INFORMATION EMBEDDED IN UNINTENTIONAL ELECTROMAGNETIC EMANATIONS FROM COMPUTING DEVICES: IDENTIFICATION, QUANTIFICATION, AND EMERGING APPLICATIONS

A Thesis Proposal
Presented to
The Academic Faculty

by

Robert L. Callan

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
March 2016

INFORMATION EMBEDDED IN UNINTENTIONAL ELECTROMAGNETIC EMANATIONS FROM COMPUTING DEVICES: IDENTIFICATION, QUANTIFICATION, AND EMERGING APPLICATIONS

To be approved by:

Professor Moinuddin K. Qureshi,
Committee Chair
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Alenka Zajic, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Milos Prvulovic
School of Computer Science, College of Computing
Georgia Institute of Technology

Date Approved: TBD

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The objective of the proposed research is to identify, quantify, and use information leaked in Electromagnetic (EM) emanations for a broad range of computing devices in a general (i.e. not application specific) way by synthesizing techniques from the fields of electromagetics, computer architecture, and software engineering. Computers emit EM radiation (emanations) due to the voltage and current variations required to perform computation. The field of Electromagnetic Compatibility (EMC) systematically characterizes and analyses EM emanations across the frequency spectrum. EMC testing identifies and quantifies EM emanations to design and test computing systems to ensure the emissions don't interfere with communications signals. Therefore EMC ignores any information embedded in the emissions and treats all emanations as unwanted "noise" whose level must be minimized. Until recently, the study of information embedded (i.e. leaked) in this EM noise was limited to cryptanalysis. Cryptography researchers have developed application specific techniques that analyze EM emanations to extract secret cryptographic keys from simple computing devices as the devices performs encryption operations. These techniques are generally are ad-hoc and application specific, as the goal is to demonstrate and fix weaknesses in existing cryptographic hardware and software implementations. These weaknesses can often be found without thoroughly understanding the electromagnetic and computer architectural mechanisms for the EM emanations and information leakage.

In addition to the existing uses in cryptography, EM emanations provide other useful information about a system's operation. A number of emerging applications make use of EM emanations to extract new types of information from computing devices. For example, EM emanations can be used to determine or verify the execution

path through a program which can be used in applications such as program profiling, debugging, and malware detection. These new applications require a more general approach that can be rapidly and automatically applied to numerous and diverse arbitrary programs and computing devices. This approach requires automatic and systematic identification, quantification, and analysis of information embedded in EM emanations. The proposed research consists of four main contributions toward this goal. The first contribution, the *Signal AVailable to the ATtacker (SAVAT)* metric, measures the side channel signal created by a single-instruction difference in program execution and develops benchmarks which generate side channel signals caused by specific types of system activity. The second contribution, *Finding Amplitude-modulated Side-channel Emanations (FASE)*, uses the SAVAT benchmarks to identify and characterize existing strong periodic signals generated by computer systems which are modulated by specific system activities such as processor computation or memory accesses. The third contribution, *Zero-Overhead Profiling (ZOP)*, accurately profiles unmodified computer programs by predicting a program's execution path based on demodulated EM emanations alone. The final contribution, *Malware Detection on Internet of Things (IoT) Devices at a Distance*, applies the ZOP framework to verify control flow on IoT devices from distances of 30 cm to 3 meters.

CHAPTER I

INTRODUCTION

Previous research of the information embedded in EM emanations have focused almost exclusively on the studying how emanations can be used to compromise a device's security (i.e. side channel attacks). EM emanations have been used in a variety of side channel attacks for many different types of computing devices. However, as attacks are found system designers actively modify and improve systems to weaken and remove very specific types of information leakage (the leakage of cryptographic keys). This makes such work very application specific and focused on a perpetual cycle of developing attacks to exploit very specific vulnerabilities. Once vulnerabilities are demonstrated, system designers implement fixes to reduce information leakage. With each iteration of attack and countermeasure, the information leakage becomes weaker or harder to extract, making the attacks and countermeasures increasing sophisticated and application specific.

The existing methods used for side channel attacks are relatively effort intensive, requiring significant work to get the attack to work on a single device and single version of a single program or implementation. This effort is acceptable for side channel attacks because the potential reward is very high. However, recent research has demonstrated that EM emanations leak information about a very wide range of system activity and that this leaked information may be useful for many applications such as profiling, malware detection, and debugging. In these applications the monitored system is not hostile, making the extraction of useful information from EM emanations less difficult. On the other hand, these uses of EM emanations have lower reward per device or program. Therefore these applications will require systematic

and automatic identification, quantification and usage of EM emanations to allow techniques to be applied across many different types of devices and software with minimal additional effort.

Information leakage can be quantified at many levels of granularity, ranging from differences in emanations across phases of a program's execution down to information leakage caused by specific hardware components such as transistors. However, in order to identify specific leaking circuits or parts of a computer program, a level of granularity that simultaneously exposes the contributions of hardware and software, i.e. the instruction level. The first contribution of the proposed research, SAVAT, quantifies information leakage at the instruction level and develops benchmarks which can be used to quantify information leakage from specific instructions and system activities such as arithmetic operations or memory accesses.

