagation, vol. 2019, Article ID 8421809, 12 pages, 2019..

[3] H. Legay and L. Shafai, "New stacked microstrip antenna with large bandwidth and high gain," Inst. Elect. Eng. Proc. Microw. Antennas Propagation, vol. 141, no. 3, pp. 199–204, Jun. 1994.

[4] P. Juyal, L. Shafai, "Sidelobe Reduction of TM12 Mode of Circular Patch via Non-resonant Narrow Slot", IEEE Trans. Antennas Propagation, vol. 64, pp. 3361-3369, 2016.

[5] A. Vosoogh and P.-S. Kildal, "Simple formula for aperture efficiency reduction due to grating lobes in planar phased arrays," IEEE Trans. Antennas Propagation, vol. 64, no. 6, pp. 2263–2269, Jun. 2016.

[6] K. C. Kerby and J. T. Bernhard, "Sidelobe level and wideband behavior of arrays of random subarrays," IEEE Trans. Antennas Propagation, vol. 54, no. 8, pp. 2253–2262, Aug. 2006.

[7] S. A. Razavi et al., "2x2-Slot Element for 60-GHz Planar Array Antenna Realized on Two Doubled-Sided PCBs Using SIW Cavity and EBG-Type Soft Surface fed by Microstrip-Ridge Gap Waveguide," in IEEE Transactions on Antennas and Propagation, Sept. 2014.

[8] M. Prvulovic, A. Zajić, R. Callan, and C. Wang, "A method for finding frequency-modulated and amplitude-modulated electromagnetic emanations in computer systems," IEEE Transactions on Electromagnetic Compatibility, vol. 59, no 1, pp. 34-42, 2017.

[9] R. Callan, A. Zajic, M. Prvulovic, "A Practical Methodology for Measuring the Side-Channel Signal Available to the Attacker for Instruction-Level Events", Microarchitecture (MICRO) 2014 , pp. 242-254, 2014.

[10] A. Nazari, N. Sehatbakhsh, M. Alam, A. Zajic, and M. Prvulovic, “Eddie: Em-based detection of deviations in program execution,” in Proceedings of the 44th Annual International Symposium on Computer Architecture, 2017, pp. 333–346.

[11] N. Sehatbakhsh, A. Nazari, A. Zajic, and M. Prvulovic, “Spectral profiling: Observer-effect-free profiling by monitoring em emanations,” in Microarchitecture, 2016 49th Annual IEEE/ACM International Symposium on. IEEE, 2016, pp. 1–11.

[12] Nader Sehatbakhsh, Alireza Nazari, Monjur Alam, Frank Werner, Yuanda Zhu, Alenka Zajic, and Milos Prvulovic, “REMOTE: Robust External Malware Detection Framework by Using Electromagnetic Signals,” in IEEE Transactions on Comp., 69, no. 3, pp. 312-326,2020.

[13] B. Yilmaz, F. Werner, S. Park, E. Ugurlu, E.Jorgensen, M. Prvulovic, A. Zajic, “MarCNNet: a markovian convolutional neural network for malware detection and monitoring multi-core systems,” submitted to IEEE Trans. on Information Forensics and Security, 20.

[14] N. Sehatbakhsh, et. al, “A new sidechannel vulnerability on modern computers by exploiting electromagnetic emanations from the power management unit,” IEEE International Symposium on High-Performance Computer Architecture, 2020.

[15] B. B. Yilmaz, A. Zajic, and M. Prvulovic, “Modelling jitter in wireless channel created by processor-memory activity,” in 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), April 2018, pp. 2037–2041..

[16] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, “Mibench: A free, commercially representative embedded benchmark suite,” in Proc. of the Fourth Annual IEEE International Workshop on Workload Characterization.IEEE, 2001,.

[17] B. B. Yilmaz, N. Sehatbakhsh, M. Prvulovic, and A. Zajic, “Communication model and capacity limits of covert channels created by software activities,” IEEE Trans. on Information Forensics and Security, vol. 15, pp. 776–789, 2019..

[18] R. Callan, F. Behrang, A. Zajic, M. Prvulovic, and A. Orso, “Zerooverhead profiling via em emanations,” in Proceedings of the 25th International Symposium on Software Testing and Analysis. 2016,pp. 401–412.

[19] B. B. Yilmaz, E. M. Ugurlu, A. Zajic, and M. Prvulovic, “Instruction level program tracking using electromagnetic emanations,” in SPIE, 2019.

[20] Haider Khan, Sunjae Park, Alenka Zajic, and Milos Prvulovic, "TESLA: program Tracing through Electromagnetic Side-channeL Analysis" submitted IEEE Transactions on Computers, 2020.

[21] B. B. Yilmaz, A. Zajic, and M. Prvulovic, “Modelling jitter in wireless channel created by processor-memory activity,” in 2018 IEEE International Conference on Acoustics, Speech and Signal Processing IEEE, 2018, pp. 2037–2041..

[22] A. Zajic and M. Prvulovic, “Experimental demonstration of electromagnetic information leakage from modern processor-memory systems,” IEEE Transactions on Electromagnetic Compatibility, vol. 56, no. 4, pp. 885–893, 2014..

[23] M. Dey, A. Nazari, A. Zajic, and M. Prvulovic, “Emprof: Memory profiling via em-emanation in iot and hand-held devices,” in 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture IEEE, 2018, pp. 881–893..

[24] Elvan Mert Ugurlu, Baki Berkay Yilmaz, Alenka Zajic, and Milos Prvulovic, "PITEM: Permutations-based instruction tracking via electromagnetic side-channel signal analysis" submitted to IEEE Trans. on Computers, 2020 .

[25] May 2019. [Online]. Available: https://www.intel.com/content/www/us/en/programmable/solutions/partners/partner-profile/terasic-inc-/board/altera-de1-board.html.

[26] OlinuXino A13, https://www.olimex.com/Products/OLinuXino/A13/A13-OLinuXino/open-source-hardware..

[27] M. Omran, A. Engelbrecht, and A. Salman, “An overview of clustering methods,” Intell. Data Anal., vol. 11, pp. 583–605, 11 2007..

[28] Yee Leung, Jiang-She Zhang, and Zong-Ben Xu, “Clustering by scale-space filtering,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, no. 12, pp. 1396–1410, 2000..

[29] S. van Dongen and A. J. Enright, “Metric distances derived from cosine similarity and pearson and spearman correlations,” 2012..

[30] G. Mayhew-Ridgers et al., "Accuracy of the gain-transfer method for a standard gain antenna and a test antenna with equal aperture dimensions," C COMSIG '98. Proceedings of the 1998 South African Symposium on, Rondebosch, 1998.

[31] “Lewansoul learm 6dof full metal robotic arm,” retrieved on April 2019 from https://www.amazon.com/LewanSoul-Controller-Wireless-Software-Tutorials/dp/B074T6DPKX..

[32] H. Shacham, “The geometry of innocent flesh on the bone: Return into-libc without function calls (on the x86),” in Proceedings of the 14th ACM Conference on Computer and Communications Security,ser. CCS ’07. New York, NY, USA: ACM, 2007, pp. 552–561..

[33] T. Bletsch, X. Jiang, V. W. Freeh, and Z. Liang, “Jump oriented programming: A new class of code-reuse attack,”in Proceedings of the 6th ACM Symposium on Information,Computer and Communications Security, ser. ASIACCS ’11..

[34] B. Wijnen, E. J. Hunt, G. C. Anzalone, and J. M. Pearce,“Open-source syringe pump library,” PLOS ONE, vol. 9, no. 9,pp. 1–8, 09 2014..

[35] T. Abera, N. Asokan, L. Davi, J.-E. Ekberg, T. Nyman, A. Paverd,A.-R. Sadeghi, and G. Tsudik, “C-flat: Control-flow attestation for embedded systems software,” in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security,.

[36] J. Salwan and A. Wirth, “Ropgadget: Gadgets finder for multiple architectures,” https://github.com/JonathanSalwan/ROPgadget, 2011 (accessed Feb. 1, 2018)..

[37] “Horn antenna datasheet,” https://www.com-power.com/ah118hornantenna.html, 2015 (accessed Nov. 5, 2017)..

[38] S. A. Carr and M. Payer, “Datashield: Configurable data confidentiality and integrity,” in Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, ser. ASIA CCS ’17. New York, NY, USA: ACM, 2017, pp. 193–204..

[39] D. Quarta, M. Pogliani, M. Polino, F. Maggi, A. M. Zanchettin, and S. Zanero, “An experimental security analysis of an industrial robot controller,” in 2017 IEEE Symposium on Security and Privacy (SP), May 2017, pp. 268–286..

[40] “Lewansoul learm 6dof full metal robotic arm,” retrieved on April 2019 from https://www.amazon.com/LewanSoul-Controller-Wireless-Software-Tutorials/dp/B074T6DPKX..

[41] “Arduino servo library,” https://www.arduino.cc/en/Reference/Servo, accessed April 2019)..

[42] Ubuntu, “Lightdm,” https://wiki.ubuntu.com/LightDM, 2017 (accessed Feb. 1, 2018).

[43] Keysight Signal Analyzer, https://www.keysight.com/en/pdx-x202266-pn-N9020A/mxa-signal-analyzer-10-hz-to-265-ghz?pm=spc&nid=-32508.1150426&cc=US&lc=eng..

[44] G. Navarro, “A guided tour to approximate string matching,” ACM computing surveys (CSUR), vol. 33, no. 1, pp. 31–88, 2001..

[45] G. Rothermel, S. Elbaum, A. Kinneer, and H. Do, “Software-artifact infrastructure repository,” UR L http://sir. unl. edu/portal, 2006..

[46] (2019). Trusthub. [Online]. Available: http://www.trusthub.org/benchmarks/trojan.

APPENDIX A – PUBLICATIONS

Journal Papers

[1] B. Yilmaz, F. Werner, S. Park, E. Ugurlu, E.Jorgensen, M. Prvulovic, A. Zajic, "MarCNNet: a markovian convolutional neural network for malware detection and monitoring multi-core systems," submitted to *IEEE Trans. on Information Forensics and Security*, 2020.

[2] B. Yilmaz, N. Sehatbakhsh, M. Dey, C.-L. Cheng, M. Prvulovic, and A. Zajic, "A generalized approach to estimation of covert channel information leakage capacity," submitted to *IEEE Trans. on Information Forensics and Security*, 2020, under revision.

[3] F. T. Werner, J. Dinkić, D. Olćan, A. Djordjević, M. Prvulovic, and A. Zajić, "An efficient method for localization of magnetic field sources that produce high-frequency side-channel emanations," submitted to *IEEE Transactions on Electromagnetic Compatibility*, 2020, under revision.

[4] Elvan Mert Ugurlu, Baki Berkay Yilmaz, Alenka Zajic, and Milos Prvulovic, "PITEM: Permutations-based instruction tracking via electromagnetic side-channel signal analysis" submitted to *IEEE Transactions. on Computers*, 2020, under revision.

