Performance Counters to Rescue: A Machine Learning based safeguard against Micro-architectural Side-Channel-Attacks

Manaar Alam, Sarani Bhattacharya, Debdeep Mukhopadhyay, and Sourangshu Bhattacharya

Indian Institute of Technology, Kharagpur, India,
alam.manaar@iitkgp.ac.in, {sarani.bhattacharya, debdeep, sourangshu}@cse.iitkgp.ernet.in

Abstract. Micro-architectural side-channel-attacks are presently daunting threats to most mathematically elegant encryption algorithms. Even though there exist various defense mechanisms, most of them come with the extra overhead of implementation. Recent studies have prevented some particular categories of these attacks but fail to address the detection of other classes. This paper presents a generic machine learning based multi-layer detection approach targeting these micro-architectural side-channel-attacks, without concentrating on a single category. The proposed approach work by profiling low-level hardware events using Linux *perf_event* API and then by analyzing these data with some appropriate machine learning techniques. This paper also presents a novel approach, using time-series data, to correlate the execution trace of the adversary with the secret key of encryption for dealing with false-positives and unknown attacks. The experimental results and performance of the proposed approach suggest its superiority with high detection accuracy and low performance overhead.

Keywords: Micro-Architectural Side-Channel-Attack, Hardware Performance Counters, Machine Learning, Anomaly Detection, Classification, Time-Series

1 Introduction

Computer hardware is increasingly getting shared between multiple, potentially untrusted programs, ranging from cloud services, where the workloads of multiple clients may be shared on a single computer, to mobile phones, where apps authored by different developers run on a single hardware, to web browsers displaying pages from various sites. However, for most of these applications security is imperative. But in spite of developments of strong cryptographic algorithms, confidentiality or integrity of information is threatened on these platforms due to the presence of covert information leakage channels, which are exploited in several attacks. These attacks target the micro-architecture of the platforms and can collapse the strongest of crypto-algorithms, like the Advanced Encryption

Standard (AES) or the mathematically elegant RSA or Elliptic Curve Cryptosystems. Popular examples of these attacks are those which target the events like cache accesses [1], branch instructions [2], etc. Modern microprocessors contain a set of special purpose registers to measure hardware related activities known as hardware performance counters, which leak valuable information regarding the encryption algorithm [3]. Some attacks [4] analyze these performance counters for compromising the security of the system. It is now well-established that modern computers will exhibit information leakages and we have to develop our security defenses amidst the presence of these leakage avenues [5].

There are many state-of-the-art countermeasures to prevent these types of breach of security, but with the cost of a severe extra overhead which can blow up the code size and cut down the performance and increase the energy consumption of the device manifold. Some past works to defeat side-channel-attacks work by significantly changing the hardware [6, 7] or by obscuring timing information of a program's execution [8]. Implementations of these counter-measures are not feasible in resource-constrained environments like IoT devices, Smart phones, etc. Even for standard commercial platforms, the protection costs may hinder them as unusable.

In recent times, researchers are working to detect and prevent the side-channel-attacks without modifying the hardware of the system rather by implementing a user-level process which monitors the execution of all other processes [9, 10] in the system or by implementing lightweight patch [11] in the operating system. These works concentrated on traditional non-trivial detection techniques like signature-based detection and anomaly-based detection. In the former technique pattern recognition is used to detect known attack methods, and in the latter method, known and unknown attacks are detected by observing significant deviations from the normal behaviors. The former method is low in false negatives but also have the risk of missing unknown adversaries, whereas, the latter method can successfully detect unknown attacks with the known ones but can incur significant false positives, as sudden changes produced by benign programs can be wrongly classified and raise triggers in the system.

M. Chiappetta et al. [9] experimented with both the approaches with three different methods but have not addressed the difficulties caused by false negatives. On the other hand, T. Zhang et al. [11] used a multi-layer approach to deal with false positives and false negatives in a Cloud Environment, but the discrimination between a side-channel-attack and a memory intensive program is difficult to obtain with high confidence with their approach. Moreover, most of these recent works address the detection of cache-based side-channel-attacks and not considering the other forms of micro-architectural side-channel-attacks like branch mis-prediction attacks or software driven fault attack such as Rowhammer attacks, etc. Also, these works did not adequately address the detection of an 'intelligent' potentially evasive attack process which changes its behavior by reducing the cache probing speed or by introducing some random cache accesses in the code thereby confusing the detection mechanism.