Information leakage can be caused by many system components and occurs across the EM spectrum, and leakage signals are intermingled with noise created by other system components and signals from the external environment such as radio broadcasts, wireless communications, and power equipment. The strongest high frequency leakage signals are generated by system components that generate strong periodic signals (carriers) which are modulated by the information of interest. In order to effectively use or minimize the information embedded in these modulated EM emanations, it is necessary to determine system activities that modulate these carriers, determine the frequency range and strength of the leakage signals, and determine the modulation mechanism causing the leakage. The second contribution of the proposed research, FASE, presents a method for finding existing computer system signals that are amplitude modulated by specific types of system activity.

SAVAT and FASE automate the processing of finding and quantifying information leakage via EM emanations. To be viable, applications making use of EM emanations information must also be able to be automatically applied to arbitrary computer

programs running on the monitored computing devices. The third contribution of the proposed research, Zero-Overhead Profiling (ZOP) accomplishes whole program path profiling by profiling without instrumentation. ZOP requires no program-specific manual work because the program specific behavior is automatically characterized and used. The fourth contribution of the proposed research, *Malware Detection on Internet of Things (IoT) Devices at a Distance* demonstrates a second application which also automatically characterizes and uses EM emanations.

The remainder of this proposal is organized as follows: Chapter 2 describes in more detail the context provided by previous work and how this proposal fits into that context. Chapter **??** describes previous and current related research and makes clear how the proposed work relates to that research. Chapter 3 describes the contributions which have already been completed and Chapter 4 presents the work that remains to be completed.

CHAPTER II

BACKGROUND

The vast majority of research on information leakage via EM emanations has been devoted to developing specific side channel attacks and defenses against them. Side channel attacks circumvent traditional security protections and access controls. Unlike traditional attacks that exploit vulnerabilities in what the system does, side channel attacks access information by observing how the system does it. Computation generates many types of electronic and microarchitectural activity. Side channel attacks identify some physical or microarchitectural signal (i.e. the side channel signal) that leaks desired information about system activity or the data being processed, and then analyze that signal as the system operates. Much work has been done to prevent particular side channel attacks, either by severing the tie between sensitive information and the side channel signal, or by trying to make the signal more difficult to measure. However such work mostly focuses on preventing a particular side channel attack in a very specific piece of code, such as a cryptographic kernel.

The prototypical side channel attack is Differential Power Analysis (DPA). DPA is a side channel attack carried out on a device's power signal to extract the secret key used for encryption in algorithms such as the Advanced Encryption Standard. Like almost all side channel attacks, DPA treats the computing system as a black box where the observed emanations are a direct yet unknown function of the secret key bits. Therefore DPA is suited for directly relating observed emanations to a relatively small number of secret bits where the relationship between the emanations and secret bits is relatively simple but the leakage signal may be very weak due to countermeasures. Specifically, DPA first measures power consumption traces over

many encryption operations with many different known key values. Then for a single bit of the key all the traces for keys with this bit set to 0 are averaged together. Similarly all traces for keys with this bit set to 1 are averaged together. The trace average for the key bit set to 0 is subtracted from the trace average with the key bit set to 1. This difference signal exposes the times in the trace where a higher value indicates a likely value of 1 for the key bit. Then more power consumption traces are measured with a single unknown key value and each key bit is guessed by observing the signal level at the time positions with large differences between when average of all traces where the key bit equals 0 and the average signal where the key bit equals 1.

Side channel attack techniques such as DPA have a number of drawbacks which make them poorly suited for applications beyond cryptanalysis applications and devices. These new applications differ from the typical attack model for cryptanalysis in several ways. First, the reward for a single attack against a single device is much higher for side channel attacks vs other applications. Second, the system designers employ countermeasures that weaken the signal, increase noise, and weaken the link between the emanations and leaked information. Third, the structure of information needed for other applications is more complex and varies from problem instance to problem instance. Fourth, analyses for new applications must be easily automatically applied across a wider variety of devices, software types, and information to be extracted.

These differences require a different approach from side channel attacks. While a very application specific and effort intensive approach makes sense for side channel attacks, other applications can take advantage of the stronger (unguarded) signals but also must systematically characterize hardware and software differences between problem instances and automatically carry out many of the steps which could be done manually during side channel attacks. For example, it is necessary to automatically

identify the leakge of specific circuits or instructions using SAVAT, to find which signals are modulated by useful information using FASE, and to automatically determine the structure of a program to be profiled and make the connection between this structure and the observed EM emanations as done by ZOP.

CHAPTER III

COMPLETED WORK

3.1 A Practical Methodology for Measuring the Side-Channel Signal Available to the Attacker for Instruction-Level Events [1]

This paper presents a new metric, which we call Signal Available to Attacker (SAVAT), that measures the side channel signal created by a specific single-instruction difference in program execution. SAVAT quantifies the signal made available to a potential attacker who wishes to decide whether the program has executed instruction/event A or instruction/event B. Computation requires a large variety of electronic and microarchitectural activities and these activities can have a number of observable side effects such as EM, acoustic, or thermal emanations. Much of previous research attempts to prevent particular side channel attacks, either by severing the tie between sensitive information and the side channel signal, or by trying to make the signal more difficult to observe. Quantifying side channel exposure in general has not been well studied, and when it has been studied, the measurements use either very coarse or very fine-grain levels of granularity, such as at the level of program phases or at the transistor level respectively. SAVAT prevents an approach somewhere in between these two approaches by measuring the contribution to the side channel signal of a single instruction difference in program execution. This approach is useful to computer architects because it tells which microarchtectural features create strong leakage signals and is useful to software developers since the leakage of individual instructions becomes observable to them and so the single instruction differences can be systematically aggregated across parts of a program.