[5] Haider Khan, Sunjae Park, Alenka Zajic, and Milos Prvulovic, "TESLA: program Tracing through Electromagnetic Side-channeL Analysis" submitted *IEEE Transactions on Computers*, 2020, under revision.

[6] F. T. Werner, B. B. Yilmaz, M. Prvulovic, and A. Zajić, "Leveraging EM side-channels for recognizing components on a motherboard," to appear in *IEEE Transactions on Electromagnetic Compatibility*, 2020.

[7] S. Sangodoyin, F. Werner, B. B. Yilmaz, C-L Cheng, E. M. Ugurlu,, N. Sehatbakhsh, M. Prvulovic, and A. Zajic, "Side-channel propagation measurements and modeling for hardware security in IoT devices," to appear in *IEEE Transactions on Antennas and Propagation*, 2020.

[8] L. Nguyen, C. Cheng, F. Werner, M. Prvulovic, and A. Zajic, "A comparison of backscattering, EM, and power side-channels and their performance in detecting software and hardware intrusions," in *Journal of Hardware and System Security*, March 2020.

[9] B. Yilmaz, N. Sehatbakhsh, A. Zajić and M. Prvulovic, "Communication model and capacity limits of covert channels created by software activities," in *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1891-1904, 2020.

[10] B. Yilmaz, M. Prvulovic, and A. Zajic, "Electromagnetic side-channel information leakage created by execution of series of instructions in a computer processor in *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 776-789, 2020.

[11] Nader Sehatbakhsh, Alireza Nazari, Monjur Alam, Frank Werner, Yuanda Zhu, Alenka Zajic, and Milos Prvulovic, "REMOTE: Robust External Malware Detection Framework by Using Electromagnetic Signals," in *IEEE Transactions on Computers*, 69, no. 3, pp. 312-326, 1 March 2020.* – chosen for the Featured Paper

[12] H. Khan, N. Sehatbakhsh, L. Nguyen, Milos Prvulovic, and Alenka Zajic, "Malware detection in embedded systems using neural network model for electromagnetic side-channel signals," *Journal of Hardware and System Security*, 305–318, August 2019.

[13] Haider A. Khan, Nader Sehatbakhsh, Luong N. Nguyen, Robert Callan, Arie Yeredor, Milos Prvulovic, and Alenka Zajic, "IDEA: intrusion detection through electromagnetic-signal analysis for critical embedded and cyber-physical systems," in *IEEE Transactions on Dependable and Secure Computing*, 2019.

[14] J. Dinkić, D. Olcan, A. Djordjević, and A. Zajic," Design and Optimization of Nonuniform helical antennas with linearly varying geometrical parameters," *IEEE Access* 7, pp. 855-866, 2019.

[15] Sinan Adibelli, Prateek Juyal, Chia-Lin Cheng, and Alenka Zajic, "THz near field focusing using a 3D printed Cassegrain configuration for backscattered side-channel detection," *IEEE Transactions on Antennas and Propagation*, vol. 67, no. 10, pp. 6627-6638, October 2019.

[16] M. Ruble, C. E. Hayes, M. Welborn, A. Zajić, M. Prvulovic and A. M. Pitruzzello, "Hyperdimensional Bayesian Time Mapping (HyperBaT): A Probabilistic Approach to Time Series Mapping of Non-Identical Sequences," in *IEEE Transactions on Signal Processing*, vol. 67, no. 14, pp. 3719-3731, July, 2019.

[17] Jelena Dinkić, Dragan Olcan, Antonije Djordjević, and Alenka G. Zajic," High-Gain Quad Array of Nonuniform Helical Antennas," *International Journal of Antennas and Propagation,* vol. 2019, Article ID 8421809, 12 pages, 2019.

[18] Prateek Juyal, Sinan Adibelli, Alenka Zajic "A Directive Antenna Based on Conducting Disc for Detecting Unintentional EM Emissions at Larger Distances," *IEEE Transactions on Antennas and Propagation*, vol.66, pp. 6751-6761, December 2018.

[19] Baki Berkay Yilmaz, Robert Callan, Milos Prvulovic, and Alenka Zajic, "Capacity of the EM Covert/Side-Channel Created by the Execution of Instructions in a Processor," *IEEE Transactions on Information Forensics and Security*, vol. 13, pp. 605-620, June 2018.

[20] Frank Werner, Derrick Chu, Antonije R. Djordjevic, Dragan I. Olcan, Milos Prvulovic, and Alenka Zajic, "A Method for Efficient Localization of Magnetic-field Sources Excited by the Execution of Instructions in a Processor," *IEEE Transactions on Electromagnetic Compatibility*, vol. 60, pp. 613-622, June 2018.

[21] M. Prvulovic, A. Zajić, R. Callan, and C. Wang, "A method for finding frequency-modulated and amplitude-modulated electromagnetic emanations in computer systems," *IEEE Transactions on Electromagnetic Compatibility*, vol. 59, no 1, pp. 34-42, 2017.

Conference Papers

[1] J. Dinkic, D. Olcan, A. Djordjevic, A. Zajic, "Comparison of the optimal uniform and nonuniform lossy helical antennas," IEEE Proceedings of AP-S/URSI, pp. 1-2, July 2020, Montreal, Canada.

[2] B. Yilmaz, E. Ugurlu, A. Zajic, and M. Prvulovic, "Cell-phone classification: a convolutional neural network approach exploiting electromagnetic emanations," in Proceedings of ICASSP, pp. 1-5, May 2020, Barcelona, Spain.

[3] B. Yilmaz, E. Ugurlu, F. Werner, A. Zajic, and M. Prvulovic, "Program profiling based on Markov models and EM emanations," in Proceedings of SPIE, April 2020, Anaheim, CA.

[4] L. Nguyen, B. Yilmaz, C. Cheng, M. Prvulovic, and A. Zajic, "A novel clustering technique using backscattering side-channel for counterfeit IC detection," in Proceedings of SPIE, April 2020, Anaheim, CA.

[5] Sangodoyin, F. Werner, B. B. Yilmaz, C. Cheng, E. M. Ugurlu, N. Sehatbakhsh, M. Prvulovic, and A. Zajic, "Remote monitoring and propagation modeling of EM side-channel signals for IoT device security," in Proceedings of 14th European Conference on Antennas and Propagation (EuCAP), pp. 1-5., March 2020, Copenhagen, Denmark.

[6] Nader Sehatbakhsh, Baki Yilmaz, Alenka Zajic, and Milos Prvulovic, "A New Side-Channel Vulnerability on Modern Computers by Exploiting Electromagnetic Emanations from the Power Management Unit," in Proceedings of the 26th IEEE International Symposium on High-Performance Computer Architecture, 2020.

[7] Nader Sehatbakhsh, Baki Yilmaz, Alenka Zajic, Milos Prvulovic, "EMSim: A Microarchitecture-Level Simulation Tool for Modeling Electromagnetic Side-Channel Signals," in Proceedings of the 26th IEEE International Symposium on High-Performance Computer Architecture, 2020. **– nominated for the best paper award**

[8] Baki Yilmaz, Elvan Ugurlu, Alenka Zajic, and Milos Prvulovic, "Detecting Cellphone Camera Status at Distance by Exploiting Electromagnetic Emanations," in Proceedings of IEEE MILCOM, November 2019, pp. 1-6, Norfolk, VA.

[9] Baki Yilmaz, Alenka Zajic, and Milos Prvulovic, "Capacity of EM Side Channel Created by Instruction Executions in a Processor," in Proceedings of IEEE IEMCON, October 2019, pp. 1-5, Vancouver, CA.

[10] Nader Sehatbakhsh, Alireza Nazari, Haider Khan, Alenka Zajic, and Milos Prvulovic, "EMMA: Hardware/Software Attestation Framework for Embedded Systems Using Electromagnetic Signals," in Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, 2019, pp.1-11, 12-15 October 2019, Columbus, OH.

[11] Frank Werner, Antonije Djordjevic, and Alenka Zajic, "A compact probe for EM side-channel attacks on cryptographic systems," in Proceedings of IEEE International Symposium on Antennas and Propagation, pp. 1-2, July 2019, Atlanta, GA.

[12] Richard Rutledge, Sunjae Park, Haider Khan, Alessandro Orso, Milos Prvulovic, and Alenka Zajic, "Zero-overhead path prediction with progressive symbolic execution,"

In Proceedings of the IEEE 41st International Conference on Software Engineering (ICSE ’19), pp. 234-245, May 2019, Montreal CA.

[13] Baki Yilmaz, Elvan Ugurlu, Alenka Zajic, Milos Prvulovic, “Instruction level program tracking using electromagnetic emanations,” Proceedings of SPIE, pp.1-6, April 2019, Baltimore, MD.

[14] Baki Yilmaz, Alenka Zajic, and Milos Prvulovic, “Capacity of deliberate side-channels created by software activities,” Proceedings of IEEE MILCOM, October 2018, pp. 1-6, Los Angeles, CA.

[15] M. Dey, A. Nazari, A. Zajic and M. Prvulovic, “EMPROF: Memory Profiling Via EM-Emanation in IoT and Hand-Held Devices,” 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Fukuoka, Japan, 2018, pp. 881-893.

[16] M. Alam, H. Khan, M. Day, R. Callan, N. Sinha, A. Zajic, and M. Prvulovic, “One & done – A Single-Decryption EM-Based Attack on OpenSSL’s Constant-Time Blinded RSA ,” USENIX Security, August 2018.

[17] J. Dinkić, D. Olćan, A. Zajić, A. Djordjević, “Comparison of optimization approaches for designing nonuniform helical antennas,” Proceedings of 2018 IEEE AP-S Symposium on Antennas and Propagation and URSI CNC/USNC, Boston, USA, July 8-13, 2018, pp. 1581-1582.

[18] F. Werner, A. R. Djordjevic, D. I. Olcan, M. Prvulovic, and A. Zajic, “Experimental validation of localization method for finding magnetic sources on IoT devices,” *IEEE Proceedings of EMC Europe*, pp. 1-5, Amsterdam, Netherlands, August 2018.

[19] N. Sehatbakhsh, A. Nazari, M. Alam, A. Zajić, and M. Prvulovic, "Syndrome: spectral analysis for anomaly detection on medical IoT and embedded devices," IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 1-8, May 2018, Washington DC.

[20] N. Sehatbakhsh, H. Hong, B. Lazar, B. Johnson-Smith, O. Yilmaz, M. Alam, A. Nazari, A. Zajic, and M. Prvulovic" Spectral Analysis for Anomaly Detection on Medical IoT and Embedded Devices- Experimental Demonstration," IEEE International Symposium on Hardware Oriented Security and Trust (HOST) Hardware Demo, pp.1, May 2018, Washington DC.

[21] H. Khan, M. Alam, A. Zajic, and M. Prvulovic, "Detailed tracking of program control flow using analog side-channel signals: A promise for IoT malware detection and a threat for many cryptographic implementations," Proceedings of SPIE, April 2018, Orlando FL.