In this paper, we investigate an idea of three-step real-time detection mechanism to deal with these difficulties and for mitigation of the mentioned security threats. First, our approach is to generate significant volumes of low-level data by profiling the hardware performance counters. The Linux *perf* utility allows for a granular profiling of chosen hardware events at a selected frequency, albeit with some lower margin. The first step is to process this data through elaborate data analysis and machine learning methods to raise alarms in the system. In the next step, we determine the cause of abnormality for the warned process and identify the possible attack-category the anomaly belongs to, by a pre-learned classifier which is learned with the behaviors of some state-of-the-art side-channel-attacks. Finally, we correlate the hardware performance counter values (determined based on the classification result) of the execution of the potential attack process with the secret key (which is known beforehand) of the encryption algorithm using a time-series based approach. A large correlation value with the secret key signifies true positive. We then with high confidence say that the warned process is an actual side-channel-attack and can take appropriate measure against the process.

Main idea and motivation

Most of the micro-architectural side-channel-attacks primarily work by analyzing the behavior of the CPU-cache, branch-predictor hardware for the secret computation. Abnormalities in the number of micro-architectural events in the normal environment can be observed in the presence of side-channel-attacks. The first phase of our approach detects the presence of these abnormalities in a system and raises a warning of a possible side-channel-attack. Not only the malicious processes, but there can be some other benign processes also which may create these abnormalities to the micro-architectures. To deal with these types of processes an anomaly detection module is used with a suitable threshold such that the abnormalities exceeding threshold can be termed as a potential side-channel-attack.

In the second phase, we analyze the cause of abnormality for the processes and categorize them using a pre-trained classifier (trained with the behavior of some state-of-the-art side-channel-attacks). The result of this phase would help us to understand the type of the possible attack process (i.e., a cache based attack or branch prediction based attack, and so on).

The classification phase can be confused with the presence of some false positive processes with the approximately same behavior as that of the malicious processes trained in the classifier. To remove these false positives from the detection mechanism we use the third phase of our detection where we correlate the execution of the program with the secret key which is already known to the root user. If the execution correlates with the key, then the warned process is a true positive and can be removed from the system. This third module could also be used for removing the false negatives (i.e., the unknown 'intelligent' attacks, which can modify its behavior to evade the detection mechanism), as to be a successful attack its execution needs to correlate with the secret key.

Most of the micro-architectural side-channel-attacks work by continuously monitoring a particular event (such as cache misses, branch misses, etc.) of the execution traces of the encryption algorithm. This abbreviates to the fact that, if we know the execution trace of the attacker exploited event for the encryption algorithm (say T_1), this trace will be contained in the execution trace of the successful attack code (say T_2) in addition to some noise because of the attackers own execution. To find the correlation, in this case, is nothing but the problem of temporal sequence alignment. This was the primary motivation behind using time-series analysis for correlating the hardware performance counter values with the secret encryption key. We discuss this approach in more details in the subsequent sections.

The correlation of performance counter values with the secret key can also help us to handle some corner cases. For example, in a multi-user system if any user (say user_1) is using the encryption algorithm with his own secret key and the side-channel-attack on the other hand is trying to compromise the secret key of another user (say user_2), most of the recent detection algorithms will categorize it as a case of side-channel-attack which really is not the case in terms of the user_1. Our detection algorithm can successfully show that the execution traces of the side-channel-attack does not correlate with the key of the user_1, which is analyzed in the results and discussion section.

The outcome of the third module can help us to re-train the classifier for improving its classification accuracy and also to accommodate new types of attacks in the classifier. As training a classifier is an expensive operation, this step can cause a significant performance overhead in a real-time implementation. This trade-off between accuracy and performance overhead can become a new research direction in future.

Our Contribution

The main contributions of our work are -

- A generalized detection mechanism. We tried to deal with maximum possible types micro-architectural side-channel-attacks including cache-based attacks, branch-prediction-based attacks, etc.
- Detection of an attack by correlating its execution trace with the secret encryption key. We proposed a novel approach which could be useful to remove false positives in the system and also be used to detect 'intelligent' unknown attacks.