This work presents a practical methodology for measuring SAVAT in real systems

using only user-level access permissions and common measurement equipment. SAVAT is defined as a pairwise metric between a pair of instructions A and B. SAVAT is the difference in side channel signal caused by executing A once instead of B. A simplistic approach might measure the side channel signal while executing a section of code containing one A instruction, and then separately measure the same section of code again, substituting the B instruction for the A instruction. This approach has a number of drawbacks when applied to complex processors due to variations in execution timing, limitations of measurement equipment, and the very small effect of a single instruction difference. Instead, SAVAT executes a section of code containing the A instruction, then executes the same section with the B instruction, and repeats alternating between A and B for the duration of the measurement. This is accomplished by running a benchmark similar to that shown in Figure 1. This benchmark creates a periodic signal as shown on the right of Figure 3 at the frequency corresponding to the period of the outer loop of the benchmark.

For this paper, several x86 instructions were selected, and load and store instructions were measured with different address access patterns to separately measure the effect of accessing different portions of the memory hierarchy as shown in Figure 2. These different cache behaviors were accomplished in the benchmarks by updating the address of the accessed memory location so that the memory access repeatedly sweeps over an array of appropriate size with an appropriate stride (in lines 4 and 10 of Figure 1), such that the array fits in L1 cache, or does not fit in L1 but fits in L2 cache, or does not fit in the L2 cache.

These benchmarks were used to perform a case study where we measure electromagnetic (EM) emanations SAVAT among the 11 different instructions in Figure 2 for three different laptop systems. Since there are 11 instructions, this results in an 11 by 11 table of SAVAT values as shown in Figure 4. The findings from these experiments confirm key intuitive expectations, for example that SAVAT between on-chip

```
1       while(1){
2         // Do some instances of the A inst/event
3         for(i=0;i<inst_loop_count;i++){
4           ptr1=(ptr1&~mask)|((ptr1+offset)&mask);
5           // The A-instruction, e.g. a load
6           value=*ptr1;
7         }
8         // Do some instances of the B inst/event
9         for(i=0;i<inst_loop_count;i++){
10          ptr2=(ptr2&~mask)|((ptr2+offset)&mask);
11          // The B-instruction, e.g. a store
12          *ptr2=value;
13        }
14      }
```

Figure 1: The A/B alternation pseudocode.

	Instruction	**Description**
LDM	mov eax,[esi]	Load from main memory
STM	mov [esi],0xFFFFFFFF	Store to main memory
LDL2	mov eax,[esi]	Load from L2 cache
STL2	mov [esi],0xFFFFFFFF	Store to L2 cache
LDL1	mov eax,[esi]	Load from L1 cache
STL1	mov [esi],0xFFFFFFFF	Store to L1 cache
ADD	add eax,173	Add imm to reg
SUB	sub eax,173	Sub imm from reg
MUL	imul eax,173	Integer multiplication
DIV	idiv eax	Integer division
NOI		No instruction

Figure 2: x86 instructions for our A/B SAVAT measurements.

instructions and off-chip memory accesses tends to be higher than between two on-chip instructions. However, findings show that particular instructions, such as integer divide, have much higher SAVAT than other instructions in the same general category (integer arithmetic), and that last-level-cache hits and misses have similar (high) SAVAT. Overall, we confirm that our new metric and methodology can help discover the most vulnerable aspects of a processor architecture or a program, and thus inform decision-making about how to best manage the overall side channel vulnerability of a processor, a program, or a system. The results of these measurements can be used by circuit designers and microarchitects to reduce susceptibility to side channel attacks by focusing on high-SAVAT aspects of their designs (e.g. off-chip memory accesses,

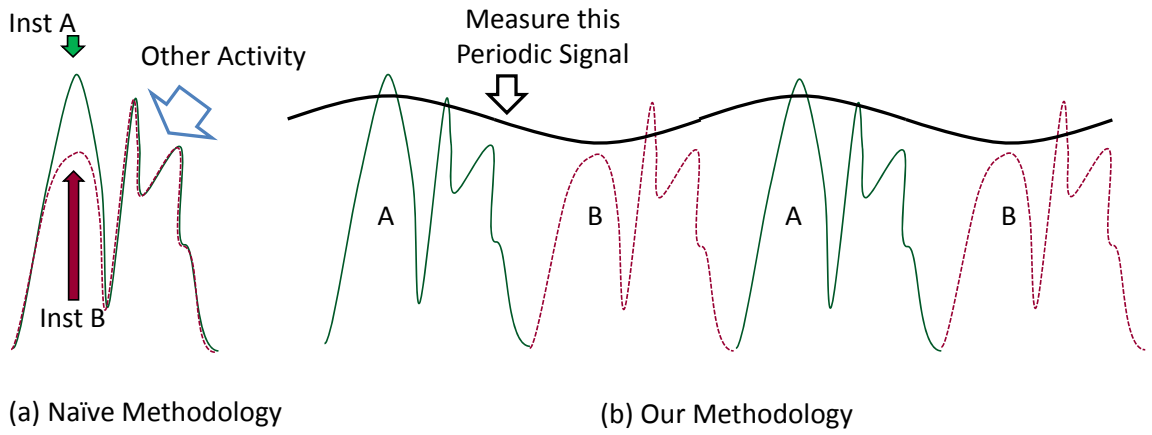

Figure 3: SAVAT measures the (a) signal difference by (b) alternating the signals then filtering and measuring the resulting periodic signal at the alternation frequency.

