Power-based Side-Channel Instruction-level Disassembler

Jungmin Park, Xiaolin Xu, Yier Jin, Domenic Forte and Mark Tehranipoor

FICS Research, University of Florida
Gainesville, FL 32611, USA
jungminpark@ufl.edu,{xiaolinxu,yier.jin,dforte,tehranipoor}@ece.ufl.edu

ABSTRACT

Modern embedded computing devices are vulnerable against malware and software piracy due to insufficient security scrutiny and the complications of continuous patching. To detect malicious activity as well as protecting the integrity of executable software, it is necessary to monitor the operation of such devices. In this paper, we propose a disassembler based on power-based side-channel to analyze the real-time operation of embedded systems at instruction-level granularity. The proposed disassembler obtains templates from an original device (e.g., IoT home security system, smart thermostat, etc.) and utilizes machine learning algorithms to uniquely identify instructions executed on the device. The feature selection using Kullback-Leibler (KL) divergence and the dimensional reduction using PCA in the time-frequency domain are proposed to increase the identification accuracy. Moreover, a hierarchical classification framework is proposed to reduce the computational complexity associated with large instruction sets. In addition, covariate shifts caused by different environmental measurements and device-to-device variations are minimized by our covariate shift adaptation technique. We implement this disassembler on an AVR 8-bit microcontroller. Experimental results demonstrate that our proposed disassembler can recognize test instructions including register names with a success rate no lower than 99.03% with quadratic discriminant analysis (QDA).

KEYWORDS

Power side-channel, instruction level disassembly, instruction set architecture, embedded processors

1 INTRODUCTION

Thanks to the Internet of Things (IoT) and modern embedded computing devices, our world is now more automated and connected than ever before. However, such devices have become a regular target for many attacks. In a recent attack, IoT devices were targeted by Mirai malware to be used as part of a botnet in distributed denial-of-service (DDoS) attacks [2]. Another attack that is of concern when a device is physically accessed by the attacker is software theft or piracy [17]. This usually means unauthorized copying, either by individuals for their own use or by companies who then sell the illegal copies to users. Hence, reverse engineering of software [3, 6] for piracy or copyright analysis is a common practice in industry. Reverse engineering is a process that takes a software program's binary code and recreates it so as to trace it back to the original source code. That is, a security engineer performs reverse engineering to verify that the software running on a competitor's device is not a copy of his own genuine software.

Unfortunately, most embedded devices are incompatible with conventional software-based malware detection mechanisms such as antivirus software. Hardware-based malware detection using low-level architectural events such as number of memory accesses and immediate branches requires modification of embedded devices [19]. Reverse-engineering of protected firmware or software for copyright litigation is very difficult since the software is stored in the secure memory [11]. In order for a competitor to conceal software intellectual property (IP) piracy, code and data can be encrypted and then stored in the tamper-resistant memory.

A promising disassembly approach that does not require any changes to the design of an embedded computing device would be based on measurement of side-channel signals. This method could rely on an external monitor, which passively observes the device behavior without being directly connected to its I/O; thus avoiding bypass by malware and other possible malicious modifications. The disassembler can check whether a malicious code is inserted in the IoT device by tracking the executing instructions and comparing them with the original instruction flow. While reverse-engineering of a protected software is extremely difficult, the side-channel disassembler, however can recognize the behavior of decrypted code and detect software IP piracy. Further, since side channel analysis is nondestructive, it is the perfect candidate for reverse engineering the software of legacy/obsolete systems, which are scarce by nature. Among all side channel signals, power side-channel analysis provides a great candidate since it is already a well-studied approach for key exfiltration in crypto hardware and only requires access through a power pin.

Instruction-level reverse engineering solely through side-channel is still relatively uncommon especially compared with side-channel attacks [14]. Both exploit side-channel signals such as power [13] or electromagnetic emanation [1] and require physical access, but the goal of side-channel attack (SCA) is different. Its objective is to only extract the secret key. Disassembly is more difficult than side-channel attacks (SCA) for the following reasons. An instruction from the program executes only once per execution path. Hence, information to identify the instruction may be limited compared to side-channel attacks which have the luxury of repeating thousands of experiments with the secret key. Due to the limited information available for disassembly, covariate shift problem [24] caused by power measurement at different times or devices makes the identification even more challenging . Moreover, the instruction-level classifier or distinguisher only has as much time to classify as the processor's throughput for real-time malware detection. For instance, if a processor executes 4 instructions every clock cycle at 1 GHz, the distinguisher only has approximately $0.25(= 1/4)$ nano-seconds per instruction.

A few research works have used side-channel traces for the disassembly of instructions executing in a device. Statistical techniques such as Bayesian classifiers can be used to construct classification models from the known power consumption traces. Eisenbarth *et al.* [9] achieved a recognition rate of 70.1% on 35 test instructions and 50.8% on real code by applying a hidden Markov model. Msgna *et al.* [18] accomplished 100% recognition rate on a chosen set of 39 instructions in ATMega163-based smart cards running at a clock frequency of $4MHz$. They classified the power traces by applying PCA in combination with the $k(= 1)$-nearest neighbors (kNN) algorithm. The side-channel disassembler of Strobel *at al.* [23] has a recognition rate 96.24% on test code and 87.69% on real code on a PIC16F687 using multiple EM channels (antennas) with a decapsulated package. The work described in [18] and [23] use only instruction classes as distinguishing features to classify power traces. More recently, McCann *et al.* [16] have used instruction-level power models to spot even subtle leakage in implementations. This is unlike the traditional power side-channel models to estimate secret data, where only data specific switching is modeled.