The organization of rest of the paper is as follows: In Section 2, we present a brief but required background of different side-channel-attacks, hardware performance counters, data preprocessing techniques, classification task and time-series data. In Section 3, we discuss our detection methodology in details. In Section 4, we analyze the performance and evaluation of our proposed approach. Section 5 and Section 6 discuss the conclusion of our work and possible future research in this domain.

2 Preliminaries

In this section, we discuss the necessary information for better understanding of working of different micro-architectural side-channel-attacks and also the proposed detection mechanism for their efficient detection.

2.1 Side Channel Attacks

Side-Channel-Attack is an attack which works by the information gained from the physical implementation of a cryptosystem. These attacks can collapse any cryptosystem despite having any theoretical weaknesses. One of the special types of these attacks is micro-architectural side-channel-attack, in which attacker try to exploit different hardware events during the execution of the encryption algorithm (like, cache-miss, branch-miss, etc.) to compromise its secret encryption key. In this section, we brief the working of some of these well-known attacks.

Cache based Attacks In a cache-based side-channel-attack, the attacker tries to gain sensitive information from the victim by exploiting shared CPU-cache. This confidential information includes cryptographic operations, which are leaked as an attacker observable cache usage pattern. The adversary uses several techniques to manipulate content in this shared cache and tries to gain knowledge about the cache access pattern of the victim and make an inference about the sensitive operations that are responsible for this pattern.

There are different types of cache-based attacks, cache timing based, cache access based and cache trace based attacks. All of these attacks fundamentally rely on the variation of the timing of the cache hit and misses. The access pattern of the data being secret dependent these difference in timing tends to leak a significant amount of information.

In this paper, we have first considered the classical cache timing attack on AES by Bernstein [12] and alternatively on the Cache-timing attack on Clefia [13] to understand the behavior of cache timing based attacks. Cache trace based attacks rely on the same property of the non-uniformity in the access pattern of cache in a much special setting of spy and the victim process.

Without loss of generality, we consider that if our detection methodology can detect cache timing attack, it can do so for all genre of timing attacks.

Branch Prediction based Attacks In branch prediction based attack, the adversary exploits the common branch predictor implemented in all modern CPUs. The extra clock cycle due to the branch mis-prediction leaks valuable information to the attacker.

For Public key cryptographic systems, the control flow of the execution is secret key dependent. The most popular asymmetric key algorithms such as RSA and ECC have the key-dependent conditional statements executions for their underlying exponentiation and multiplication primitives. Since the control-flow of these algorithms vary depending on the key, the branch mis-prediction

counts bear a dependence to the underlying secret. This relationship has been exploited by the attackers to leak the secret of mathematically secure ciphers.

In our paper, we show that we can successfully identify these attacks from the benign processes.

DRAM based Attacks Rowhammer is a term coined for disturbances observed in recent DRAM devices where repeated row activation causes the DRAM cells to electrically interact among themselves [14–17]. Repeated access to rows in the DRAM results in bit flips [15] in its adjacent rows due to continuous charging and discharging of the neighboring cells in the rows. This induces a new genre of software induced fault attacks in the DRAM, and the effect is disastrous in some cases. One specific requirement to induce rowhammer fault is to make persistent accesses to the DRAM, either by causing regular cache misses using clflush or by using huge page mapping support. Thus to induce rowhammer faults, the attacker has to make accesses to the DRAM repeatedly, and this phenomenon intuitively results in higher number of cache misses. This event of high cache misses are used in our detection system.

In this paper, we thus also consider the software driven fault attack into our consideration for our following analysis.

2.2 Hardware Performance Counters

Hardware Performance Counters (HPCs) are a set of special purpose registers, which are present in most of the modern microprocessor's Performance Monitoring Unit (PMU). These registers can be programmed to store the number of occurrences of different types of hardware and software events related to the execution of a program, such as cache misses, retired instructions, retired branch instructions, and so on. HPCs were primarily designed to debug the performance of complex software systems, but currently, they are widely used for collecting the run-time behavioral information of software execution. HPCs work along with the event selectors, which specify the hardware events to be monitored and a digital logic which increments a counter based on the occurrence of the specified hardware events. These performance counters can be accessed very fast without affecting or slowing down any software execution. Moreover, to get access to these registers no source code modification is required. Hardware Performance counters has been used in many recent literatures [18–21] for dynamic profiling of a system.