	LDM	STM	LDL2	STL2	LDL1	STL1	NOI	ADD	SUB	MUL	DIV
LDM	1.8	2.4	7.9	11.5	4.6	4.4	4.3	4.2	4.4	4.2	5.1
STM	2.3	2.4	8.8	11.8	4.3	4.2	3.8	3.9	3.9	4.3	4.2
LDL2	7.7	7.7	0.6	0.8	3.9	3.5	4.3	3.6	4.8	3.8	6.2
STL2	11.5	10.6	0.8	0.7	5.1	6.1	6.1	6.1	6.1	6.2	10.1
LDL1	4.4	4.2	3.3	5.8	0.7	0.6	0.7	0.7	0.7	0.7	1.3
STL1	4.5	4.2	3.8	4.9	0.7	0.6	0.7	0.6	0.6	0.6	1.2
NOI	4.1	3.8	4.1	6.4	0.7	0.7	0.6	0.6	0.7	0.6	1.0
ADD	4.2	4.1	4.1	7.0	0.7	0.7	0.6	0.7	0.6	0.6	1.0
SUB	4.4	4.0	3.8	7.3	0.7	0.6	0.7	0.6	0.6	0.6	1.1
MUL	4.4	3.9	3.7	5.7	0.7	0.7	0.6	0.6	0.6	0.6	1.1
DIV	5.0	4.6	6.9	9.3	1.3	1.2	1.0	1.1	1.1	1.1	0.8

Figure 4: SAVAT values (in zJ) for a Core 2 Duo laptop at 10 cm and at the 80 kHz alternation frequency.

last-level-cache hits, and possibly the integer divider in the systems we measured). Programmers, compilers, and algorithm designers can also use SAVAT to guide code changes to avoid using "loud" activity when operating on sensitive data. Although we only present results for the EM side channel, we believe that our overall methodology can be used to characterize the relationship between instruction-difference and signal-difference in other side channels, especially acoustic and power-consumption side channels where instruments are readily available to measure the power of the periodic signals created by our methodology.

3.2 FASE: Finding Amplitude-modulated Side-channel Emanations [2]

This paper presents a methodology for rapidly finding and characterizing amplitude modulated side-channel emanations. Computing devices generate many types of EM signals, but generally the strongest and farthest propagating side channel signals occur when sensitive information can be related to a system activity that modulates an existing strong, periodic signal (such as a clock driving a large circuit). In order to use this information or develop countermeasures, such a signal must first be identified and the signal's root cause must be found. This process is difficult because computing devices generate emanations at thousands of different frequencies and only a small number of these emanations are modulated and carry useful information. Many side channel attack descriptions only briefly or implicitly address the underlying mechanisms that cause information leakage because finding information carrying signals and determining their causes are separate processes, and because secret information can be extracted without knowing what causes the information leakage.

FASE uses the SAVAT benchmarks to create recognizable spectral patterns and then processes recorded spectra to identify signals that exhibit amplitude-modulation behavior. SAVAT benchmarks were originally used to characterize the difference between individual instructions by observing the signal created at the predictable and known alternation frequency, known as the "baseband" signal. In contrast, FASE uses the same benchmarks with different parameters to identify carrier signals generated by computing devices that are modulated by signals that carry leaked information. The frequency and sources of these carrier signals are difficult or impossible to determine with existing techniques. The EM spectrum is full of amplitude-modulated signals (e.g. radio broadcasts) that are not modulated by program activity with complicates finding carriers generated by system activity. FASE filters out such external unrelated signals by generating several different activity patterns and reporting only

those signals which are specifically modulated by these system activities.

The carriers generated by computing devices can be particularly difficult to identify compared to communications signals for several reasons. First, the computing devices generate numerous other signals and noise which can overlap with the modulated carriers. Second, carriers from computing devices are not designed or intended to be received or analyzed. The maximum strength of these carriers (i.e. noise level in EMC) is regulated by EMC standards, so system designers apply techniques to strong carriers to spread the radiated energy over a range of frequencies. For clock signals this is often done using spread spectrum clocking and for voltage regulators this is usually accomplished by using an RC oscillator that has large frequency variations. These characteristics spread the signal energy of the carrier, making the signal weaker, but also making it harder to detect that a given signal is being modulated by system activity since the carrier's shape in the frequency domain gets convolved with the modulation signal, spreading the sideband carrying the leaked information.

The SAVAT benchmarks generate repetitive changes in processor and memory activity which appear in the baseband spectrum at the alternation frequency as a narrow large peak. FASE measures the spectrum over a wide frequency range while the benchmarks are running, and then processes the resulting spectra to find spectral patterns corresponding to amplitude modulation. Each benchmark generates a baseband signal at the alternation frequency f_{alt} which corresponds to the time required for one iteration of the outer loop in Figure 1. If the activity exercised by the benchmark modulates a particular carrier at a frequency f_c, then this baseband signal will become modulated onto the carrier, and will create a peak at $f_c \pm f_{alt}$. If the spectrum was observed with no code running, only the carrier at f_c would be visible. If the spectrum was also observed with code running and a peak occurs at $f_c \pm f_{alt}$, we would have some confidence that a carrier exists at f_c. However, this technique alone is often not reliable enough for several reasons. First, the sidebands