DAC '18, June 24–29, 2018, San Francisco, CA, USA

ACM ISBN 978-1-4503-5700-5/18/06…$15.00
https://doi.org/10.1145/3195970.3196094

Table 1: Comparison of existing side-channel disassembly.

	[9]	[18]	[23]	[16]	Ours
Target μ-controller	PIC16F687	ATMega 163	PIC16F687	ARM Cortex-M0	ATMega 328P
Clock	1 MHz	4 MHz	4 MHz	8 MHz	**16 MHz**
# of Instructions	33 and 0 Regs.	39 and 0 Regs.	33 and 0 Regs.	Emulating leakage	**112 Insts. and 64 Regs.**
Recognition rate	70.1 %	100 %	96.24 %	–	99.03 %
Reduction	PCA, Fisher's LDA	PCA	Polychotomous LDA	–	PCA, KL-divergence
Classifier	Multivariate Gaussian	kNN	kNN	Linear regression	LDA, QDA, SVM, Naïve
Side channel	Power	Power	Multiple EM	Power	Power
CSA[1]	No	No	No	No	**Yes**

[1] Covariate Shift Adaptation

Table 1 shows the comparison of existing side-channel disassembly and our proposed disassembler. Existing solutions suffer from the following shortcomings: (1) the small number of instruction classes to recognize makes applicability of existing disassemblers limited. The existing methods are not able to recognize operands such as address of registers, making the reverse-engineering incomplete. (2) the target devices are running at low clock frequency. Disassembling these devices are easier than those with the higher clock-frequency since the higher the frequency, the more difficult signal acquisition would be and consequently more noise to handle during analysis [10]. In order to reverse-engineer instructions running on the high-frequency devices, specifically for run-time detection of malware on critical systems, more advanced signal processing and machine learning techniques are required.

In this paper, we present a power side-channel based disassembler, implemented on an AVR micro-controller [15]. This disassembler can be used for effective malware detection and reverse engineering. We make the following contributions in this paper:

- A hierarchical framework is developed to classify significantly more instructions (over 100 instructions in case of AVR) as well as the address of the source register (Rr) and the destination register (Rd) without statistical control flow analysis (e.g., using Markov chains) for reverse engineering of unknown software in real-time.
- New feature selection and dimensionality reduction using Kullback-Leibler divergence and PCA in the *time-frequency domain* are proposed and various machine learning classifiers are compared. In addition, a majority vote method is taken into account in case of multiple classes to further support real-time malware detection.
- We deal with the covariate shift problem which occurs inevitably in the non-stationary environment such as different-time or different-device measurement. To minimize the covariate shift problem, covariate shift adaptation is applied.
- Our results from implementation on AVR micro-controller indicate that the successful recognition rate (SR) of the classifier is as high as 99.03% including the identification of registers using test data.

Even though AVR micro-controller is not a the most complex modern device, it still represents a more challenging case study than those in the current state-of-the-art research. We also believe that our method can be a great starting point to disassemble even recent embedded micro-controllers. The rest of the paper is organized as follows. Section 2 outlines our instruction-level disassembly, implemented on an AVR processor. Section 3 presents the proposed feature selection and reduction technique. Section 4 deals with the covariate shift problem. Section 5 presents the experimental setup and results. Finally, Section 6 concludes the paper.

2 POWER BASED DISASSEMBLER

The main focus of the side-channel based disassembler is to extract assembly level code from the side-channel leakage. A complete side-channel disassembler should estimate which instruction is executing, which register is used, or what value is processed with *only one power sample*. In this section, we discuss our proposed disassembler in the context of an AVR micro-controller (ATMega 328P) but emphasize that our approach is generalizable to devices of similar or higher complexity.

Fig. 1 shows the process flow for our proposed disassembler. First, power traces for pre-defined instruction classes are collected using an oscilloscope through a power side-channel of a training device. Second, the time-varying power traces are mapped into the time-frequency domain by continuous wavelet transform. Third, the feature selection using Kullback-Leibler (KL) divergence is executed. Fourth, selected feature values are normalized. Fifth, the number of feature variables is reduced by the dimensionality reduction using PCA. Sixth, power traces with reduced feature variables are trained by classifiers and then templates such as decision boundaries are generated. Finally, based on the templates, power traces collected from the target device (i.e., device under assessment) are classified and then the disassembler generates reverse-engineered assembly codes. Note that power traces of the target device (which may be different from the training device) go though the same pre-processing and dimensionality reduction as those of the training device.