[22] B. Yilmaz, M. Prvulovic, and A. Zajic, "Wireless communication channel created by processor memory activity," IEEE Proceedings of ICASSP, pp. 1-5, April 2018, Calgary, Canada.

[23] S. Adibelli, R. Golubović, A. Djordjević, D. Olćan, and A. Zajić, "Design and fabrication of non-uniform helical antennas for detection of side-channel attacks in computer systems," IEEE Proceedings of 12th European Conference on Antennas and Propagation (EuCAP), pp. 1-5, April 2018, London, UK.

[24] Baki Berkay Yilmaz, Robert Callan, Milos Prvulovic, and Alenka Zajic, "Quantifying Information Leakage in a Processor Caused by the Execution of Instructions," Proceedings of IEEE MILCOM, October 2017

[25] A. Nazari, N. Sehatbakhsh, M. Alam, A. Zajic, and M. Prvulovic, "EDDIE: EM-Based Detection of Deviations in Program Execution," Proceedings of the 44th International Symposium on Computer Architecture (ISCA), June2017. (acceptance rate 17 %)

[26] N. Sehatbakshsh, R. Callan, M. Alam, M. Prvulovic, and A. Zajic, "Leveraging Electromagnetic Emanations for IoT Security, "Hardware Demo at IEEE International Symposium on Hardware Oriented Security and Trust (HOST) May 1-5, 2017. –**Best Demo Award**

[27] A. Zajic, Milos Prvulovic, and Derrick Chu, "Path Loss Prediction for Electromagnetic Side-Channel Signals," Proceedings of the 11th European Conference on Antennas and Propagation EUCap11, pp.1-5, Paris, France, March 2017.

[28] N. Sehatbakhsh, A. Nazari, A. Zajić, and Milos Prvulovic, "Spectral Profiling: Observer-Effect-Free Profiling by Monitoring EM Emanations," IEEE MICRO 16, pp.1-11, Taipei, Taiwan, October 2016. (acceptance rate 20 %) – **The Best Paper Award**

[29] R. Callan, F. Behrang, M. Prvulovic, A. Zajic, and A. Orso, "Zero-Overhead Profiling via EM Emanations," accepted to The International Symposium on Software Testing and Analysis, 18-20 July 2016, Saarbrücken, Germany. (acceptance rate 25 %)

APPENDIX B –ABSTRACTS

[1] B. Yilmaz, F. Werner, S. Park, E. Ugurlu, E. Jorgensen, M. Prvulovic, A. Zajic, "MarCNNet: a markovian convolutional neural network for malware detection and monitoring multi-core systems," submitted to *IEEE Trans. on Information Forensics and Security*, 2020.

Abstract: Leveraging side-channels enables zero-overhead detection of anomalies. These channels offer a non-instrumented program profiling capability by processing unintentional signals emitted while executing programs, codes, etc. In this paper, we propose a Markov based monitoring for multi-core devices utilizing the features extracted by a convolutional neural network (CNN). We refer to the proposed framework as MarCNNet. The input of the overall model is the magnitude-averaged short-time- Fourier-transform (STFT) of the emanated electromagnatic (EM) signals. To reduce the dimension of the signal fed to the model, the states of the Markov Model are considered as the hot paths. Transitions between states are only possible if the program can follow the path. In the framework, the CNN model generates features which are utilized by the Markov Model to estimate the likelihood of the state transitions. If the estimated transitions do not comply with the Markov Model, it alerts anomaly, otherwise, it keeps monitoring in real time. The framework also simplifies the training process because dependency among states is only crucial for the Markov Model, but not for the CNN. Therefore, the neural network is trained

assuming the signals generated by each hot paths are independent. However, for a test signal, both the CNN and the Markov Models are considered for malware detection. We tested the proposed model for various devices with different number of cores and programs, and demonstrated that the framework can detect malware with no false negatives, and a false positive rate less than 2%.

[2] B. Yilmaz, N. Sehatbakhsh, M. Dey, C.-L. Cheng, M. Prvulovic, and A. Zajic, "A generalized approach to estimation of covert channel information leakage capacity," submitted to *IEEE Trans. on Information Forensics and Security*, 2020, under revision.

Abstract: Foreseeing severity of leakages through covert channels is a necessity for designers to minimize information leakage. Covert channels can be created due to digital and/or analog characteristics of computer's switching activities. Hence, a judicious approach has to be followed to make these systems more resilient to any covert channel attacks. Having a method to estimate the capacity of information leakage in design-state provides an opportunity for designers to adjust their systems to minimize leakages of worst-case scenarios. In this paper, we propose a methodology to estimate the worst-case information leakage through various covert channels which can be adopted for both analog and digital covert channels. In that respect, we first model the communication channel as a deletion-insertion channel to mimic the possible losses due to software activities. Unlike conventional communication systems where the noise is assumed to be Additive White Gaussian Noise (AWGN), covert channels also suffer from changes in signaling time of transmitted bits. We show that the noise caused by signaling time variation can be combined with AWGN to explain the overall effective noise on the covert channel communication system. Secondly, based on the effective noise, we model the communication channel between the receiver and the transmitter. Then, we define the channel capacity as the maximum leakage for a

given covert channel. Finally, we provide experimental results to show that the proposed model is an effective and a general method to attain the resilience of a given system.

[3] F. T. Werner, J. Dinkić, D. Olćan, A. Djordjević, M. Prvulovic, and A. Zajić, "An efficient method for localization of magnetic field sources that produce high-frequency side-channel emanations," submitted to *IEEE Transactions on Electromagnetic Compatibility*, 2020, under revision.

Abstract: A new, low-cost system for locating sources of high frequency EM side-channel emanations on a printed-circuit board (PCB) is presented. The challenges inherent in high frequency measurements are addressed through careful design of the measurement and localization system. The system is time efficient, requiring only measurements taken around the edge of the device. The accuracy of the measurement setup was verified by comparing measurements with simulated results. The setup was then used to locate the instruction-dependent sources at 1 GHz on an FPGA and an IoT development board. The 1 GHz sources are compared to previously identified sources on the same devices taken at significantly lower frequencies. The results demonstrate that the sources of the EM side-channel can vary not only with the executed instruction but also with the frequency at which the side-channels are observed.

[4] Elvan Mert Ugurlu, Baki Berkay Yilmaz, Alenka Zajic, and Milos Prvulovic, "PITEM: Permutations-based instruction tracking via electromagnetic side-channel signal analysis" submitted to *IEEE Transactions. on Computers*, 2020, under revision.

Abstract: The emergence of cyber-physical systems (CPS) and internet of things (IoT) devices impose significant security and privacy concerns that necessitate robust monitoring and malware detection systems for protection. This paper proposes PITEM, a framework for instruction-level monitoring and malware detection using electromagnetic (EM) side-channels. PITEM identifies instruction types with similar EM emanations using hierarchical clustering. To track all combinations of these instruction types, we generate EM signatures for all permutations of them. In testing, we predict the permutation class of testing traces by a matched-filter-like predictor. We test the performance on two devices (FPGA-based and ARM-based) with 50 MHz and 1 GHz clock frequencies. We achieve 92.8% and 62% accuracies for these devices for single execution of permutations. We note that the accuracy increases to as high as 100% when permutation blocks are repeated. Furthermore, we test the limits of the system by tracking permutations of instructions of the same type. The results show that with sufficient bandwidth and number of repetitions, individual instructions can be resolved with 92.4% and 98% accuracies for these devices. Finally, the performance is evaluated for different signal-to-noise ratio (SNR) levels and performance is found to be stable for SNR values higher than 15 dB.

[5] Haider Khan, Sunjae Park, Alenka Zajic, and Milos Prvulovic, "TESLA: program Tracing through Electromagnetic Side-channeL Analysis" submitted *IEEE Transactions on Computers*, 2020, under revision.

Abstract: We present TESLA, a novel framework for zerooverhead program execution tracing. TESLA leverages device's electromagnetic (EM) side-channel signals for basic-blockgranularity execution tracing. TESLA is completely non-invasive and does not require any resource or any modification of the monitored device. Thus, this approach is especially suitable for monitoring resource-constrained devices such as embedded devices and Internet of Things (IoT) devices. In the training phase, TESLA learns a signal emanation model that associates code segments or program subpaths with corresponding signal snippets. In the testing phase, TESLA uses signal matching to establish a correspondence between the test signal and the training signal, and then exploits the learned signal emanation model to reconstruct the program execution path. We evaluate TESLA by monitoring benchmark applications on different embedded devices. TESLA achieves 99% path reconstruction accuracy for monitoring an FPGA device (Altera DE1). We further evaluate TESLA by monitoring an IoT device (A13- OLinuXino board with 1 GHz processor and Linux operating system), for which TESLA achieves roughly 95% accuracy. Furthermore, experimental evaluations reveal that TESLA can monitor these devices from 1 m distance.

[6] F. T. Werner, B. B. Yilmaz, M. Prvulovic, and A. Zajić, "Leveraging EM side-channels for recognizing components on a motherboard," to appear in *IEEE Transactions on Electromagnetic Compatibility*, 2020.

Abstract: This paper proposes leveraging EM side-channels to recognize/authenticate electronic components integrated onto a motherboard. By focusing on components on a motherboard, our method provides an opportunity to authenticate devices assembled by third parties. This method identifies components based on the modulated signals emanated while they are excited in a controlled manner. When testing an unknown component, the spectrum is compared to previously recorded training signatures. To improve efficiency, the size of the spectrum is reduced by projecting it into a vector space generated from training signatures. The identity of the tested component is then determined using a k-Nearest Neighbors algorithm. This method successfully classified memory, processor, and Ethernet transceiver components integrated on seven types of Internet-of-Things devices. Since manufacturers commonly use the same components in multiple designs, cross-type testing of motherboards is conducted. Collecting the training signatures on one motherboard and testing components from different motherboards speeds up the process and decreases the cost. Using measurements taken while exciting the components for 1 s, our method achieves a classification accuracy greater than 96% across all components tested. These results demonstrate that this method can recognize

components based on their emanations, even if the components are integrated onto completely different motherboards.

[7] S. Sangodoyin, F. Werner, B. B. Yilmaz, C-L Cheng, E. M. Ugurlu,, N. Sehatbakhsh, M. Prvulovic, and A. Zajic, "Side-channel propagation measurements and modeling for hardware security in IoT devices," to appear in *IEEE Transactions on Antennas and Propagation*, 2020.