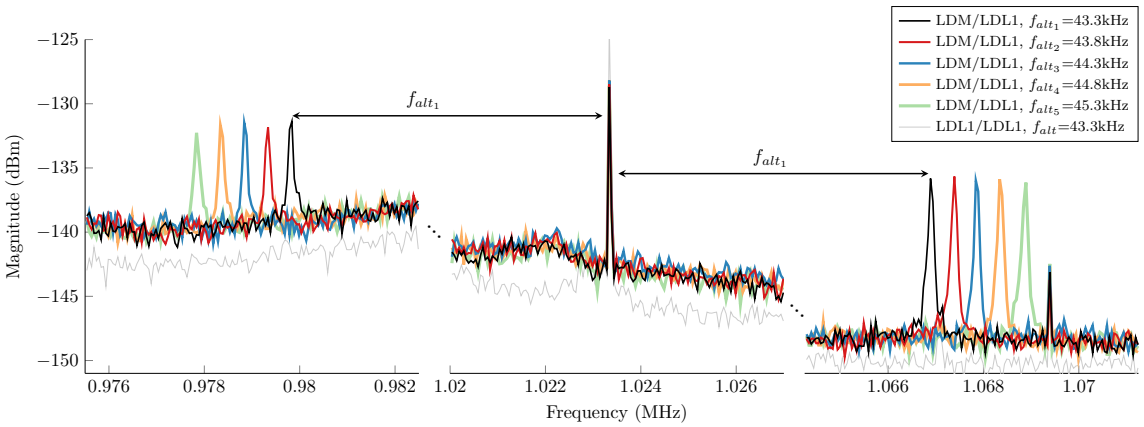

Figure 5: A carrier at f_c and its right and left side-bands generated by memory activity.

(signal at $f_c \pm f_{alt}$) can be obscured by stronger unrelated signals. Second, the carriers and baseband signals are almost always not pure sinusoids and so have strong harmonics. Therefore peaks are also created at frequencies $Nf_c \pm Mf_{alt}$ for integers N and M. Several such signals can occur near each other creating a spectral pattern that matches a true $f_c \pm f_{alt}$ modulation pattern. To create a more reliable detection procedure the benchmarks are run at several different f_{alt} frequencies and the resulting spectra are overlayed. For a strong, obvious carrier, these overlayed spectra will look similar to Figure 5. A more obscured, spread out carrier and its sidebands are shown in Figure 6.

In both Figure 5 and Figure 6, for f_{alt_1} =43.3kHz, a peak appears 43.3kHz to the right and to the left of the carrier in the center of the figure. The spacing between f_{alt_1} and f_{alt_2} is 500 Hz, which is the same spacing between all f_{alt_2} and f_{alt_3} and so on. This results in a set of peaks in each sideband separated by 500 Hz, with one peak occuring in each spectra at the alternation frequency. Therefore, to find modulated carriers we look for this pattern (one peak in each spectra separated by 500 Hz) and then calculate the position of the carrier from the known f_{alt_1}.

We apply this method to several computer systems and find several such modulated signals. To illustrate how our methodology can benefit side-channel security

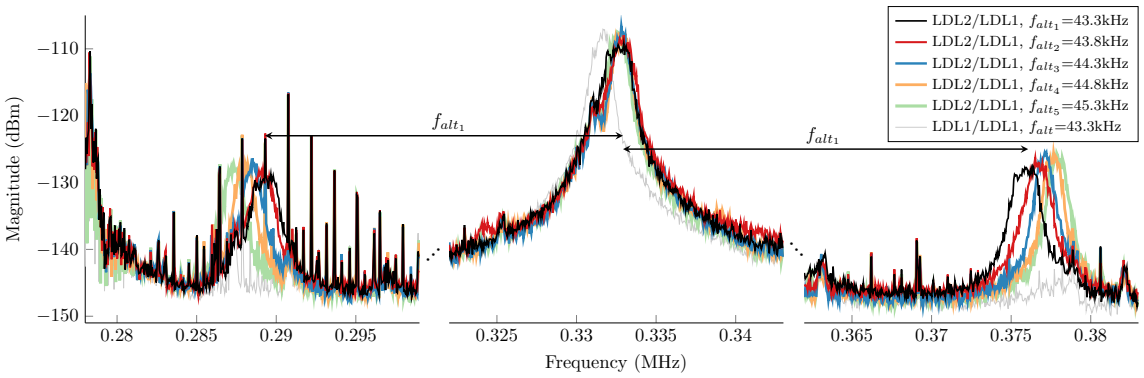

Figure 6: A carrier at f_c and its right and left side-bands generated by L2 cache activity.

research and practice, we also identify the physical mechanisms behind those signals, and find that the strongest signals are created by voltage regulators, memory refreshes, and DRAM clocks. The benchmarks expose several properties of the modulation signal and carrier which are useful for identifying the source of the signal. For example, the sideband signals in Figure 5 are only present for benchmarks which access the DRAM memory, and the carrier and signals have very narrow peaks, implying the clock source is well controlled, such as is the the case for digital logic clocks. Figure 5 is actually the 8th harmonic of a signal with a fundamental frequency of 128kHz. Also, more interestingly, experimentation showed that the carrier was strongest when there were no memory accesses, and weakest when the memory was accessed contantly. This is the opposite of what might be expected, since at first it seems the signal gets weaker as the circuit is used more frequently. However, from these clues it was determined that the signal is caused by the periodic DRAM memory refresh. The signal is weaker as the memory is accessed more frequently because as the memory is accessed more frequently, the timing of the refresh signals becomes more randomized, weakening the signal. The signal in Figure 6 was determined to be caused by voltage regulators supplying power to the processor using similar analysis and techniques. At higher frequencies, FASE discovered clock signals