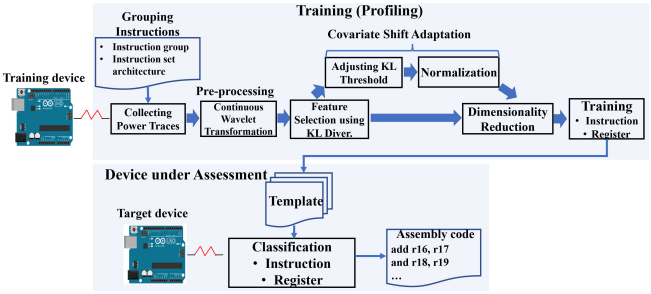


Figure 1: Process flow for our proposed disassembler.

2.1 Grouping Instructions

The ATMega 328P has 131 instructions [12]. Most of the instructions have single clock cycle execution. 112 instructions except for residual control instructions, multiplication instructions, and residual branch instructions can be recognized by our disassembler. Since the number of classes corresponding to instructions is very large, 112 instructions are divided into 8 groups based on operands. The operands of the instruction are related to which micro-architectural components such as ALU and data memory being used. Since the different group of instructions uses different architectural components, power signatures between different groups are more distinguishable. For example, the first group instructions consist of opcode, a destination register (Rd) and a source register (Rr). 12 arithmetic and logic instructions belong to the first group. Table 2 summarizes all instructions, needed operands and description in 8 groups.

Classification is performed hierarchically in three levels. A measured power trace $\boldsymbol{l}$ is classified into an instruction group among

Table 2: Grouping AVR instructions.

	Group1	Group2	Group3	Group4	Group5	Group6	Group7	Group8
Insts.	ADD, ADC, SUB SBC, AND, OR EOR, CPSE, CP, CPC, MOV, MOVW	ADIW, SUBI, SBCI SBIW, ANDI, ORI SBR, CBR, CPI LDI	COM, NEG, INC DEC, TST, CLR SER, LSL, LSR ROL, ROR, ASR SWAP	RJMP, JMP, BREQ BRNE, BRCS, BRCC BRSH, BRLO, BRMI BRPL, BRGE, BRLT BRHS, BRHC, BRTS BRTC, BRVS, BRVC BRIE, BRID	LDS LD LDD STS ST STD	SEC, CLC, SEN CLN, SEZ, CLZ SEI, SES, CLS SEV, CLV, SET CLT, SHE, CLH	SBRC, SBRS SBIC, SBIS BRBS, BRBC SBI, CBI BST, BLD BSET, BCLR	LPM ELPM
Operands	Rd, Rr	Rd, K	Rd	k	Rd, k Rd, (−)X(+) Rd, (−)Y(+(q)) Rd, (−)Z(+(q))		Rr(Rd), b A, b s, k s	Rd, Z(+)
# of Insts.	12	10	13	20	24	15	12	6
Description	Arith.[1]	Arith.,[1] Data.[2]	Bit.,[3] Arith.[1]	Bran.[4]	Data.[2]	Bit.[3]	Bran.,[4] Bit.[3]	Data.[2]

[1] Arithmetic and logic instruction, [2] Data instruction, [3] Bit and bit-test instruction, [4] Branch instruction

8 groups at the first phase and then it is classified into a specific instruction in the selected group at the second phase. Finally, the disassembler decides which operands are used for the execution in the third phase. Our proposed hierarchical classification significantly reduces computational complexity in case of classification of large classes significantly. For example, if binary classifiers such as SVM are exploited and one-vs-one strategy is selected, $K(K-1)/2$ classifiers where K is the number of classes, should be required. In case of 112 classes, 6216 classifiers should be trained. On the other hand, using hierarchical one-vs-one SVM, at most 218 classifiers are necessary when a power trace is recognized as an instruction in the 4th group: $218 = \binom{8}{2} + \binom{20}{2}$. In addition, components of the operand are automatically determined after finishing instruction classification.

3 FEATURE SELECTION AND DIMENSIONALITY REDUCTION IN TIME-FREQUENCY DOMAIN

Measured power traces are mapped into two-dimensional time-frequency region by continuous wavelet transform (CWT) [7]. The wavelet transform has some advantages for side-channel analysis. It is used to remove noise from side-channel leakage traces obtained by oscilloscopes and to perfectly align collected traces [8]. In addition, the wavelet transform is required to solve covariate shift problem which will be described in Section 4. Based on positive results with CWT, we use the wavelet transform to extract distinct features among all instruction classes to discriminate or classify instructions. These distinct features should not be varying in non-stationary environment. That is, distinct and not-varying features in the non-stationary time-frequency domain are extracted from the wavelet transform of the collected traces.

In the time domain, a fetch/decode clock cycle and an execution clock cycle of each instruction are associated with 315 (= $\lfloor 2.5G/16M * 2 \rfloor + 2$ additional) sampling points based on our setup at Section 5.1. But due to CWT transformation, each sampling point is expanded into a 50-length vector including frequency components. Thus, 315 sampling points from the time domain result in 15750 (315×50) sampling points in the transformed domain. In order to extract distinct and not-varying points among 15750 sampling points, Kullback-Leibler (KL) divergence [22] is used. The KL divergence is an useful metric for the feature selection. The higher the KL divergence between two random variables, the more distinguishable two random variables are. The specific sampling points should have large KL-divergence value. An additional desirable property is that the specific sampling points do not have dependency (or co-linearity). The specific sampling points have locally maximum value to satisfy the two conditions. In order to make two classes more distinguishable, principle component analysis (PCA) can be applied to these specific sampling points.