The most useful mode of operation of PMUs is the interrupt-based mode. The central working principle behind this mode of operation is, a system interrupt is generated when a specified event occurs more than or equal to a predefined threshold value or a preset amount of time has elapsed. This mode of operation makes both event-based and time-based sampling possible.

High-level libraries like PAPI [22], OProfile [23] provide interfaces to HPCs. Linux perf [24] among them is a widely used new implementation of performance counters support for all Linux 2.6+ based systems, which we can access from

user-space. This tool is capable of providing per-process, per-CPU, and system-wide statistical profile. We used this tool for our experimentation purpose. Perf tool is based on Linux perf_event_open() system call, which can be used to profile system in very low granularity.

Almost every popular operating systems have HPC-based profilers, though the type and number of hardware events may vary across different Instruction Set Architectures (ISA) [25]. Most of the modern processors may offer thousands of hardware and software events to monitor, however, only a selected few of them can be monitored in parallel due to the limitation in the number of built-in HPC registers. Intel 64 and IA-32 architectures [25] provide facilities for monitoring performance via a Performance Monitoring Unit (PMU). There are more than 100 performance events which can be monitored to measure the performance of a program. Since we target micro-architectural attack in our study, we considered the hardware events which are more likely to be affected by these attacks.

Micro-architectural attacks like branch prediction based attack work in such a way that we can observe its influence in hardware events such as Branch Instruction Retired and Branch Misses Retired. The cache-based attacks will affect the hardware events such as LLC References, and LLC Misses more than the other events. Some other attacks may influence some other hardware counters also.

Case Study: I In the form of a case study, we try to emphasize on the fact that the detection of different side-channel-attacks depends on the selection of appropriate hardware events. In our experimentation system, we were able to monitor eight events in parallel, which we discuss in brief -

1. Branch Instruction Retired : This event counts the number of branch instructions at retirement.
2. Branch Misses Retired : This event counts the number of mis-predicted branch instructions at retirement.
3. Last Level Cache References : This event counts the number of requests originating from the core that references a cache line in the last level cache.
4. Last Level Cache Misses : This event counts the number of cache miss conditions for references to the last level cache.
5. Instruction Retired : This event counts the total number of instructions at retirement.
6. UnHalted Core Cycles : This event counts the number of core clock cycles when the clock signal on a particular core is running.
7. UnHalted Reference Cycles : This event counts the number of reference clock cycles at a fixed frequency while the clock signal on the core is running.
8. Bus Cycles : This event counts the number of bus cycles required with the system clock signal on the core running.

We collected a system-wide profile for different state-of-the-art branch-prediction attacks and cache-based attacks to form a dataset of benign and malicious behavior on the system. After preprocessing the dataset, which we will discuss

next, we use a standard Stability Selection [26] method. The following figure shows the importance of each of the selected hardware events over the others for the detection of the attacks.

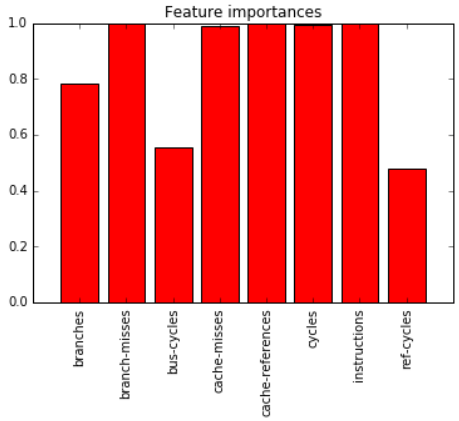


Fig. 1: Importance of each hardware event

From Figure 1, we can easily see that the hardware events like Branch Misses Retired, LLC Misses, LLC References, UnHalted Core Cycles, Instructions Retired are the most useful events when detecting these side-channel-attack, which is justifiable since theoretically, these micro-architectural attacks exploit these hardware events most while compromising the secret key.