and their harmonics that are modulated by activity in the clock's domain. Because most high frequency clock signals are subject to electromagnetic interference (EMI) regulations [3], they are subjected to measures (such as spread-spectrum clocking) that spread the resulting EM emanations over a range of frequencies [4]. In spite of this, FASE discovers such signals and provides insight into the nature of the activity that modulates them. In particular, FASE identified that DRAM clocks generate EM emanations which are modulated by DRAM activity. The systems tested also generated weak spread-spectrum signals at the CPU clock frequency. The FASE results indicate that each signal may carry unique information about system activity, potentially enhancing an attacker's capability to extract sensitive information. FASE correctly separates emanated signals that are affected by specific processor or memory activities from those that are not. FASE can be used to find which parts of a system leak information about some aspect of program activity. This can be used to reduce the strength of modulated signals and to weaken their modulation, i.e. disrupt the connection between program behavior and the variations in activity that modulate such signals. Using memory refresh signals as an example, this would involve randomization of the interval between refresh commands, while modulation-weakening efforts might involve careful scheduling of memory accesses to avoid their interaction with refresh activity.

3.3 ZOP: Zero-Overhead Profiling via EM Emanations

This paper presents zero-overhead profiling (ZOP). Path profiling is one of the most common types of program profiling. It consists of counting the executions of particular basic blocks and using these counts to identify heavily executed paths (also called *hot paths*). Knowledge of the most frequently executed paths can be used in code optimization and other types of performance analysis. Profiling techniques employ code instrumentation to count the executions of particular basic blocks. To

do this, profiling tools add software probes (instrumentation points) to the program to be profiled. These instrumentation points either log events of interest or update statistics about such events at runtime. This approach can produce nearly perfect program profiles, but introduce some runtime and memory overhead during profiling. This overhead is undesirable and may be unacceptable for some applications, such as in embedded systems with limited resources, in real-time systems, and in deployed systems.

ZOP has two main phases: training and profiling. ZOP uses instrumentation only during the training phase. After generating a training model for a program, ZOP can profile this program with any arbitrary inputs. During profiling, ZOP uses no program modifications and no instrumentation and therefore introduces zero runtime and memory overheads. The trade-off for zero overhead is a potential loss of accuracy in the profiling results compared to instrumentation based approaches, though our results show this loss in accuracy is small. ZOP's average accuracy is 94% of these other approaches.

Instead of using instrumentation during profiling, ZOP relies on the electromagnetic (EM) emanations generated by computing systems and leverages them to track a program's execution path and generate profiling information. As shown by FASE's results, computing devices often have strong periodic signals (such as processor clock signals) whose amplitude varies depending on what the device is doing. When these signals are demodulated, they can provide a signal which carries reliable information about which parts of the code are executed, how often they are executed, and how much time is spent in each part of the code. This signal does contain noise and variations, but one of the main contributions of ZOP is that in spite of the noise and variation, ZOP can use this demodulated signal to accurately reconstruct the execution path through a program, a result is which is not intuitively obvious and which refutes the misconception that the EM emanations from computing systems