3.1 KL Divergence-based Feature Selection

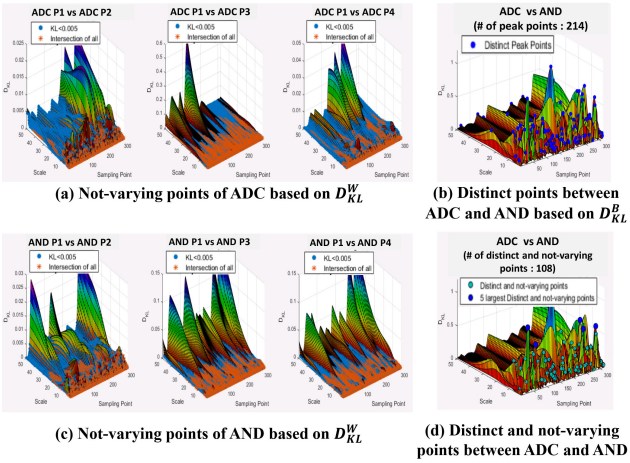
Let $f_X(z)$ and $f_Y(z)$ be the probability density functions of random variables X and Y, respectively. The KL divergence captures the distance between two distributions and is defined by the following equation [22]:

$$D_{KL}(X||Y) = \int f_X(z) \log \frac{f_X(z)}{f_Y(z)} dz. \quad (1)$$

The KL divergence of two random variables with normal distributions can be computed easily [20]. The time-frequency varying KL divergence between two group's power signatures shows a probabilistic distance between two different instructions. The peaks of the KL divergence exhibit distinct differences between the two traces. Wavelet transformed values at the specific time-frequency points where the KL divergence exhibits a peak can be given as a feature set to any classifier, such as Bayesian classifiers. However, all distinct feature points are not stationary. This means that values at the specific time and frequency index can be changed according to different measurement times or different program files even if the same instructions are executed. These distinct feature points based on KL divergence should be not varying in the non-stationary environment. We define the distinct and not-varying feature as follows:

Definition 3.1. 1) The between-class KL divergence is defined as $D^B_{KL}(cfs_{c_1}(j,k) \parallel cfs_{c_2}(j,k))$, where $cfs_{c_i}(j,k)$ is a set of CWT coefficients at *the frequency (or scale) index j* and *the time (or sampling) index k* in a class c_i.
2) The within-class KL divergence is defined as $D^W_{KL}(cfs_{c_i,p_1}(j,k) \parallel cfs_{c_i,p_2}(j,k))$, where $cfs_{c_l,p_m}(j,k))$ is a set of CWT coefficients at *the frequency (or scale) index j* and *the time (or sampling) index k* of a program p_m in a class c_l.
3) If $D^B_{KL}(cfs_{c_1}(j,k) \parallel cfs_{c_2}(j,k))$ is local maxima (or peak value) and $D^W_{KL}(cfs_{c_i,p_1}(j,k) \parallel cfs_{c_i,p_2}(j,k))$, for $i = 1,2$ is less than KL threshold, KL_{th}, the index pair (j,k) is called a distinct and not-varying feature point.

Fig. 2 shows five extracted feature points for classification between ADC and AND instructions. Not-varying feature points, $NVP_c = \{(j,k)\}$ are selected such that $D^W_{KL}(cfs_{c,p_m}(j,k) \parallel cfs_{c,p_n}(j,k)) < KL_{th} = 0.005$ for $1 \le m \ne n \le 10$, $j = 1,2,\ldots 50$, $k = 1,2,\ldots 315$ within each class $c \in \{\text{ADC}, \text{AND}\}$ at Fig. 2 (a) and (c). Note that 2500 instructions for profiling in each class are distributed into 10 different program files p_i. Distinct feature points, $DP = \{(j,k)\}$ between different classes are extracted such that $\frac{\partial^2}{\partial j \partial k} D^B_{KL}(cfs_{\text{ADC}}(j,k)$



(a) Not-varying points of ADC based on D_{KL}^{W}

(b) Distinct points between ADC and AND based on D_{KL}^{B}

(c) Not-varying points of AND based on D_{KL}^{W}

(d) Distinct and not-varying points between ADC and AND

Figure 2: Extracting feature points in the time-frequency domain to classify between ADC and AND using KL divergence.

$||cfs_{\mathsf{AND}}(j,k)) = 0$ at Fig. 2 (b). Distinct and not-varying feature points are extracted such that $NVP_{\mathsf{ADC}} \cap NVP_{\mathsf{AND}} \cap DP$ and then 5 highest distinct and not-varying feature points, $DNVP^{(5)}$ are finally extracted at Fig. 2 (d). 5 distinct and not-varying feature points of all possible combinations of two instructions in each each group are unified such that $\cup_{i=1}^{n_c} DNVP_i^{(5)}$, where $n_c = \binom{|C|}{2}$ is the number of all possible combinations. The number of unified is 205 in the first instruction group, which means reduction from 15,750 by 98.7%. These points are used as variables for PCA for further dimensionality reduction.