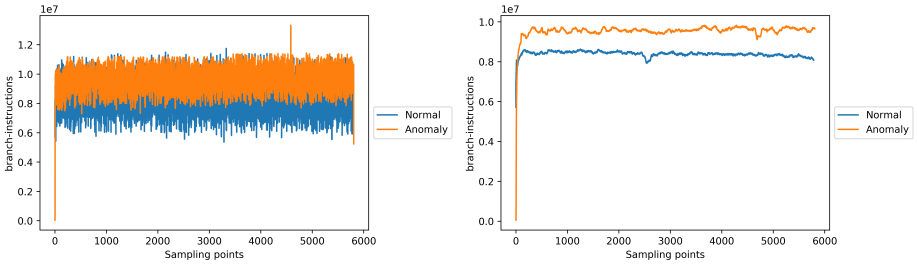
Eloquent case studies like this helped us to select the appropriate hardware events for our detection mechanism.

2.3 Data Preprocessing

Data collection techniques have some uncontrollable parameters, resulting in some garbage values. Also, there could be much irrelevance and redundancy in the data, which if not properly handled can generate misleading conclusions and the knowledge discovery from the data set becomes much more challenging. Thus, the data preprocessing is a major step before running any analysis.

In this section, we discuss some of these data preprocessing techniques which we have used in our experimentation.

Data Smoothing We use one of the simplest types of finite impulse response filter, namely Simple Moving Average (SMA) [27], to remove the short term fluctuations in the data and highlight the longer-term trends to smooth the data. SMA is the unweighted mean of an equal number of data on either side of an intermediate value.



(a) Raw values in sampling-mode (b) Sampling mode values after smoothing

Fig. 2: Data Smoothing

Figure 2 explains the importance of data smoothing technique in the preprocessing of data. Figure 2a is the sampling mode values of hardware event branch instructions for two possible processes in a system. We can easily see that the data is full of short term fluctuations and many redundancies. Discrimination of the two processes seems difficult in this case. There is a need of preprocessing in the dataset. The transformed data of the hardware event after the application of SMA (with a window size equal to 100) is shown in Figure 2b. From this figure, one can discriminate between two processes easily.

Feature Scaling The ranges of values for each feature may vary widely. Many machine learning algorithms find difficulties to optimize the prediction function as they use Euclidean distance as the distance between two points. In that case, if the range of one feature is considerably larger than all other features in the dataset, the distance will be dominated by this particular feature. Thus, all the features need to be normalized so that each of them contributes to the final distance proportionally.

We use one of such feature scaling technique, Standardization [28], to scale the values of the hardware events. Standardization converts the values of each feature to the data of zero mean and unit variance. Standardization sometimes helps machine learning algorithm to converge quickly.

2.4 Anomaly Detection

Anomalies in data are those data that do not satisfy a well-defined notion of normal behavior [29]. Anomalies might be induced in data for different reasons - one of which is the execution of malicious activities. Typically, to get a labeled set of an anomalous dataset which covers all possible types of anomalies is considered to be harder than to get a labeled dataset of normal observations. Moreover, anomalies are often dynamic in nature, which can create a new behavior for which there is no labeled data.

There are different techniques to deal with these anomalies - among which we choose to use a semi-supervised method for anomaly detection. This method

assumes that the training dataset contains the labeled instances of only the normal class. Since we can provide ample training examples of normal system behavior and the label of anomalous behavior is not required, this method works better in our situation.

We used one of the widely used semi-supervised anomaly detection technique One-Class Support Vector Machine (OC-SVM) [30] with non-linear "RBF" kernel in our detection mechanism. One-Class SVM treat the origin as the only member of the second class after transforming the feature via the kernel. Then using "relaxation parameters", it separates the image of the one class from the origin. Then the standard two-class SVM (which we discuss later) techniques are employed.

2.5 Classification

Classification is a type of supervised learning process. The primary purpose of a classification process is to build up a model which can predict a new unlabeled data based on the experience obtained from the previously collected labeled data. Classification consists of two steps - Training and Testing.

In the training phase, a classier is fed with a broad cross-section of data from the original dataset, known as training data, containing a set of features and corresponding label. Here, the label is a function of features, which the classier intends to learn. After the completion of each training phase, the classier is fed with another set of data, known as test data, containing the data not present in training set to estimate the accuracy or effectiveness of the model, referred to as the testing phase of the classier. Once the overall training is complete for the classier, i.e., the mapping function from the feature to label is learned, we can feed a new data with an unknown label to predict the class of the data.