Abstract: The ubiquitous inter-connectivity of electronic devices offered by Internet-of-Things (IoT) networks has been increasingly embraced in a wide range of applications. In IoT networks, threats to hardware security are often not perceived as serious, with the assumption that an attack could only be carried out at close proximity. However, in this paper, we show that through Electromagnetic (EM) side-channel signal leakage, operational information and program activities of IoT devices and Field Programmable Gate Array (FPGA) modules can be garnered from approximately 200 m away in an outdoor Line-of- Sight (LOS) environment. We describe an extensive measurement campaign conducted to investigate the aforementioned leakage and provide propagation models that can be used to predict the power (and corresponding variation i.e., shadowing gain) of the EM side-channel signal emanation at various distances, scenarios and environments. Our results show that the received power of the emanated EM side-channel (carrier) signal varies from about -61.64 dBm at 1 m to about -112 dBm at 200 m in the outdoor LOS environment. Furthermore, a received signal power of about -73.55 dBm was observed at 1 m and -88 dBm was recorded at 10 m in an indoor LOS environment. Power variation (shadowing gain) of about 3.6 dB and 2.0 dB were observed in the outdoor and indoor environments, respectively. This work is relevant for EM side-channel leakage countermeasure development and

provides pertinent information to embedded systems and wireless network security engineers.

[8] L. Nguyen, C. Cheng, F. Werner, M. Prvulovic, and A. Zajic, “A comparison of backscattering, EM, and power side-channels and their performance in detecting software and hardware intrusions,” in *Journal of Hardware and System Security*, March 2020.

Abstract: Side-channel analysis is a powerful tool from both an attacker's and defender's perspective. Understanding similarities and differences among types of side-channels is a necessary step in better utilization of side-channels. This paper addresses this problem by modeling and quantitatively comparing backscattering, electromagnetic (EM), and power side-channels and discusses the performance of these three side-channels for detecting software malware and hardware Trojans (HT). The results show that for larger changes in the signals, such as those caused by malware intrusions, all three side-channels perform similarly. However, when smaller changes need to be observed, such as those caused by HTs, the backscattering side-channel outperforms EM and power side channels.

[9] B. Yilmaz, N. Sehatbakhsh, A. Zajić and M. Prvulovic, "Communication model and capacity limits of covert channels created by software activities," in *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1891-1904, 2020.

Abstract: It has been shown that digital and/or analog characteristics of electronic devices during executing programs can create a side-channel which an attacker can exploit to extract sensitive information such as cryptographic keys. When the attacker modifies the software application to exfiltrate sensitive information through a channel, this channel is called a covert channel. In this paper, we model this covert channel as a communication channel and derive upper and lower capacity bounds. Because the covert channels are not designed to transmit information, they are exposed not only to the errors created by the transmission, but also by varying the execution time of computer activities, and/or by insertions from other activities such as interrupts, stalls, etc. Combining all of these effects, we propose to model the covert channel as an insertion channel where the transmitted sequence is a pulse amplitude modulated signal with random pulse positions. Utilizing this model, we derive capacity bounds of the covert channel with random insertion and substitution due to the noise and jitter errors, and propose a receiver design that can correctly detect the computer-activitycreated signals. To illustrate the severity of leakages, we perform experiments with high clock speed devices at some distance. Further, the theoretical derivations are compared to empirical results, and show good agreement.

[10] B. Yilmaz, M. Prvulovic, and A. Zajic, "Electromagnetic side-channel information leakage created by execution of series of instructions in a computer processor in *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 776-789, 2020.

Abstract: The side-channel leakage is a consequence of program execution in a computer processor, and understanding relationship between code execution and information leakage is a necessary step in estimating information leakage and its capacity limits. This paper proposes a methodology to relate program execution to electromagnetic side-channel emanations, and estimates side-channel information capacity created by execution of series of instructions (e.g. a function, a procedure, or a program) in a processor. To model dependence among program instructions in a code, we propose to use Markov Source model, which includes the dependencies among sequence of instructions as well as dependencies among instructions as they pass through a pipeline of the processor. The emitted EM signals during instruction executions are natural choice for the inputs into the model. To obtain the channel inputs for the proposed model, we derive a mathematical relationship between the emanated instruction signal power (ESP) and total emanated signal power while running a program. Then, we derive leakage capacity of electromagnetic (EM) side channels created by execution of series of instructions in a processor. Finally, we provide experimental results to demonstrate that leakages could be severe and that a dedicated attacker could obtain important information.

[11] Nader Sehatbakhsh, Alireza Nazari, Monjur Alam, Frank Werner, Yuanda Zhu, Alenka Zajic, and Milos Prvulovic, "REMOTE: Robust External Malware Detection Framework by Using Electromagnetic Signals," in *IEEE Transactions on Computers*, 69, no. 3, pp. 312-326, 1 March 2020.* – chosen for the Featured Paper

Abstract: Cyber-physical systems (CPS) are controlling many critical and sensitive aspects of our physical world while being continuously exposed to potential cyber-attacks. These systems typically have limited performance, memory, and energy reserves, which limits their ability to run existing advanced malware protection, and that, in turn, makes securing them very challenging. To tackle these problems, this paper proposes, REMOTE, a new robust framework to detect malware by externally observing Electromagnetic (EM) signals emitted by an electronic computing device (e.g., a microprocessor) while running a known application, in real-time and with a low detection latency, and without any a priori knowledge of the malware. REMOTE does not require any resources or infrastructure on, or any modifications to, the monitored system itself, which makes REMOTE especially suitable for malware detection on resource-constrained devices such as embedded devices, CPSs, and Internet of Things (IoT) devices where hardware and energy resources may be limited. To demonstrate the usability of REMOTE in real-world scenarios, we port two real-world programs (an embedded medical device and an industrial PID controller), each with a meaningful attack (a code-reuse and a code-injection attack), to four different hardware platforms. We also port shellcode-based DDoS and Ransomware attacks to five different standard applications on an

embedded system. To further demonstrate the applicability of REMOTE to commercial CPS, we use REMOTE to monitor a Robotic Arm. Our results on all these different hardware platforms show that, for all attacks on each of the platforms, REMOTE successfully detects each instance of an attack and has < 0.1% false positives. We also systematically evaluate the robustness of REMOTE to interrupts and other system activity, to signal variation among different physical instances of the same device design, to changes over time, and to plastic enclosures and nearby electronic devices. This evaluation includes hundreds of measurements and shows that REMOTE achieves excellent accuracy (< 0.1% false positive and >99.9% true positive rates) under all these conditions.

[12] H. Khan, N. Sehatbakhsh, L. Nguyen, Milos Prvulovic, and Alenka Zajic, "Malware detection in embedded systems using neural network model for electromagnetic side-channel signals," *Journal of Hardware and System Security*, 305–318, August 2019.

Abstract: We propose a novel malware detection system for critical embedded and cyber-physical systems (CPS). The system exploits electromagnetic (EM) side-channel signals from the device to detect malicious activity. During training, the system models EM emanations from an uncompromised device using a neural network. These EM patterns act as fingerprints for the normal program activity. Next, we continuously monitor the target device's EM emanations. Any deviation in the device's activity causes a variation in the EM fingerprint, which in turn violates the trained model, and is reported as an anomalous activity. The system can monitor the target device remotely (without any physical contact), and does not require any modification to the monitored system. We evaluate the system with different malware behavior (DDoS, Ransomware and Code Modification) on different applications using an Altera Nios-II soft-processor. Experimental evaluation reveals that our framework can detect DDoS and Ransomware with 100% accuracy (AUC = 1.0), and stealthier code modification (which is roughly a 5 µs long attack) with an AUC = 0.99, from distances up to 3 m. In addition, we execute control- ow hijack, DDoS and Ransomware on different applications using an A13-OLinuXino – a Cortex A8 ARM processor single board computer with Debian Linux OS. Furthermore, we evaluate the practicality and the robustness of our system on a medical CPS, implemented using two different devices (TS-7250

and A13-OLinuXino), while executing a control-ow hijack attack. Our evaluations show that our framework can detect these attacks with 100% accuracy.

[13] Haider A. Khan, Nader Sehatbakhsh, Luong N. Nguyen, Robert Callan, Arie Yeredor, Milos Prvulovic, and Alenka Zajic, "IDEA: intrusion detection through electromagnetic-signal analysis for critical embedded and cyber-physical systems," in *IEEE Transactions on Dependable and Secure Computing*, 2019.

Abstract: We propose a novel framework called IDEA that exploits electromagnetic (EM) side-channel signals to detect malicious activity on embedded and cyber-physical systems (CPS). IDEA first records EM emanations from an uncompromised reference device to establish a baseline of reference EM patterns. IDEA then monitors the target device's EM emanations. When the observed EM emanations deviate from the reference patterns, IDEA reports this as an anomalous or malicious activity. IDEA does not require any resource or infrastructure on, or any modification to, the monitored system itself. In fact, IDEA is isolated from the target device, and monitors the device without any physical contact. We evaluate IDEA by monitoring the target device while it is executing embedded applications with malicious code injections such as DDoS, Ransomware and code modification. We further implement a control-flow hijack attack, an advanced persistent threat, and a firmware modification on three CPSs: an embedded medical device called SyringePump, an industrial PID Controller, and a Robotic Arm, using a popular embedded system, Arduino UNO. The results demonstrate that IDEA can detect different attacks with excellent accuracy (AUC > 99.5%, and 100% detection with less than 1% false positives) from distances up to 3 m.

[14] J. Dinkić, D. Olcan, A. Djordjević, and A. Zajic," Design and Optimization of Nonuniform helical antennas with linearly varying geometrical parameters," *IEEE Access* 7, pp. 855-866, 2019.

ABSTRACT Nonuniform helical antennas have many degrees of freedom, which makes the search space for the optimal design very challenging. The objective of this paper is to systematically analyze nonuniform helical antennas with linearly varying geometrical parameters and to provide analytical equations that approximate the optimal design and the gain of the designed antennas. Using various optimization algorithms, we made a large database of the optimal nonuniform helical antennas with linearly varying geometrical parameters. Based on these results, we made analytical equations that approximate the optimal design and the gain of the designed antennas. These equations allow for a fast design procedure yielding all necessary parameters needed for the design and fabrication of nonuniform helical antennas that meet specified characteristics. Special attention is devoted to antenna losses. Antennas designed following the presented procedure achieve around 2.5 dB higher gain than uniform helical antennas of the same axial length, while maintaining the bandwidth and axial ratio. As a verification of the proposed design procedure, a helical antenna with the central operating frequency of 1 GHz was designed, simulated, fabricated, and measured. The

comparison between measured and simulated results confirms the validity of the presented design procedure.

[15] Sinan Adibelli, Prateek Juyal, Chia-Lin Cheng, and Alenka Zajic, "THz near field focusing using a 3D printed Cassegrain configuration for backscattered side-channel detection," *IEEE Transactions on Antennas and Propagation*, vol. 67, no. 10, pp. 6627-6638, October 2019.

Abstract: This paper presents the use of THz near field focusing for backscatter side channel detection. Near field focusing is done by using cassegranian reflector configuration. The focuser is designed to produce the focused beam 28 cm away from the antenna aperture. The focusing is done in the near field region by axially moving the sub-reflector from the focal point. It is observed that the sub-reflector position has to shift approximately 11 wavelengths along the axis to create the focus at the required location. The focused antenna gain is 46 dBi while the 3 dB focus width and depth of the designed antenna is ~ 4 mm and 10 cm, respectively. It is found that the focal plane position is sensitive to the sub-reflector shifts and it is observed that 1 mm change in the sub-reflector position can shift the focal plane by ~ 2 cm. The simulations are compared with measurement results of a fabricated prototype and good agreement is observed. The antenna is fabricated by using 3D printing technology, which allows rapid prototyping. Finally, we have demonstrated the detection of backscatter side channel from the board placed at 28 cm away from the designed antenna. The received power level of the backscatter signal increases by 6 dB as compared to horn antenna.