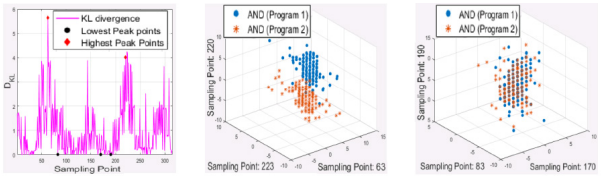
3.2 Principal Component Analysis

Principal component analysis (PCA) is generally used for unsupervised dimensionality reduction. It forms new variables which are linear combinations of the original variables. The new variables retain as much of the original variance as possible and are uncorrelated with each other. The number of new variables (k) is equal to the number of original variables (p). A small number $k \ll p$ of the principal components can account for much of the total system variability. The feature points selected with KL divergence are used as the original variables for PCA. The projected variables onto the new coordinate system are more distinguishable.

4 COVARIATE SHIFT PROBLEM

In our initial experiment, 3000 power traces per each class are obtained by the execution of instructions in 10 different program files. 2500 power traces and 500 power traces are randomly selected for training and testing, respectively. This means that some power traces for testing are generated from the same program files as profiled power traces. This scenario is different from practical scenario in which power traces are obtained by a real program file to disassemble. The classifiers trained by the experimental scenario cannot distinguish power traces in the real program due to the covariate shift problem. Covariate shift problem is caused by the different probability distribution of training data and testing data such that $Pr_{te}(\boldsymbol{x}) \neq Pr_{tr}(\boldsymbol{x})$ even if the conditional probability of classes given training data is the same as the conditional probability given testing data ($\Pr[C|\boldsymbol{x}_{tr}] = \Pr[C|\boldsymbol{x}_{te}]$) [24].

Power traces in the same instruction should be identical in ideal cases but when they are executed in different programs, they are measured differently in our experiment. They have the similar shape but different DC offsets. This covariate shift problem also occurs in power measurement at different times or across devices [5]. Even if power traces are generated by the same instructions, they can be recognized into different instructions because of covariate shift problem. Fig. 3 shows that power traces of 2 different AND programs

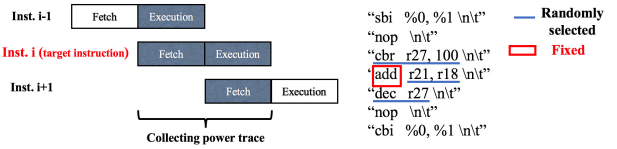


(a) KL divergence between 2 different programs

(b) Worst case feature selection (3 highest peak points)

(c) Best case feature selection (3 lowest peak points)

Figure 3: Best and worst feature selection based on KL divergence.



(a) Instruction sequence in a pipeline

(b) Program segment template

Figure 4: Instruction sequence in a pipeline and a program segment template for ADD instruction.

are completely separate or combined depending on KL-divergence based feature selection. When 3 lowest peak points are selected as the feature set, AND power traces executed in two different programs are gathered in a cluster. On the other hand, AND power traces are scattered into 2 different groups when 3 highest peak points are chosen as the feature set.

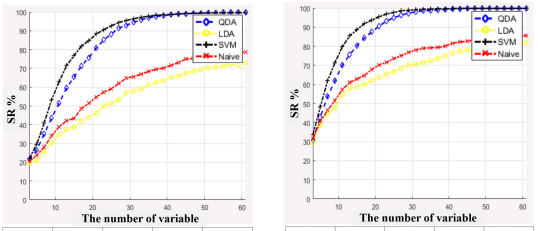
2700 power traces executed in 9 different programs are trained and 300 power traces executed in a real program are tested per each instruction like the practical scenario. Training data of any two instructions such as ADC and ADD are separate but two test data such as ADC and AND are rotated into the opposite direction as well as having more overlapping area. SRs of the training data and test data using QDA are 94.3% and 18.5 %, respectively. In order to solve this problem, covariate shift adaptation at Section 5.5 will be applied.

5 EXPERIMENTS

5.1 Experimental Setup

For training or profiling, power traces are collected from an ATMega 328P micro-controller with a clock frequency of 16 MHz. Other five ATMega 328P micro-controllers are used as target devices. Using Tektronix MDO3102 oscilloscope, the voltage of a shunt resistor (330 Ω) between the GND pin and the ground is measured. The setup of the oscilloscope is 2.5GS/s, 250MHz bandwidth, 10k sample points and sample mode.

Since the AVR micro-controller has 2 pipeline stages, a target profiled instruction is affected by a previous instruction and a following instruction. Each power trace is measured with the following program segment template: `SBI, NOP, a randomly selected instruction, target instruction, a randomly selected instruction, NOP, CBI`, where `SBI(CBI)` means that set (clear) bit in I/O register and `NOP` means no operation. The `SBI` and `CBI` instructions are executed for the trigger signal. In order to remove power consumption of `SBI` and `CBI` instruction and electrical noise, we compute the difference between each power trace and the reference power traces of `SBI, 5 NOPs` and `CBI` sequence. Fig. 4 shows the instruction sequence in a pipeline and a program segment template for obtaining a power trace for `ADD` instruction. For training and classification, 3000 power traces for each instruction with randomly selected source register `Rr` and destination register `Rd` (the values of the `Rr` and `Rd` also are randomly distributed) are sampled. We also measure 3000 power traces per unique `Rd` with randomly selected instruction and `Rr` and 3000 power traces per unique `Rr` with randomly selected instruction and `Rd`.