The primary challenge in the classification process is to find the features that distinguish each class very well. Every supervised learning comprises of a common problem known as over-fitting. The model may fit the training data accurately but performs unsatisfactorily on the new unlabeled data. This issue can be prevented partially by applying some regularization techniques.

A major theme in supervised machine learning problems has been having algorithms generalize from the training data rather than simply memorizing it. But there is a subtle issue that plagues all machine learning algorithms, summarized as the "no free lunch theorem." [31]. This theorem states that there is no model which works best for all types of datasets. A good performance of one classifier for a particular dataset could not ensure its good performance for other datasets also.

In this section, we discuss the basic working of some of the widely accepted classifiers, which we have considered to select a best one for our problem via a comparative study.

Random Forest Random Forest [32] is an ensemble learning method that works by constructing multiple decision trees at training time. An ensemble

learning is a process in which multiple classifiers are grouped together, and the decisions of each classifier are combined to get a decision about the class of the input data. In a random forest, the mode of the classes of each decision tree for a particular input decides the class of the data. We briefly discuss the growing of each tree in a random forest as follows -

1. Each tree is trained with approximately $2/3^{\text{rd}}$ of the total training data, which are taken at random but with replacement from the original data.
2. A set of features, with predefined, constant size, is selected at random from the total feature set, and the best split on these is used to split a node. The Gini impurity determines the best split.
3. Each tree is grown to the maximum extent without any pruning. New data is predicted from the trained random forest by taking majority votes from all the trees.

Adaboost The AdaBoost [33] algorithm is used in combination with many other types of weak learning algorithms to improve their performances. The output of the boosted classier is determined by taking a weighted sum of the output of the weak learners. The parameters of each weak learner in AdaBoost is modified for those instances which were misclassified by the previous parameter settings. The AdaBoost algorithm has the benefit that during the training process only those features are selected which contribute more to the predictive power of the model, thereby, reduces the dimensionality and improves the execution time.

Multi-Layer Perceptron A commonly used Multilayer Perceptron (MLP) is a feed-forward artificial neural network model. An MLP contains multiple layers of neurons with an activation function, and each layer fully connects the next layer with numerical values called weights. This activation function maps weighted sum inputs to the output of the neuron. The objective of the MLP is to learn these weights for matching the inputs to the outputs as efficiently as possible.

The input layer of an MLP contains neurons equal to the number of features, and the output layer contains neurons equal to the number of classes. The data is fed to the input layer of the network, and after the feed-forward propagation, the output layer of the network contains a vector of values. The neuron containing maximum value determines the class of the data.

The training process of the perceptron network involves multiple steps of back-propagation [34]. The error in prediction is calculated and using this error the weights of the network is modified by gradient descent algorithm.

Naïve Bayes Naïve Bayes Classier is one of the simple probabilistic classier based on Bayes theorem. This classier has a strong assumption that the features are independent among themselves. From the training data, a likelihood probability is calculated for each feature. For an unknown input data, the posterior probability for each class is calculated using Bayes' theorem. The class having maximum posterior probability value becomes the predicted level for the input data.

Support Vector Machine Support Vector Machine (SVM) [35] is a non-probabilistic linear binary classier, which assigns unknown examples to one class or the other. In SVM the training examples are represented as data points in the feature space and are separated by choosing an optimal hyperplane. The hyperplane is selected in such a way that the distance from the nearest points on the either side of the plane, known as support vectors, to it is maximum.

SVM can be used to classify non-linearly separable data by converting it to linear separable one using some kernel transformation. SVM can solve multi-class classification problem by reducing it to multiple binary class classification problems using one-vs-one or one-vs-rest strategy. In one-vs-one strategy, the class with the maximum number of votes is predicted as the actual class whereas in one-vs-rest strategy the classier with the highest value of output function assigns the class.

2.6 Time-Series Analysis

Time series is a sequence of data points taken successively over equally spaced points time. Hence, it is a sequence of discrete-time data. The analysis of time-series data comprises of extracting meaningful data and other characteristics of the data.