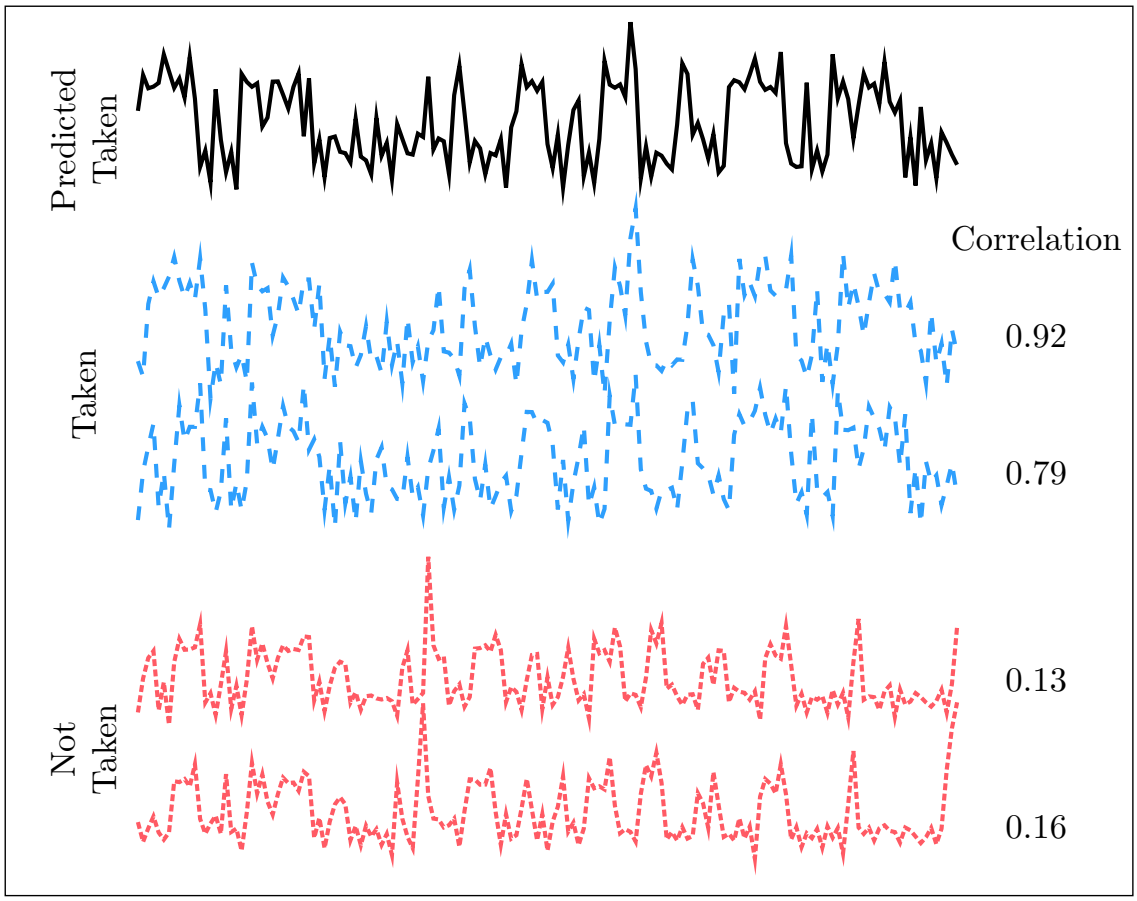

Figure 7: Examples of waveforms collected by measuring EM emanations produced by several executions.

are purely (or even mostly) noise.

Figure 7 shows several waveforms that were recorded for a short portion of a program's execution. All of these waveforms start at the same static location in the program, and each follows one of two paths, depending on whether the *true* or the *false* path of a conditional statement is followed. In particular, the dashed waveforms correspond to execution along the *true* (conditional branch instruction is "taken") path, whereas the two dotted waveforms correspond to execution along the *false* path (branch instruction is "not taken"). These waveforms are recorded during the training phase. During training instrumentation records the branch outcome so these waveforms can be correctly labeled. Observe that while there are some differences between waveforms that correspond to the same path, these differences are much

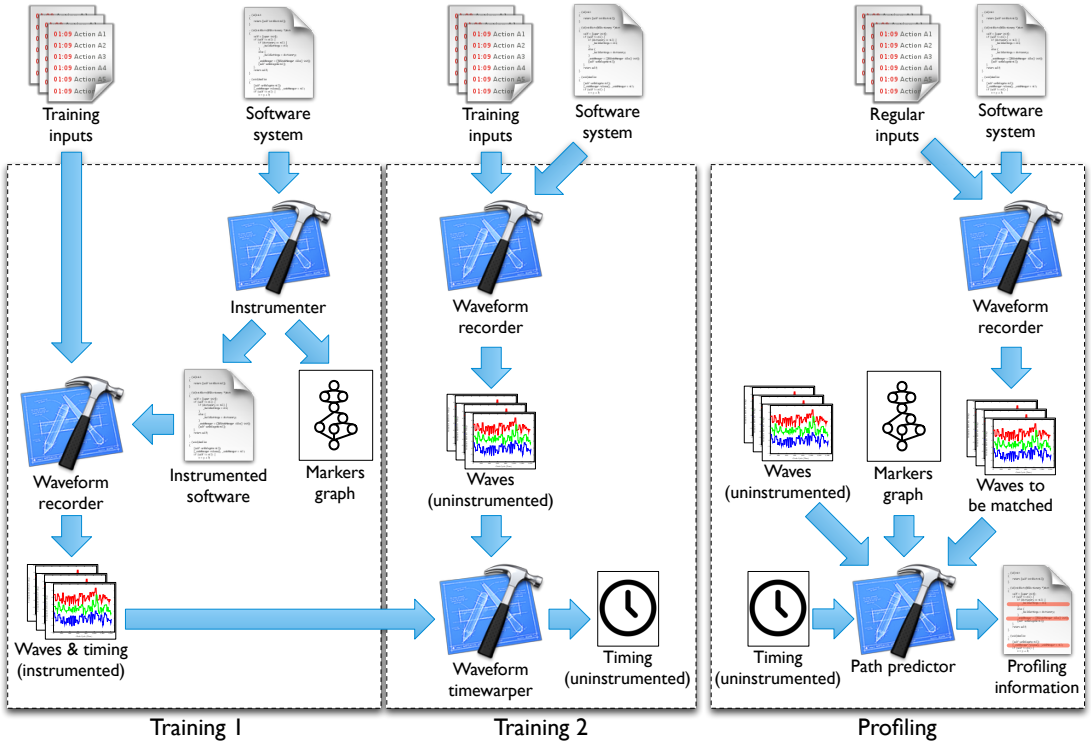

Figure 8: Workflow of ZOP.

smaller than those between the *true* and *false* paths. The training example with the highest correlation to the unknown execution is very likely to have taken the same path as the unknown execution, so for this example it can be assumed the branch was taken, since the solid waveform is more similar to the dashed waveforms than to the dotted waveforms.

Figure 8 shows a workflow of the ZOP system. In the first 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 second training phase, ZOP runs the uninstrumented program with same inputs, and matches the times in the waveforms from the first training phase to the waveforms from the second training phase. Then the recorded timing information from the first training phase (i.e. a record of which code was executing at which times in the waveform) is applied to the uninstrumented waveform.