[16] M. Ruble, C. E. Hayes, M. Welborn, A. Zajić, M. Prvulovic and A. M. Pitruzzello, "Hyperdimensional Bayesian Time Mapping (HyperBaT): A Probabilistic Approach to Time Series Mapping of Non-Identical Sequences," in *IEEE Transactions on Signal Processing*, vol. 67, no. 14, pp. 3719-3731, July, 2019.

Abstract: A common problem in time series analysis is mapping the related elements between two sequences as they progress in time. Methods such as dynamic time warping (DTW) and hidden Markov models (HMM) have good performance in mapping time series signals with repeated (warped) elements relative to a reference signal. However, there is not an adequate method for mapping time series signals with inserted or deleted elements. This work introduces hyper-dimensional Bayesian time-mapping (HyperBaT), a machine learning algorithm that maps two time sequence signals that may contain inserted, deleted, or warped elements. Additionally, HyperBaT estimates the common underlying signal shared between the two sequences. The algorithm is presented in a general context so that it can be used in a variety of applications. There are many relevant areas, including speech processing, genetic sequencing, electronic warfare, communications, and radar processing, that process signals containing inserted or deleted elements. As an example, HyperBaT is applied to side-channels where it maps radio frequency (RF) side-channel signals emitted from a computing device processor, which can be used to track control flow execution and monitor for malicious activity.

[17] Jelena Dinkić, Dragan Olcan, Antonije Djordjević, and Alenka G. Zajic," High-Gain Quad Array of Nonuniform Helical Antennas," *International Journal of Antennas and Propagation,* vol. 2019, Article ID 8421809, 12 pages, 2019.

Abstract: We present a design of a high-gain quad array of non-uniform helical antennas. The design is obtained by optimization of a 3-D numerical model of four non-uniform helical antennas placed above a ground plane, including a model of a feeding network, utilizing the method of moments with higher-order basis functions. The gain of one optimal non-uniform helical antenna can be for about 2.5 dB higher than the gain of a uniform helical antenna of the same axial length. Creating a 2×2 array further increases the gain for up to about 6 dB. The resulting quad array fits into a box of dimensions 2.5×3.3×3.3 wavelengths and the gain in the main radiating direction is about 20.5 dBi in the frequency range from 0.9 GHz to 1.1 GHz. The design is verified by measurements of a prototype of the quad array.

[18] Prateek Juyal, Sinan Adibelli, Alenka Zajic "A Directive Antenna Based on Conducting Disc for Detecting Unintentional EM Emissions at Larger Distances," *IEEE Transactions on Antennas and Propagation*, vol.66, pp. 6751-6761, December 2018.

Abstract: This paper proposes a novel high gain planar antenna design that consists of conducting metallic discs suspended on air and operates at 1 GHz. The antenna is designed for receiving the unintentional EM emanations generated by one or multiple embedded, "smart" electronic systems. The antenna consists of two layers of slotted conducting metal discs suspended on air and placed above the ground plane using teflon screws. The circular discs are designed to operate in higher order $\boldsymbol{TM12}$ mode. The screws location are the electric field nulls along the disc radius. The upper layer is 2×2 array of slotted circular discs electromagnetically coupled by lower identical disc which is fed directly by a single coaxial feed. The complete fabrication of antenna is done using aluminum metal sheets and involves no use of dielectric substrate. The antenna has a peak gain of 19 dBi with impedance bandwidth ($\boldsymbol{S11} \leq \boldsymbol{-6}$ dB) of 6.7%. The simple and cost effective design can be easily scaled to higher frequencies.

[19] Baki Berkay Yilmaz, Robert Callan, Milos Prvulovic, and Alenka Zajic, "Capacity of the EM Covert/Side-Channel Created by the Execution of Instructions in a Processor," *IEEE Transactions on Information Forensics and Security*, vol. 13, pp. 605-620, June 2018.

Abstract: The goal of this paper is to answer how much information is "transmitted" by execution of particular sequence of instructions in a processor. Introducing such a measure would provide quantitative guidance for designing programs and computer hardware that minimizes inadvertent (side channel) information leakage, and would also help detect parts of a program or hardware design that have unusually high leakage (i.e. were designed to function as covert channel "transmitters"). To answer this question, we propose a new method to estimate the maximum information leakage through EM signals generated by execution of instructions in a processor. We start by deriving a mathematical relationship between electromagnetic side-channel energy (ESE) of individual instructions and the measured pairwise side-channel signal power. Then, we use this measure to calculate the transition probabilities needed for estimating capacity. Finally, we propose a new method to estimate side/covert channel capacity created by the execution of instructions in a processor and illustrate our results in several computer systems.

[20] Frank Werner, Derrick Chu, Antonije R. Djordjevic, Dragan I. Olcan, Milos Prvulovic, and Alenka Zajic, "A Method for Efficient Localization of Magnetic-field Sources Excited by the Execution of Instructions in a Processor," *IEEE Transactions on Electromagnetic Compatibility*, vol. 60, pp. 613-622, June 2018.

Abstract: This paper proposes a method for efficient identification of instruction dependent sources on a printed-circuit board (PCB) by localizing magnetic field sources from

a limited number of measurements around the PCB. We first excite the processor by generating an artificial leakage signal at a specific frequency that is directly related to processor instructions. Then, we collect all three components of the magnetic field, but only at locations around the edge of the board. Furthermore, we model these magnetic field sources and then solve a forward-backward optimization problem using the model and measured data to identify the locations of the magnetic field sources, the magnitudes of the moments, and their orientations. The localization results are first verified using simulations, then tested when noise is added to the simulation results, and finally verified against measurements on FPGA and IoT development boards. The results show that the number of strong magnetic field sources on a board depends on the instructions used to excite the board. Furthermore, the results show that the proposed localization algorithm can accurately identify those sources, regardless of the frequency at which the measurements are conducted and the instruction pairs that are executed. Finally, the proposed method can significantly reduce the number of measurement points and the time needed to identify magnetic field sources on a PCB.

[21] M. Prvulovic, A. Zajić, R. Callan, and C. Wang, "A method for finding frequency-modulated and amplitude-modulated electromagnetic emanations in computer systems," *IEEE Transactions on Electromagnetic Compatibility*, vol. 59, no 1, pp. 34-42, 2017.

Abstract: This paper presents an algorithm for finding carriers of frequency-modulated (FM) and amplitude-modulated (AM) electromagnetic (EM) emanations from computer systems. Computer systems create EM emanations across the RF spectrum making it difficult, error-prone, and time-consuming to find the relatively few emanations that expose sensitive information. One of the most common and simplest mechanisms for information leakage occurs when an amplitude or a frequency of an existing strong signal (e.g., a processor or memory clock) is amplitude or frequency modulated by a system activity. If the system activity can be linked to sensitive information, this results in information leakage. We present an algorithm for automatically finding these AM and FM modulated signals, demonstrate the algorithm's performance on several different types of processors and systems (desktop, laptop, and smart phone), and compare the results to an exhaustive manual search. We also verify that all signals identified by the algorithm can be traced to plausible unintentional modulation mechanisms to illustrate that these signals can potentially cause information leakage. This algorithm can be an important tool for system designers to quickly identify circuits that are leaking sensitive information.

[22] J. Dinkic, D. Olcan, A. Djordjevic, A. Zajic, "Comparison of the optimal uniform and nonuniform lossy helical antennas," IEEE Proceedings of AP-S/URSI, pp. 1-2, July 2020, Montreal, Canada.

Abstract: Various uniform and non-uniform helical antennas are used in practice. In this paper we compare uniform and non-uniform helical antennas with respect to their losses. Further, we present the ranges of wire conductivities for which the non-uniform antennas are preferable choice and the conductivities where uniform antennas are recommended.

[23] B. Yilmaz, E. Ugurlu, A. Zajic, and M. Prvulovic, "Cell-phone classification: a convolutional neural network approach exploiting electromagnetic emanations," in Proceedings of ICASSP, pp. 1-5, May 2020, Barcelona, Spain.

Abstract: In this paper, we propose a methodology to identify both the brand of a cell-phone, and the status of its camera by exploiting electromagnetic (EM) emanations. The method composes two parts: Feature extraction and Convolutional Neural Netwotk (CNN). We first extract features by averaging magnitudes of short-time Fourier transform (STFT) of the measured EM signal, which helps to reduce input dimension of the neural network, and to filter spurious emissions. The extracted features are fed into the proposed CNN, which contains two convolutional layers (followed by max-pooling layers), and four fully-connected layers. Finally, we provide experimental results which exhibit more than 99% classification accuracy for the test signals.

[24] B. Yilmaz, E. Ugurlu, F. Werner, A. Zajic, and M. Prvulovic, "Program profiling based on Markov models and EM emanations," in Proceedings of SPIE, April 2020, Anaheim, CA.

Abstract: As one of the fundamental approaches for code optimization and performance analysis, profiling software activities can provide information on the existence of malware, code execution problems, etc. In this paper, we propose a methodology to profile a system with no overhead. The approach leverages electromagnetic (EM) emanations while executing a program, and exploits its flow diagram by constructing a Markov model. The states of the model are considered as the heavily executed blocks (called hot paths) of the program, and the transition between any two states is possible only if there exists a branching operation which enables execution of corresponding states without any intermediate state. To identify the state of the program, we utilize a supervised learning method. To do so, we first collect signals for each state, extract features, and generate a dictionary. The features are considered as the activated frequencies when the program is executed. The assumption here is that there exists at least one unique frequency component that is only active for one unique state. Moreover, to degrade the effect of interruptions and other signals emanated from other parts of the device, and to obtain signals with high Signal-to-Noise Ratio (SNR), we average the output of Short-Time Fourier Transform (STFT). After extracting features, we apply Principle Component Analysis (PCA) for dimension reduction which helps monitoring systems in real

time. Finally, we describe experimental setup and show results to demonstrate that the proposed methodology can detect malware activity with high accuracy.

[25] L. Nguyen, B. Yilmaz, C. Cheng, M. Prvulovic, and A. Zajic, "A novel clustering technique using backscattering side-channel for counterfeit IC detection," in Proceedings of SPIE, April 2020, Anaheim, CA.