# of var.	QDA	LDA	SVM	Naive
43	99.6	77.35	99.85	83.35

(a) Classification of groups (Group1 ~ Group8)

# of var.	QDA	LDA	SVM	Naive
61	99.93	74.53	99.7	80.22

(b) Classification of the first group's instructions

Figure 5: Successful recognition rate (SR) of (a) instruction groups and (b) 1st group's instructions.

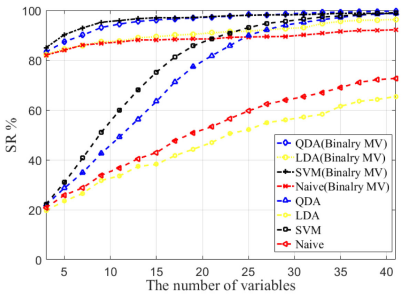


Figure 6: Successful recognition rate (SR) of the 1st group's instructions using majority voting method and general method.

In order to collect power traces automatically, we built an acquisition framework which consists of a PC, an oscilloscope, an Arduino board and software. 10 program files per each instruction or register are uploaded to the Arduino sequentially to collect 3000 power traces since the flash memory of the Arduino is limited. A file includes 300 program segment templates. We use Perl script to generate Arduino code automatically. The generated files (.ino) is uploaded to the board using Arduino IDE tool. For automatically uploading 10 files per each class, MATLAB makes Arduino IDE to upload a file as soon as previous program is complete. During execution, power traces are collected using an oscilloscope and stored in PC automatically. TekVISA (Tektronix's Virtual Instrument Software Architecture) provides communication interface between the oscilloscope and the PC. All required software such as Perl compiler, Arduino IDE and TekVISA can be controlled by MATLAB.

5.2 Training and Classification for Instructions

After dimensionality reduction, 2500 power traces per each class are used for the training. Linear discriminant analysis (LDA), quadratic discriminant analysis (QDA), support vector machine (SVM) and naïve Bayes method are tested. `fitcdiscr` function in MATLAB statistical toolbox is used to construct LDA and QDA classifiers and `fitcnb` function is used to train a naïve Bayes classifier. LIBSVM [4] is used for the SVM classifier with RBF kernel. The best penalty parameter C and the best Gaussian kernel parameter $\gamma = 1/\sigma^2$ are selected by the grid search with 3-fold cross-validation.

Fig. 5 shows SR of instruction groups and the 1st groups's instructions depending on the number of principal components. SVM classifier with RBF kernel has the best performance in terms of the successful recognition rates (SR). The SR of instruction groups saturates to 99.85% using SVM classifier with 43 variables and the SR of the first group instructions is saturated to 99.7%. QDA classifier using 43 variables also has 99.93% SR but it has lower SR than SVM classifier when using less than 43 variables. SRs of other group's instructions saturates to greater than 99.5% using SVM with more than 50 variables. Thus, SR of QDA to identify an opcode is at most 99.53% (99.6% × 99.93%) and at least 99.1% (99.6% × 99.5%). SR of SVM to identify an opcode is from 99.55% (99.85% × 99.7%) to 99.35% (99.85% × 99.5%).

5.3 Training and Classification for Registers

In order to recognize the address of the destination register `Rd0` ~ `Rd31` and the source register `Rr0` ~ `Rr31`, we follow the same process. We collect power traces, profile to extract an appropriate feature vector, and then classify. For each fixed target register, the instruction opcode and the other register are randomly selected. Dimensionality reduction techniques extract a feature vector from each of the 2.5K power traces, which are used for training. The other 500 power traces are used as the test set. QDA has the best performance with 99.9% and 99.6% SR of `Rd` and `Rr` using 45 variables, respectively. As a result, SR of a instruction set including opcode and both registers is at most 99.03% (99.53%×99.9%×99.6%) using QDA.

5.4 Majority Voting Method

When classifying multiple classes (greater then 3 classes), one-vs-one strategy using binary classifiers can be used. Let $f_{i,j}$ be the binary classifier to distinguish between two classes c_i and c_j :

$$f_{i,j}(\boldsymbol{x}_{i,j}) = \begin{cases} +1 & \text{if } \Pr[c_i|\boldsymbol{x}_{i,j}] \geq \Pr[c_j|\boldsymbol{x}_{i,j}] \\ -1 & \text{if otherwise} \end{cases} \quad (2)$$

where $\boldsymbol{x}_{i,j}$ is the feature vector after dimensionality reduction based on $D^B_{KL}(c_i||c_j)$, D^W_{KL} and PCA. By majority voting method, the class $\hat{c}$ is selected as follows:

$$\hat{c} = \arg\max_{c_i \in C} \sum_{i \neq j} f_{i,j}(\boldsymbol{x}_{i,j}). \quad (3)$$

The number of required binary classifiers is equal to $\binom{|C|}{2} = K(K-1)/2$ where K is the number of classes.