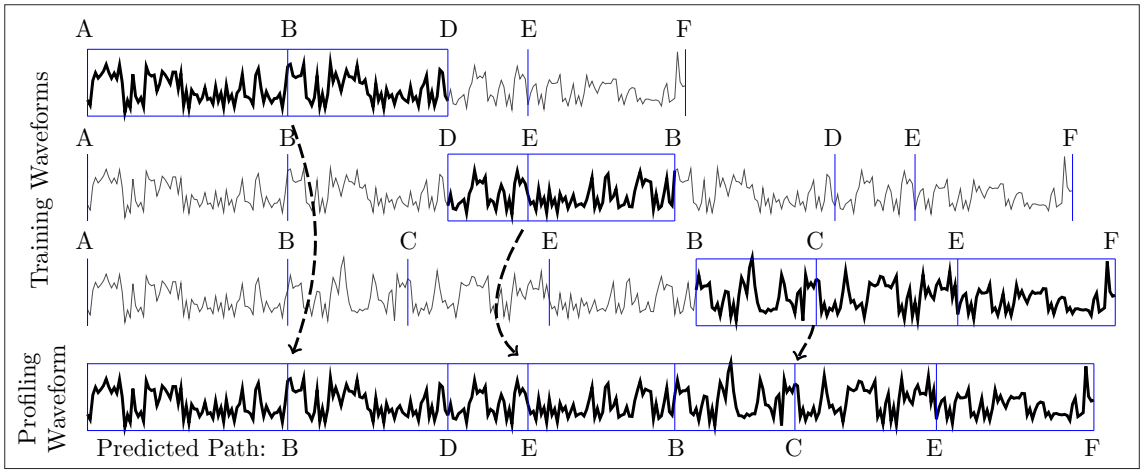

Figure 9: Predicting an example execution path by matching waveform segments of three training executions to an unknown execution waveform.

During the profiling phase, the program is run with new inputs for which profiling information is desired. A simplified example illustrating how the waveform and timing information collected during training is used to generate profiling information for a new input is shown in Figure 9. In this figure, the marked letters above the training waveforms (e.g. A,B,C, etc.) correspond to the position in the source code at that point in time in the waveform (this information is collected during the first phase). To predict the execution path taken for an example execution about which there is no apriori information, portions of the profiling waveform are matched to portions of the training waveform. For example, by comparing the beginning of the profiling waveform to the training waveforms, it is clear that the training waveforms for the ABD code path more closely match the profiling waveform than the ABC path does, so the ABD path is predicted as the correct path for the start of this example. This procedure can be applied iteratively to determine the path through the program by selecting segments of the training execution whose waveform matches the profiled execution waveform.

ZOP was implemented and tested for several benchmark programs. To determine the average accuracy across all the inputs tested for each benchmark, perfect (correct)

profiling information for each benchmark and each input in the profiling set was calculated. Then ZOP was used to generate predicted profiling information for the same set of inputs. The number of times each acyclic path was actually executed (the correct or ground truth count) was then compared to the number of times ZOP predicted that path was executed. The average accuracy for each benchmark was calculated using the following equations:

$$\text{accuracy} = \frac{\sum_{i=1}^{n} g_i a_i}{\sum_{i=1}^{n} g_i}$$

where

$$n = \text{number of acyclic paths per benchmark.}$$

$$g_i = \text{actual (ground truth) number executions of acyclic path } i.$$

$$z_i = \text{ZOP (predicted) number of executions of acyclic path } i.$$

$$a_i = \min\left(\frac{g_i}{z_i}, \frac{z_i}{g_i}\right) = \text{ accuracy for acyclic path } i.$$

The results showed that ZOP can predict path profiling information for the tested benchmarks with an average accuracy above 94%.

CHAPTER IV

REMAINING WORK

4.1 Malware Detection on Internet of Things Devices at a Distance

One of the emerging applications that makes use of the information embedded in EM emanations is the detection of malware and more generally the verification of control flow through a program. A rapidly growing number of embedded devices with internet connectivity are used in consumer electronics, Internet of Things devices, as well as industrial control and data acquisition applications. Securing these devices presents new and unique challenges due to their very limited software and hardware resources, the difficulty of applying software updates in the field, and the lack of standardization across the many hardware and software platforms used across devices. Techniques that make use of EM emanations to secure these devices make be a good fit for this application because such techniques do not require intrusive access or modification to the devices, and because remotely monitoring the devices in an air-gapped manner makes it impossible for an attacker to override security measures once the device is compromised. This work will adapt the ZOP approach to this security application. ZOP predicts the control flow through entire programs based on training data from previous program executions. Therefore when new previously unseen code (malware) runs on the device, ZOP can be used to detect that the new EM emanations do not match previous known behavior. While ZOP profiled devices using a small EM probe very close to the device being profiled, this work will also explore monitoring EM emanations at distances required to make this application viable (30 cm to 3 meters).

REFERENCES

[1] Callan, R., Zajic, A., and Prvulovic, M., "A Practical Methodology for Measuring the Side-Channel Signal Available to the Attacker for Instruction-Level Events," in *Proceedings of the 47th International Symposium on Microarchitecture (MICRO)*, 2014.

[2] Callan, R., Zajic, A., and Prvulovic, M., "FASE: Finding Amplitude-modulated Side-channel Emanations," in *Proceedings of the 42nd ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*, pp. 592–603, 2015.

[3] Erickson, R. and Maksimovic, D., *Fundamentals of Power Electronics*. Springer, 2001.

[4] Hardin, K. B., Fessler, J. T., and Bush, D. R., "Spread spectrum clock generation for the reduction of radiated emissions," in *Electromagnetic Compatibility, 1994. Symposium Record. Compatibility in the Loop., IEEE International Symposium on*, pp. 227–231, IEEE, 1994.