Two time-series data may vary in speed and to measure the similarity between them a sequence alignment approach namely Dynamic Time Warping (DTW) is widely used. In general, DTW calculates an optimal match between two given sequences with certain restrictions. The sequences are "warped" non-linearly in the time dimension to determine a measure of similarity, which is independent of some non-linear variations in the time dimension.

There are many variations of DTW among them we chose to use Fast Dynamic Time Warping (fastDTW) [36] for our experimentation purpose.

3 Proposed Methodology

In this section we introduce the basic operations of our detection mechanism along with a detailed description of different steps.

3.1 Overview

The basic overview of the proposed approach is graphically presented in Figure 3. Our detection technique consists of three major steps. These three phases are briefly discussed as follows -

1. **Anomaly Detection**: In this step, we generate significant volumes of low-level data at a chosen frequency by granular profiling the performance counters with perf tool. Then this data are processed through some data preprocessing steps and passed to the anomaly detector module, which is nothing but a One-Class Support Vector Machine (OC-SVM). This module outputs whether the input data is an anomaly or a normal, benign process.

2. **Classification**: If there is any abnormality, the abnormal process is then passed to the pre-trained classifier, in this step, to know the category of abnormality. This step outputs the possible type (like cache-based or branch-prediction based) of side-channel-attack for the abnormal process.
3. **Correlation Module**: This step profiles the appropriate hardware or software event, determined from the classification step, for the anomaly process and tries to find the similarity with this to the profile of the same event for the encryption algorithm. The similarity between these two temporal sequences with different spreads is determined through the Fast Dynamic Time Warping (fast-DTW) algorithm. If the similarity is more than a predefined threshold value, then we can term the anomalous process as a side-channel-attack.

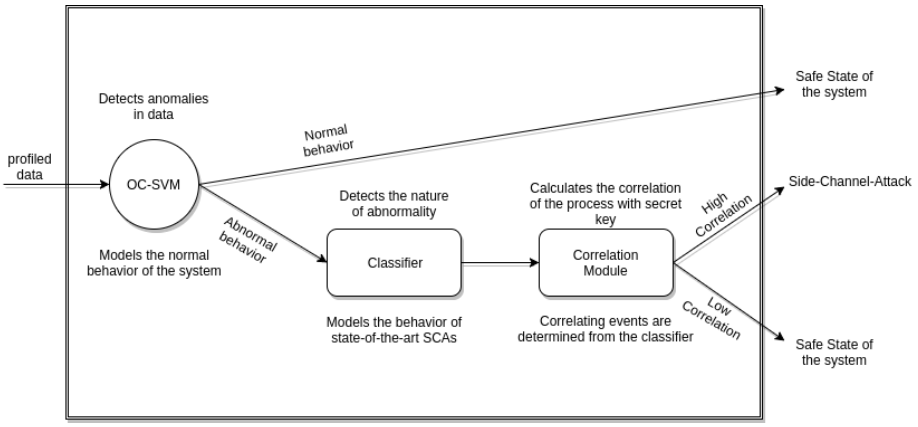


Fig. 3: Basic overview of the detection mechanism

These steps are explained below with a lot more details in the following subsections.

3.2 First Phase: Anomaly Detection

We used One Class Support Vector Model to deal with the anomalies in this step. This module is particularly helpful in scenarios where a lot of "normal" data are available but not many cases of anomalies that we try to detect. This anomaly detection model needs to be trained with a dataset that contains all or most normal data.

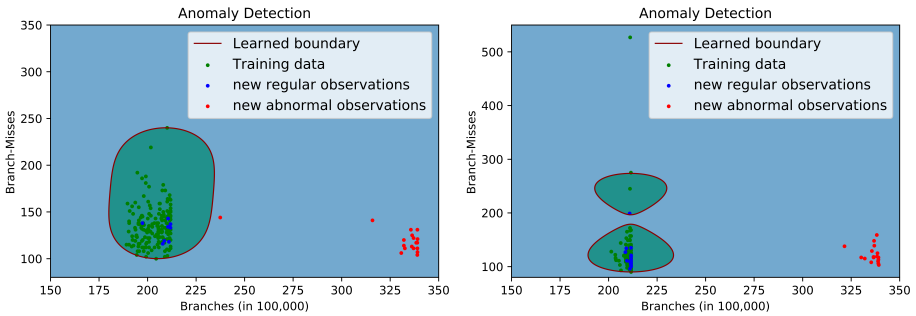
The data collection for the normal behavior of a system can be achieved by profiling some frequently used standard Linux commands like `cd`, `mv`, `cp`, `apt-get`, `bzip2`, `gzip`, `echo`, `grep`, `passwd`, `pwd`, `ls`, and so on. We collect the data using user level Linux perf tool in sampling mode of operation with

a chosen time interval. We used most significant hardware events to monitor which are responsible for differentiating between a benign and an attack process as described in *Case Study: I.*