Distinct and not-varying feature points based on KL divergence depends on two instructions. Principal components transformed from unified $DNVP$ at Sec. 3.1 cannot be the best components in the classification between any two instructions. But at Eq. (2), $\boldsymbol{x}_{i,j}$ is the best feature vector for the classification between c_i and c_j. Even if the number of needed binary classifier is large, the number of variables or the length of the feature vector in each binary classifier can be reduced significantly without the loss of performance. The number of variables can be an important parameter for disassembly of high-clock frequency micro-controllers. In order to get over 99% SR, at least 40 feature points (or variables) during 2 clock cycles are required in our experiments. This means that the sampling rate should be at least 20 times higher than the clock frequency. For example, for a target device working at 1 GHz, a 20 Gs/s oscilloscope is required. It is very expensive and not practical. If 10 feature points are the requirement to satisfy 99% SR, an oscilloscope with 5 Gs/s is enough. Thus, the number of feature points should be reduced significantly.

Fig. 6 shows successful recognition rates of the 1st group's instructions using majority voting method and previous general method depending on the number of variables. SRs using majority voting method with only 3 variables are 82.25 %, 83.22 %, 85 % and 82.02 % corresponding to LDA, QDA and SVM and Naïve Bayes classifier, respectively, which are significantly improved compared with the previous general method. The SVM classifier with 9 variables has 95.2 % SR and it has the best performance in all cases.

5.5 Covariate Shift Adaptation

The training data of each class have been generated from 9 different programs which are not enough to estimate not-varying feature points against the training programs in the practical scenario. The number of training programs are increased to 19 which means that the number of training data is equal to 5700. Also, KL_{th} at Def. 2.1

Table 3: Successful recognition rate (SR) of classification between ADC and AND with covariate shift adaptation (CSA).

Classifier	Without CSA	Without Norm.	With Norm.
QDA	18.5%	54.3%	92%
SVM	19.2%	57.8%	93.2%

Table 4: Successful recognition rate (SR) of classification between ADC and AND in 5 different devices.

Classifier	Dev. 1	Dev. 2	Dev. 3	Dev. 4	Dev. 5
QDA	89.3%	91.5%	88.9%	92.3%	94.5%
SVM	90.4%	92.8%	90.8%	93.4%	95.6%

is reduced from 0.005 to 0.0005. That is, not-varying feature points in each class are selected as follows:

$$NVP_c = \Big\{(j,k)|D_{KL}^{W}\left(cfs_{c,p_m}(j,k) \parallel cfs_{c,p_n}(j,k)\right) < KL_{th} = 0.0005, 1 \le m \ne n \le 19, j = 1,2,\ldots,50, k = 1,2,\ldots,315\Big\} \quad (4)$$

The values of distinct and not-varying feature points between two different classes are normalized in order to reduce the range of shifted space. The normalization is an important process and compared to without-normalization results. The number of variables are reduced by PCA dimensionality reduction and 3 principle components are selected as the variables. Table 3 shows the successful recognition rate of classification between ADC and AND instructions when modified sampling method is applied and this table shows the comparison between normalized values and without-normalized values. The SR using QDA is increased by 73.5 % compared with 18.5 %.

5.6 Covariate Shift Caused by Different Devices

Covariate shift problem also occurs in measured powers from different devices which are the same model as the trained device. Based on the template from a trained device, measured powers from 5 test devices are classified. 300 test power traces of ADC and AND instructions per each device are collected for the comparison with the previous result. Covariate shift problem caused by different devices is similar to that by different programs. After covariate shift adaptation, the SR is increased significantly. Table 4 shows the results of 5 different devices. In summary, covariate shift problems caused by different programs and devices are minimized by expanding sample space and searching not-varying feature points with normalization.

5.7 Case Study: Malware Detection

An adversary can insert a malicious code which has the same functionality as the original code but provides significant information to the adversary. For example, the original key of AES encryption is masked with a random number to prevent from the first-order side-channel attack [21]. If the random number is a fixed value such as all zeros binary number or all ones binary number, the masking method is useless so that it has vulnerable against the first-order side-channel attack. In malware, the original code, `xor r16, r17` is changed into `xor r16, r0`, where an original 8-bit subkey, a 8-bit random number and a zero number are stored in `r16, r17` and `r0`, respectively. That is, the original key is still stored in `r16` after executing the instruction and a following non-linear operation (Sbox) with the unmasking key generates significant side-channel leakage. In this case, our disassembler can detect the change of the source register by malware perfectly.

6 CONCLUSION AND FUTURE WORK

Consumer electronics devices and IoT devices are targeted by malware such as Mirai. They do not have ability to detect or prevent malware. In order to detect malware, monitoring IoT devices is required. Instruction-level disassembly through power side-channel can be used as the monitor. We describe our preliminary experiences with instruction level disassembly of an AVR micro-controller. We are able to identify an instruction set including opcode and registers with 99.03% accuracy through power side-channel. This technique can be used with static code analysis in order to increase accuracy of real code. In our future work, we will apply our method on real code to mimic the complete reverse-engineering in practical world scenario. Also, state-of-the-art microcontrollers running at higher frequency clock will also be tested to evaluate the performance of our proposed disassembler.