Abstract: Over the past few years, globalization of the semiconductor supply chain has led companies to outsource much of the production cycle for integrated circuits (ICs). While outsourcing helps companies significantly reduce their cost and time-to-market, it also introduces concerns about the trustworthiness of an IC. One of the most serious problems is counterfeiting of ICs, which not only negatively impacts innovation and economic growth of the IC industry, but also creates serious threats and risks for systems that incorporate those counterfeit ICs. This paper proposes a novel method that uses the backscattering side-channel to cluster ICs such that counterfeits are separated from legitimate ICs. The backscattering side-channel, which has been introduced only recently, has been proven to outperform other side-channels in detecting hardware Trojan horses (HTs), i.e. ICs where additional logic gates (and connections to existing logic gates) have been added. In this work we use it to robustly separate ICs into legitimate and counterfeit ones, even when only layout or placement of the IC has changed, without any added logic or connections. We evalute our technique on a set of ten boards over six different counterfeit IC designs, and find that our technique tolerates manufacturing variations among different hardware instances, detecting counterfeit ICs with 100% accuracy and 0% false positives.

[26] Sangodoyin, F. Werner, B. B. Yilmaz, C. Cheng, E. M. Ugurlu, N. Sehatbakhsh, M. Prvulovic, and A. Zajic, "Remote monitoring and propagation modeling of EM side-channel signals for IoT device security," in Proceedings of 14th European Conference on Antennas and Propagation (EuCAP), pp. 1-5., March 2020, Copenhagen, Denmark.

Abstract: This paper presents results from an investigation into long-range detection and monitoring of Electromagnetic (EM) side-channel signals leaked from Internet-of-Things (IoT) and Field Programmable Gate Array (FPGA) devices. Our work shows that operational information and program activities of the IoT and FPGA modules can be garnered at distances excess of 25 m in an indoor Line-Of-Sight (LOS) environment, while at about 10 m in an indoor (through wall) Non-Line-Of-Sight (NLOS) scenario. We provide a propagation model that can be used to predict the received power (and corresponding variation i.e., shadowing gain) of leaked EM side-channel signals at various distances and scenarios. Benchmark program bitcount used in the performance evaluation of ARM-based microprocessors and a microbenchmark SAVAT running on an IoT device were detected and monitored remotely in our work.

[27] Nader Sehatbakhsh, Baki Yilmaz, Alenka Zajic, and Milos Prvulovic, "A New Side-Channel Vulnerability on Modern Computers by Exploiting Electromagnetic Emanations from the Power Management Unit," in Proceedings of the 26th IEEE International Symposium on High-Performance Computer Architecture, 2020.

Abstract: This paper presents a new micro-architectural vulnerability on the power management units of modern computers which creates an electromagnetic-based side channel. The key observations that enable us to discover this side-channel are 1) in an effort to manage and minimize power consumption, modern microprocessors have a number of possible operating modes (power states) in which various sub-systems of the processor are powered down, 2) for some of the transitions between power states, the processor also changes the operating mode of the voltage regulator module (VRM) that supplies power to the affected sub-system, and 3) the electromagnetic (EM) emanations from the VRM are heavily dependent on its operating mode. As a result, these state-dependent EM emanations create a side-channel which can potentially reveal sensitive information about the current state of the processor and, more importantly, the programs currently being executed. To demonstrate the feasibility of exploiting this vulnerability, we create a covert channel by utilizing the changes in the processor's power states. We show how such a covert channel can be leveraged to exfiltrate sensitive information from a secured and completely isolated (air-gapped) laptop system by placing a compact, inexpensive receiver in close proximity to that system. To further show the severity of this attack, we also demonstrate how such a covert channel can be established when

the target and the receiver are several meters away from each other, including scenarios where the receiver and the target are separated by a wall. Compared to the state-of-the-art, the proposed covert channel has >3x higher bit-rate. Finally, to demonstrate that this new vulnerability is not limited to being used as a covert channel, we demonstrate how it can be used for attacks such as keystroke logging.

[28] Nader Sehatbakhsh, Baki Yilmaz, Alenka Zajic, Milos Prvulovic, "EMSim: A Microarchitecture-Level Simulation Tool for Modeling Electromagnetic Side-Channel Signals," in Proceedings of the 26th IEEE International Symposium on High-Performance Computer Architecture, 2020. **– nominated for the best paper award**

Abstract: Side-channel attacks have become a serious security concern for computing systems, especially for embedded devices, where the device is often located in, or in close proximity to, a public place, and yet the system contains sensitive information. To design systems that are highly resilient to such attacks, an accurate and efficient design stage quantitative analysis of side-channel leakage is needed. For many system properties (e.g., performance, power, etc.), cycle-accurate simulation can provide such an efficient-yet-accurate design-stage estimate. Unfortunately, for an important class of side-channels, electromagnetic emanations, such a model does not exist, and there has not even been much quantitative evidence about what level of modeling detail (e.g., hardware, microarchitecture, etc.) would be needed for high accuracy. This paper presents EMSim, an approach that enables simulation of the electromagnetic (EM) side-channel signals cycle-by-cycle using the detailed micro-architectural model of the device. To evaluate EMSim, we compare the simulated signals against actual EM signals emanated from real hardware (a RISC-V processor implemented on an FPGA), and find that they match very closely. To gain further insights, we also experimentally identify how the accuracy of the simulation degrades when key micro-architectural features (e.g., pipeline stall, cache-miss, etc.) and other hardware behaviors (e.g.,

data-dependent switching activity) are omitted from the simulation model. We further evaluate how robust the simulation-based results are, by comparing them to real signals collected in different conditions (manufacturing, distance, etc.). Finally, to show the applicability of EMSim, we demonstrate how it can be used to measure side-channel leakage through simulation at design-stage.

[29] Baki Yilmaz, Elvan Ugurlu, Alenka Zajic, and Milos Prvulovic, “Detecting Cellphone Camera Status at Distance by Exploiting Electromagnetic Emanations,” in Proceedings of IEEE MILCOM, November 2019, pp. 1-6, Norfolk, VA.

Abstract: This paper investigates unintended radiated emissions from cellphones to identify operational status of rear/front camera. We implement a supervised learning method to achieve our goal. In the training phase, we collect data for possible combinations of phone model and camera status. Then, we apply two-phase-dimension-reduction method for better and effective classification. The first dimension-reduction phase is averaging magnitudes of frequency components of a sliding window, which is followed by applying principle component analysis (PCA) technique to reduce the dimension further. In testing phase, k Nearest-Neighbors (k-NN) algorithm is utilized to classify test data. Finally, we provide examples to show that emanated EM signals from cellphone cameras can exfiltrate useful information.

[30] Baki Yilmaz, Alenka Zajic, and Milos Prvulovic, “Capacity of EM Side Channel Created by Instruction Executions in a Processor,” in Proceedings of IEEE IEMCON, October 2019, pp. 1-5, Vancouver, CA.

Abstract: This paper proposes a methodology to estimate leakage capacity of electromagnetic (EM) side channels created by execution of instruction sequences (e.g. a function, a procedure, or a program) in a processor. We propose to use Markov Source model to include the dependencies that exist in instruction sequence since each program code is written systematically to serve a specific task. The channel input sources are considered as the emitted EM signals while executing an instruction. We derive a mathematical relationship between the emanated instruction power (IP) and total emanated signal power while running a microbenchmark to obtain the channel input powers. The results demonstrate that leakages could be severe enough for a dedicated attacker to obtain some prominent information.

[31] Nader Sehatbakhsh, Alireza Nazari, Haider Khan, Alenka Zajic, and Milos Prvulovic, "EMMA: Hardware/Software Attestation Framework for Embedded Systems Using Electromagnetic Signals," in Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, 2019, pp.1-11, 12-15 October 2019, Columbus, OH.

Abstract: Establishing trust for an execution environment is an important problem, and practical solutions for it rely on attestation, where an untrusted system (prover) computes a response to a challenge sent by the trusted system (verifier). The response computation typically involves calculating a checksum of the prover's program, which the verifier checks against expected values for a "clean" (trustworthy) system. The main challenge in attestation is that, in addition to checking the response, the verifier also needs to verify the integrity of the response computation itself, i.e., that response computation itself has not been tampered with to produce expected values without measuring the verifier's actual code and environment. On higher-end processors, this integrity is verified cryptographically, using dedicated trusted hardware. On embedded systems, however, constraints prevent the use of such hardware support. Instead, a popular approach is to use the request-to-response time as a way to establish confidence. However, the overall request-to-response time provides only one coarse-grained measurement from which the integrity of the attestation is to be inferred, and even that is noisy because it includes the network latency and/or variations due to micro-architectural events. Thus, the attestation is vulnerable to attacks where the adversary has tampered with response computation, but the resulting additional computation time is small relative to the overall request-to-response time. In

this paper, we make a key observation that the existing approach of execution-time measurement for attestation is only one example of using externally measurable side-channel information and that other side-channels, some of which can provide much finer-grain information about the computation, can be used. As a proof of concept, we propose EMMA, a novel method for attestation that leverages electromagnetic side-channel signals that are emanated by the system during response computation, to confirm that the device has, upon receiving the challenge, actually computed the response using the valid program code for that computation. This new approach requires physical proximity, but imposes no overhead to the system, and provides accurate monitoring during the attestation. We implement EMMA on a popular embedded system, Arduino UNO, and evaluate our system with a wide range of attacks on attestation integrity. Our results show that EMMA can successfully detect these attacks with high accuracy. We compare our method with the existing methods and show how EMMA outperforms them in terms of security guarantees, scalability, and robustness.

[32] Frank Werner, Antonije Djordjevic, and Alenka Zajic, "A compact probe for EM side-channel attacks on cryptographic systems," in Proceedings of IEEE International Symposium on Antennas and Propagation, pp. 1-2, July 2019, Atlanta, GA.

Abstract: A shielded loop probe design for evaluating EM side-channels attacks on cryptographic systems is described. This probe is compact and is sensitive enough to measure extremely weak signals that usually comprise EM side-channels. Furthermore, this probe can

greatly suppress the influence of the electric field on its measurements. At its center frequency, the probe has a sensor factor of -0.17 dB S/m and electric field suppression ratio of 30.18 dB.

[33] Richard Rutledge, Sunjae Park, Haider Khan, Alessandro Orso, Milos Prvulovic, and Alenka Zajic, "Zero-overhead path prediction with progressive symbolic execution," In Proceedings of the IEEE 41st International Conference on Software Engineering (ICSE '19), pp. 234-245, May 2019, Montreal CA.

Abstract: In previous work, we introduced zero-overhead profiling (ZOP), a technique that leverages the electromagnetic emissions generated by the computer hardware to profile a program without instrumenting it. Although effective, ZOP has several shortcomings: it requires test inputs that achieve extensive code coverage for its training phase; it predicts path profiles instead of complete execution traces; and its predictions can suffer unrecoverable accuracy losses. In this paper, we present zero-overhead path prediction (ZOP-2), an approach that extends ZOP and addresses its limitations. First, ZOP-2 achieves high coverage during training through progressive symbolic execution (PSE)—symbolic execution of increasingly small program fragments. Second, ZOP-2 predicts complete execution traces, rather than path profiles. Finally, ZOP-2 mitigates the problem of path mispredictions by using a stateless approach that can recover from prediction errors. We evaluated our approach on a set of benchmarks with promising results; for the cases considered, (1) ZOP-2 achieved over 90% path prediction accuracy, and (2) PSE covered feasible paths missed by traditional symbolic execution, thus boosting ZOP-2's accuracy.