ACKNOWLEDGMENTS

This work is supported in part by NIST, Award #60NANB17D040.

REFERENCES

[1] Dakshi Agrawal, Josyula R. Rao, and Pankaj Rohatgi. 2003. *Multi-channel Attacks.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2–16.
[2] Waqas Amir. 2016. Hackers are increasingly targeting IoT Devices with Mirai DDoS Malware. https://www.hackread.com/iot-devices-with-mirai-ddos-malware/. (Oct. 2016).
[3] Gerardo Canfora, Massimiliano Di Penta, and Luigi Cerulo. 2011. Achievements and Challenges in Software Reverse Engineering. *Commun. ACM* 54, 4 (April 2011), 142–151.
[4] Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A Library for Support Vector Machines. *ACM Trans. Intell. Syst. Technol.* 2, 3, Article 27 (May 2011), 27 pages.
[5] Omar Choudary and Markus G. Kuhn. 2014. *Template Attacks on Different Devices.* Springer International Publishing, Cham, 179–198.
[6] Teodoro Cipresso and Mark Stamp. 2010. *Software Reverse Engineering.* Springer Berlin Heidelberg, Berlin, Heidelberg, 659–696.
[7] Leon Cohen. 1995. *Time-frequency Analysis: Theory and Applications.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA.
[8] Nicolas Debande, Youssef Souissi, M. Abdelaziz El Aabid, Sylvain Guilley, and Jean-Luc Danger. 2012. Wavelet Transform Based Pre-processing for Side Channel Analysis. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture Workshops (MICROW '12).* 32–38.
[9] Thomas Eisenbarth, Christof Paar, and BjÃűrn Weghenkel. 2010. Building a Side Channel Based Disassembler. In *Transactions on Computational Science X*, Marina L. Gavrilova, C.J. Kenneth Tan, and Edward David Moreno (Eds.). Lecture Notes in Computer Science, Vol. 6340. Springer Berlin Heidelberg, 78–99.
[10] Jake Longo Galea, Elke De Mulder, Daniel Page, and Michael Tunstall. 2015. SoC it to EM: electromagnetic side-channel attacks on a complex system-on-chip. *IACR Cryptology ePrint Archive* 2015 (2015), 561.
[11] Michael Henson and Stephen Taylor. 2014. Memory Encryption: A Survey of Existing Techniques. *ACM Comput. Surv.* 46, 4, Article 53 (March 2014), 26 pages.
[12] Atmel Inc. 2016. AVR Instruction set manual. (2016). http://www.atmel.com/images/Atmel-0856-AVR-Instruction-Set-Manual.pdf
[13] Paul Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential Power Analysis. Springer-Verlag, 388–397.
[14] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. 2007. *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security).* Springer-Verlag New York, Inc., Secaucus, NJ, USA.
[15] Muhammad Ali Mazidi, Sarmad Naimi, and Sepehr Naimi. 2010. *AVR Microcontroller and Embedded Systems: Using Assembly and C* (1st ed.). Prentice Hall Press, Upper Saddle River, NJ, USA.
[16] David McCann, Carolyn Whitnall, and Elisabeth Oswald. 2016. ELMO: Emulating Leaks for the ARM Cortex-M0 without Access to a Side Channel Lab. Cryptology ePrint Archive, Report 2016/517. (2016).
[17] Amir Moradi, David Oswald, Christof Paar, and Pawel Swierczynski. 2013. Side-channel Attacks on the Bitstream Encryption Mechanism of Altera Stratix II: Facilitating Black-box Analysis Using Software Reverse-engineering. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '13).* 91–100.
[18] Mehari Msgna, Konstantinos Markantonakis, and Keith Mayes. 2014. *Precise Instruction-Level Side Channel Profiling of Embedded Processors.*
[19] M. Ozsoy, K. N. Khasawneh, C. Donovick, I. Gorelik, N. Abu-Ghazaleh, and D. Ponomarev. 2016. Hardware-Based Malware Detection Using Low-Level Architectural Features. *IEEE Trans. Comput.* 65, 11 (2016), 3332–3344.
[20] Jungmin Park and Akhilesh Tyagi. 2016. Security Metrics for Power Based SCA Resistant Hardware Implementation. In *29th International Conference on VLSI Design and 15th International Conference on Embedded Systems, VLSID 2016, Kolkata, India, January 4-8, 2016.* IEEE Computer Society, 541–546.
[21] Emmanuel Prouff and Matthieu Rivain. 2007. *A Generic Method for Secure SBox Implementation.* Springer Berlin Heidelberg, Berlin, Heidelberg, 227–244.
[22] S. Kullback and R. A. Leibler. 1951. On Information and Sufficiency. *The Annals of Mathematical Statistics* 22, 1 (1951), 79–86.
[23] Daehyun Strobel, Florian Bache, David Oswald, Falk Schellenberg, and Christof Paar. 2015. Scandalee: a side-channel-based disassembler using local electromagnetic emanations. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, DATE 2015, Grenoble, France, March 9-13, 2015.* 139–144.
[24] Masashi Sugiyama and Motoaki Kawanabe. 2012. *Machine Learning in Non-Stationary Environments: Introduction to Covariate Shift Adaptation.* The MIT Press.