[34] Baki Yilmaz, Elvan Ugurlu, Alenka Zajic, Milos Prvulovic, “Instruction level program tracking using electromagnetic emanations,” Proceedings of SPIE, pp.1-6, April 2019, Baltimore, MD.

Abstract: Monitoring computer system activities on the instruction level provides more resilience to malware attacks because these attacks can be analyzed better by observing the changes on the instruction level. Assuming the source code is available, many training signals can be collected to track the instruction sequence to detect whether a malware is injected or the system works properly. However, training signals have to be collected with high sampling rate to ensure that the significant features of these signals do not vanish. Since the clock frequencies of the current computer systems are extremely high, we need to have a commercial device with high sampling rate, i.e. 10GHz, which either costs remarkably high, or does not exist. To eliminate the deficiencies regarding the insufficient sampling rate, we propose a method to increase the sampling rate with the moderate commercial devices for training symbols. In that respect, we first generate some random instruction sequences which exist in the inspected source code. Then, these sequences are executed in a for-loop, and emanated electromagnetic (EM) signals from the processor are collected by a commercially available device with moderate sampling rate, i.e. sampling rate is much smaller than the clock frequency. Lastly, we apply a mapping of the gathered samples by utilizing modulo of their timings with respect to execution time of overall instruction sequence. As the final step, we

provide some experimental results to illustrate that we successfully track the instruction sequence by applying the proposed approach.

[35] Baki Yilmaz, Alenka Zajic, and Milos Prvulovic, "Capacity of deliberate side-channels created by software activities," Proceedings of IEEE MILCOM, October 2018, pp. 1-6, Los Angeles, CA.

Abstract: It has been shown that electromagnetic (EM) emanations that are result of instruction executions in a computer system can create a wireless side-channel which attackers can use to extract sensitive information such as a cryptography key. When an attacker modifies the software application to exfiltrate sensitive information through a channel, this channel is called a deliberate side-channel or a covert channel. Because deliberate side-channels are not created for a conventional wireless communication to transmit information, these channels are exposed not only to the errors created by the wireless transmission (the transmitted signal propagates through a channel hindered by metal and plastic), but also by varying execution time of computer activities, and by insertions from other computer activities such as interrupts. Combining all of these effects, we propose to model deliberate side-channels as an insertion channel where the transmitted sequence is a pulse amplitude modulated signal with varying pulse width. Utilizing this model, we derive upper and lower bounds for the channel capacity of the deliberate side-channels. The bounds demonstrate the severity of data leakage through deliberate side channels by revealing the potential to transmit high data rates. Moreover, the proposed bounds for the channel capacity can be employed by any noisy insertion channel and provides more confidential results.

[36] M. Dey, A. Nazari, A. Zajic and M. Prvulovic, "EMPROF: Memory Profiling Via EM-Emanation in IoT and Hand-Held Devices," 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Fukuoka, Japan, 2018, pp. 881-893.

Abstract: This paper presents EMPROF, a new method for profiling the performance impact of the memory subsystem without any support on, or interference with, the profiled system. Rather than rely on hardware support and/or software instrumentation on the profiled system, EMPROF analyzes the system's EM emanations to identify processor stalls that are associated with last-level cache (LLC) misses. This enables EMPROF to accurately pinpoint LLC misses in the execution timeline and to measure the cost (stall time) of each miss. Since EMPROF has zero "observer effect", so it can be used to profile applications that adjust their activity to their performance. It has no overhead on target machine, so it can be used for profiling embedded, hand-held, and IoT devices which usually have limited support for collecting, and limited resources for storing, the profiling data. Finally, since EMPROF can profile the system as-is, its profiling of boot code and other hard-to-profile software components is as accurate as its profiling of application code. To illustrate the effectiveness of EMPROF, we first validate its results using microbenchmarks with known memory behavior, and also on SPEC benchmarks running a cycle-accurate simulator that can provide detailed ground-truth data about LLC misses and processor stalls. We then demonstrate the

effectiveness of EMPROF on real systems, including profiling of boot activity, show how its results can be attributed to the specific parts of the application code when that code is available, and provide additional insight on the statistics reported by EMPROF and how they are affected by the EM signal bandwidth provided to EMPROF.

[37] M. Alam, H. Khan, M. Day, R. Callan, N. Sinha, A. Zajic, and M. Prvulovic, "One & done – A Single-Decryption EM-Based Attack on OpenSSL's Constant-Time Blinded RSA ," USENIX Security, August 2018.

Abstract: This paper presents the first side channel attack approach that, without relying on the cache organization and/or timing, retrieves the secret exponent from a single decryption on arbitrary ciphertext in a modern (current version of OpenSSL) fixed window constant-time implementation of RSA. Specifically, the attack recovers the exponent's bits during modular exponentiation from analog signals that are unintentionally produced by the processor as it executes the constant-time code that constructs the value of each "window" in the exponent, rather than the signals that correspond to squaring/multiplication operations and/or cache behavior during multiplicand table lookup operations. The approach is demonstrated using electromagnetic (EM) emanations on two mobile phones and an embedded system, and after only one decryption in a fixed-window RSA implementation it recovers enough bits of the secret exponents to enable very efficient (within seconds) reconstruction of the full private RSA key. Since the value of the ciphertext is irrelevant to our attack, the attack succeeds even when the ciphertext is unknown and/or when message randomization (blinding) is used. Our evaluation uses signals obtained by demodulating the signal from a relatively narrow band (40 MHz) around the processor's clock frequency (around 1GHz), which is within the capabilities of compact sub-$1,000 software-defined radio (SDR) receivers. Finally, we propose a mitigation where the bits of the exponent are only obtained from an exponent in

integersized groups (tens of bits) rather than obtaining them one bit at a time. This mitigation is effective because it forces the attacker to attempt recovery of tens of bits from a single brief snippet of signal, rather than having a separate signal snippet for each individual bit. This mitigation has been submitted to OpenSSL and was merged into its master source code branch prior to the publication of this paper.

[38] J. Dinkić, D. Olćan, A. Zajić, A. Djordjević, "Comparison of optimization approaches for designing nonuniform helical antennas," Proceedings of 2018 IEEE AP-S Symposium on Antennas and Propagation and URSI CNC/USNC, Boston, USA, July 8-13, 2018, pp. 1581-1582.

Abstract: Comparison of various optimization approaches for the design of nonuniform helical antennas is presented. The considered helices have linearly varying radius and pitch. Results show that this design has many similar (or suboptimal) solutions with significant differences in geometry. Combination of particle swarm optimization and Nelder-Mead simplex algorithm proved to be a robust and an efficient optimization approach.

[39] F. Werner, A. R. Djordjevic, D. I. Olcan, M. Prvulovic, and A. Zajic, "Experimental validation of localization method for finding magnetic sources on IoT devices," *IEEE Proceedings of EMC Europe*, pp. 1-5, Amsterdam, Netherlands, August 2018.

Abstract: Recently, we proposed a method for accurately locating instruction-dependent magnetic field sources on printed circuit boards (PCBs). This method first excites the device's processor by executing an alternating pair of two instructions at a specific frequency. Using the measurements of the magnetic field taken around the edges of the PCB and the simplex optimization algorithm, locations of the sources are evaluated. In this paper, we present extensive experimental verification of this method on two devices, a field-programmable gate array (FPGA) development board and an Internet of Things (IoT) device. The results illustrate that sources of instruction-dependent emanations are confined to a small area near the processor and that the positions of these sources are dependent on the specific instructions being executed.

[40] N. Sehatbakhsh, A. Nazari, M. Alam, A. Zajić, and M. Prvulovic, “Syndrome: spectral analysis for anomaly detection on medical IoT and embedded devices,” IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 1-8, May 2018, Washington DC.

Abstract: Recent advances in embedded and IoT (internet-ofthings) technologies are rapidly transforming health-care solutions and we are headed to a future of smaller, smarter, wearable and connected medical devices. IoT and advanced health sensors have provided more convenience to patients and physicians where physicians can now wirelessly and automatically monitor patient’s state. While these medical embedded devices provide a lot of new opportunities to improve the health care system, they also introduce a new set of security risks since they are connected to networks and run off-the-shelf operating systems. More importantly, these devices are extremely hardware and power constrained, which in turn makes securing these devices more complex. Implementing complex malware detectors or antivirus on these devices is either very costly or infeasible due to these limitations on power and resources. In this paper, we propose a new framework called SYNDROME for “externally" monitoring medical embedded devices. Our malware detector uses electromagnetic (EM) signals involuntary generated by the device as it executes a (medical) application in the absence of malware, and analyzes them to build a reference model. It then monitors the EM signals generated by the device during execution and reports an error if there is a statistically significant deviation from the reference model. To evaluate SYNDROME, we use open-source

software to implement a real-world medical device, called a Syringe Pump, on a variety of well-known embedded/IoT devices including Arduino Uno, FPGA Nios II soft-core, and two Linux IoT mini-computers: OlimexA13 and TS-7250. We also implement a control-flow hijack attack on SyringePump and use SYNDROME to detect and stop the attack. Our experimental results show that using SYNDROME, we can detect the attack for all the four devices with excellent accuracy (i.e. 0% false positive and 100% true positive) within few milliseconds after the attack starts.

[41] N. Sehatbakhsh, H. Hong, B. Lazar, B. Johnson-Smith, O. Yilmaz, M. Alam, A. Nazari, A. Zajic, and M. Prvulovic" Spectral Analysis for Anomaly Detection on Medical IoT and Embedded Devices- Experimental Demonstration," IEEE International Symposium on Hardware Oriented Security and Trust (HOST) Hardware Demo, pp.1, May 2018, Washington DC.

Abstract: This demo is the actual hardware implementation and live demonstration of our recent work "Syndrome: Spectral Analysis for Anomaly Detection on Medical IoT and Embedded Devices" that will appear in the Proceedings of HOST 2018. In this paper, we proposed a new framework called Syndrome for "externally" monitoring medical embedded devices. Our malware detector uses electromagnetic (EM) signals involuntary generated by the device as it executes a (medical) application in the absence of malware, and analyzes them to build a reference model. It then monitors the EM signals generated by the device during execution and reports an error if there is a statistically significant deviation from the reference model. To evaluate Syndrome, we use an open-source software to implement a real-world medical device, called a Syringe Pump, on a wellknown embedded system Arduino Uno. We also implement a control-flow hijack attack on SyringePump and use Syndrome to detect and stop the attack.

[42] H. Khan, M. Alam, A. Zajic, and M. Prvulovic, "Detailed tracking of program control flow using analog side-channel signals: A promise for IoT malware detection and a threat for many cryptographic implementations," Proceedings of SPIE, April 2018, Orlando FL.

Abstract: Side-channel signals have long been used in cryptanalysis, and recently they have also been utilized as a way to monitor program execution without involving the monitored system in its own monitoring. Both of these use-cases for side-channel analysis have seen steady improvement, allowing ever-smaller deviations in program behavior to be monitored (to track program behavior and/or identify anomalies) or exploited (to steal sensitive information). However, there is still very little intuition about where the limits for this are, e.g. whether a single-instruction or a single-bit difference can realistically be recovered from the signal. In this paper, we use a popular open-source cryptographic software package as a test subject to demonstrate that, with enough training data, enough signal bandwidth, and enough signal-to-noise ratio, the decision of branch instructions that cause even single-instruction-differences in program execution can be recovered from the electromagnetic (EM) emanations of an IoT/embedded system. We additionally show that, in cryptographic implementations where branch decisions contain information about the secret key, nearly all such information can be extracted from the signal that corresponds to only a single cryptographic operation (e.g. encryption). Finally, we analyze how the received signal bandwidth, the amount of training, and the signal-to-noise ratio (SNR) affect the accuracy of side-channel-based reconstruction of individual branch decisions that occur during program execution.

[43] B. Yilmaz, M. Prvulovic, and A. Zajic, “Wireless communication channel created by processor memory activity,” IEEE Proceedings of ICASSP, pp. 1-5, April 2018, Calgary, Canada.

Abstract: Electromagnetic (EM) emanations created by a software computer activity can be exploited to create a wireless channel. However, software activity experiences lack of precise synchronization and, therefore, jitter noise. In this paper, we model this type of wireless communication channel considering the jitter noise, characterize the power spectral density (PSD) of both jitter noise and signal, and analyze the performance of this channel in terms of Bit-Error-Rate (BER). We provide examples to demonstrate the capability of the EM based wireless channel.

[44] S. Adibelli, R. Golubović, A. Djordjević, D. Olćan, and A. Zajić, "Design and fabrication of non-uniform helical antennas for detection of side-channel attacks in computer systems," IEEE Proceedings of 12th European Conference on Antennas and Propagation (EuCAP), pp. 1-5, April 2018, London, UK.

Abstract: This paper presents a design, fabrication and measurement results of a helical antenna that has variable pitch angle and radius. These variations allow the nonuniform helix to be for 2–3 dB more directive compared to the optimal uniform helix of the same length (750 mm). A 3D printed support structure for the helix to be wound around is made out of ABS plastic and a 400 mm by 400 mm metal sheet is used as the ground plane. The design frequency is 2 GHz, the impedance bandwidth is 33% (1.8 GHz–2.5 GHz), and the axial ratio is less than 3.8 dB. The gain is 18.2 dBi and the two prototypes that are built match the simulation performance.

[45] Baki Berkay Yilmaz, Robert Callan, Milos Prvulovic, and Alenka Zajic, “Quantifying Information Leakage in a Processor Caused by the Execution of Instructions,” Proceedings of IEEE MILCOM, October 2017.

Abstract: Covert/side channel attacks based on electromagnetic (EM) emanations are difficult to detect because they are practiced wirelessly. Hence, quantifying information leakage is crucial when designing secure hardware and software. To address this problem, this paper establishes a connection between the signal energy available to an attacker in electromagnetic side/covert channel and capacity of the covert/side channel. We first present a mathematical relationship between electromagnetic side-channel energy (ESE) of individual instructions and measured sidechannel signal power, assuming that all instructions have equal execution time. Then, we use this measure to calculate the transition probabilities needed for estimating capacity. Furthermore, we consider each instruction as a codeword and relate our model to Shannon’s capacity. Finally, we provide practical examples to demonstrate the severity of covert/side channel due to EM emanations.

[46] A. Nazari, N. Sehatbakhsh, M. Alam, A. Zajic, and M. Prvulovic, "EDDIE: EM-Based Detection of Deviations in Program Execution," Proceedings of the 44th International Symposium on Computer Architecture (ISCA), June2017. (acceptance rate 17 %)

Abstract: This paper describes EM-Based Detection of Deviations in Program Execution (EDDIE), a new method for detecting anomalies in program execution, such as malware and other code injections, without introducing any overheads, adding any hardware support, changing any software, or using any resources on the monitored system itself. Monitoring with EDDIE involves receiving electromagnetic (EM) emanations that are emitted as a side effect of execution on the monitored system, and it relies on spikes in the EM spectrum that are produced as a result of periodic (e.g. loop) activity in the monitored execution. During training, EDDIE characterizes normal execution behavior in terms of peaks in the EM spectrum that are observed at various points in the program execution, but it does not need any characterization of the malware or other code that might later be injected. During monitoring, EDDIE identifies peaks in the observed EM spectrum, and compares these peaks to those learned during training. Since EDDIE requires no resources on the monitored machine and no changes to the monitored software, it is especially well suited for security monitoring of embedded and IoT devices. We evaluate EDDIE on a real IoT system and in a cycle-accurate simulator, and find that even relatively brief injected bursts of activity (a few milliseconds) are detected by EDDIE with high accuracy, and that it also accurately detects when even a few instructions are injected into an existing loop within the application.

[47] N. Sehatbakshsh, R. Callan, M. Alam, M. Prvulovic, and A. Zajic, “Leveraging Electromagnetic Emanations for IoT Security, ”Hardware Demo at IEEE International Symposium on Hardware Oriented Security and Trust (HOST) May 1-5, 2017. –**Best Demo Award**

Abstract: We will demonstrate a new method to detect malware by externally observing Electromagnetic (EM) signals emitted by an IoT system. The proposed demo is an extension of the work in [1] and does not require any resources or infrastructure on, or any modifications to, the monitored system itself. Specifically, our method can identify malicious code injection into a known application that is running on an IoT device with >95% accuracy and with a detection latency <45 ms of executed code.

[48] A. Zajic, Milos Prvulovic, and Derrick Chu, “Path Loss Prediction for Electromagnetic Side-Channel Signals,” Proceedings of the 11th European Conference on Antennas and Propagation EUCap11, pp.1-5, Paris, France, March 2017.

Abstract: This paper investigates propagation mechanisms that EM side-channel signals experience at different frequencies and proposes models for near-field and far-field propagation of side-channel signals. The near-field propagation is modelled as a field created by an electric monopole (Hertzian dipole) and a magnetic dipole, where the received power is collected using only magnetic components of the EM field. This model resulted in excellent match with measured data. Furthermore, this paper investigates unintentionally modulated side-channel signals. The propagation of EM side-channel signals was modelled using freespace propagation model which resulted in excellent match with measured data. In both cases we have observed that signal can be received at several meters from the side-channel source. The proposed models are the first step in understanding propagation mechanisms of EM side-channel signals and how to predict the distance at which they can be received.

[49] N. Sehatbakhsh, A. Nazari, A. Zajić, and Milos Prvulovic, "Spectral Profiling: Observer-Effect-Free Profiling by Monitoring EM Emanations," IEEE MICRO 16, pp.1-11, Taipei, Taiwan, October 2016. (acceptance rate 20 %) – **The Best Paper Award**

Abstract: This paper presents Spectral Profiling, a new method for profiling program execution without instrumenting or otherwise affecting the profiled system. Spectral Profiling monitors EM emanations unintentionally produced by the profiled system, looking for spectral "spikes" produced by periodic program activity (e.g. loops). This allows Spectral Profiling to determine which parts of the program have executed at what time. By analyzing the frequency and shape of the spectral "spike", Spectral Profiling can obtain additional information such as the per-iteration execution time of a loop. The key advantage of Spectral Profiling is that it can monitor a system as-is, without program instrumentation, system activity, etc. associated with the profiling itself, i.e. it completely eliminates the "Observer's Effect" and allows profiling of programs whose execution is performance-dependent and/or programs that run on even the simplest embedded systems that have no resources or support for profiling. We evaluate the effectiveness of Spectral Profiling by applying it to several benchmarks from MiBench suite on a real system, and also on a cycle-accurate simulator. Our results confirm that Spectral Profiling yields useful information about the runtime behavior of a program, allowing Spectral Profiling to be used for profiling in systems where profiling infrastructure is not available, or where profiling overheads may perturb the results too much ("Observer's Effect").

[50] R. Callan, F. Behrang, M. Prvulovic, A. Zajic, and A. Orso, "Zero-Overhead Profiling via EM Emanations," accepted to The International Symposium on Software Testing and Analysis, 18-20 July 2016, Saarbrücken, Germany. (acceptance rate 25 %)

Abstract: This paper presents an approach for zero-overhead profiling (ZOP). ZOP accomplishes accurate program profiling with no modification to the program or system during profiling and no dedicated hardware features. To do so, ZOP records the electromagnetic (EM) emanations generated by computing systems during program execution and analyzes the recorded emanations to track a program's execution path and generate profiling information. Our approach consists of two main phases. In the training phase, ZOP instruments the program and runs it against a set of inputs to collect path timing information while simultaneously collecting waveforms for the EM emanations generated by the program. In the profiling phase, ZOP runs the original (i.e., uninstrumented and unmodified) program against inputs whose executions need to be profiled, records the waveforms produced by the program, and matches these waveforms with those collected during training to predict which parts of the code were exercised by the inputs and how often. We evaluated an implementation of ZOP on several benchmarks and our results show that ZOP can predict path profiling information for these benchmarks with greater than 94% accuracy on average.

LIST OF ACRONYMS, ABBREVIATIONS, AND SYMBOLS

ACRONYM	DESCRIPTION
LoS	Line of Sight
NLoS	Non Line of Sight
CAMELIA	Computational Activity Monitoring by Externally Leveraging Involuntary Analog Signals
SW	software
Sys	system
HW	hardware
SLL	Side-Lobe Level
EM	Electromagnetic
E	Electric
H	Magnetic
SNR	Signal to noise ratio
λ_0	wavelength
AM	Amplitude Modulation
FM	Frequency Modulation
SAVAT	Signal Available to Attacker
RC	Resistor-capacitor
CDF	Cumulative distribution function
MCS	Modulated Carrier Score
REMOTE	Robust External Malware Detection Framework by Using Electromagnetic Signals
STFT	short-time Fourier transform
SS	Spectral samples
FFT	Fast Fourier Transform
CAPE	Clock- Adjusted Energy and Peaks

ACRONYM	DESCRIPTION
CPS	Cyber-Physical Systems
HDBSCAN	Hierarchical Density-Based Spatial Clustering of Applications with Noise
FSM	Finite-State Machine
OS	Operating System
CNN	Convolutional neural network
MarCNNet	Markov and convolutional neural network
ReLU	Rectified Linear Unit
DC component	Direct Current component
TESLA	Tracing through Electromagnetic Side-channeL Analysis
CPU	Computer processing unit
ID	identification number
CFG	control-flow-graph
TSC	Time Stamp Counter
PITEM	Permutations-based Instruction Tracking via Electromagnetic Side-channel Signal Analysis
I/O	input/output
IoT	Internet-of-Things
DDoS	Distributed Denial-of-Service
LDL2/LDL1	loads from the on-chip L2 and L1 caches
PoP	Package on Package
SDR	software defined radio
LCD	Liquid-crystal display
AWGN	Additive white Gaussian noise