This collection of data then works as a training dataset for the OC-SVM. The OC-SVM then infers the properties of normal cases and from these properties can predict which examples are unlike the normal examples, i.e., an anomalous example using the learned decision boundary.

The hyper-parameters of the learning algorithm like *fraction of outliers* (η), *stopping tolerance* (ϵ) are tuned by iterating over the training process, and the best setting is selected based on the optimal result.

Case Study: II We try to explain the decision boundary learned by the anomaly detection module with the form of a case study. We implemented the anomaly detection model on both the experimental setup and trained it with the similar approach as mentioned above. We considered only two events to monitor in parallel, namely branches and branch-misses, for the sake of 2-D representation and a better understanding of the decision boundary. The learned decision boundary is shown in Figure 4a for the first experimental setup an in Figure 4b the for second experimental setup with a thick red line with the training examples which are shown in the figure as green dots.



(a) Anomaly detection in first experimental setup

(b) Anomaly detection in second experimental setup

Fig. 4: Anomaly Detection

After the training when the anomaly detection module works in real time, the classification of the new set of real-time data are also shown in the Figure 4. The blue dots in the figures indicate the new set of normal execution data, all of which are correctly detected within the decision boundary. Then we tried to mount a branch-prediction based side-channel-attack code on the system, and rightfully most of the examples of execution traces of this attack lay outside the decision boundary and acted as anomalies in the system, which are represented

as red dots in the figures. Because of the branch prediction based attack the hardware events like branches and branch-misses will be enough to determine the anomalous behavior of the process, which is quite clear from the Figure 4.

We can also see from Figure 4b that OC-SVM is very powerful against training set errors. The training set, in this case, contains a noisy sample which it had handled successfully and considered only the most usual training examples.

3.3 Second Phase: Classification

In the second step of our proposed approach, we build a classifier to model the behaviors of different side-channel-attacks. For this purpose, we collect the execution traces of various hardware events of different state-of-the-art side-channel-attacks and learn their behaviors with a classification task. For example, all the cache-based attacks will affect the hardware events such as LLC Access, LLC Miss, etc. So, abnormalities in the cache access pattern from the normal behavior can term a process as a potential cache based attack. Similarly, for branch prediction based attacks abnormalities can be seen in branch instructions, branch-misses, etc. The Same approach is followed for different side-channel-attacks to learn their behaviors.

As mentioned previously, because of the "No Free Lunch" theorem, there is no one supervised machine learning model that works best for every problem. The assumptions of a great model for one problem may not hold for another problem. We train various widely used classifiers, as mentioned previously, for the same dataset and find the one which works best for our problem.

Case Study: III We try to give focus on the idea of comparing the different classifiers and selecting the best one for this phase with a simple case study. Here we considered three different types of attacks for our demonstration:

1. Two Cache-timing based attack on AES and Clefia respectively.
2. Branch-prediction based attack on RSA.

In this scenario, we considered only two hardware events, namely branch misses and instruction count, for the sake of representation of the decision boundaries in a 2-D plane and for a clear understanding. In Figure 5 the decision boundaries are represented for four different classifiers, namely Adaboost, Random Forest, Naïve Bayes and SVM.

The horizontal axis represents the number of branch-misses (in thousands) whereas the vertical axis represents the number of instructions (in millions). The red squares represent the training set of cache-based attacks in the figure, and blue triangles represent the training set of Branch-prediction based Attack. Accordingly, the decision boundaries are shown in the figures.

We can rightly observe from Figure 5 that for cache-based attack, number of instruction counts are high because of cache misses and for branch prediction based attack branch-misses are high.

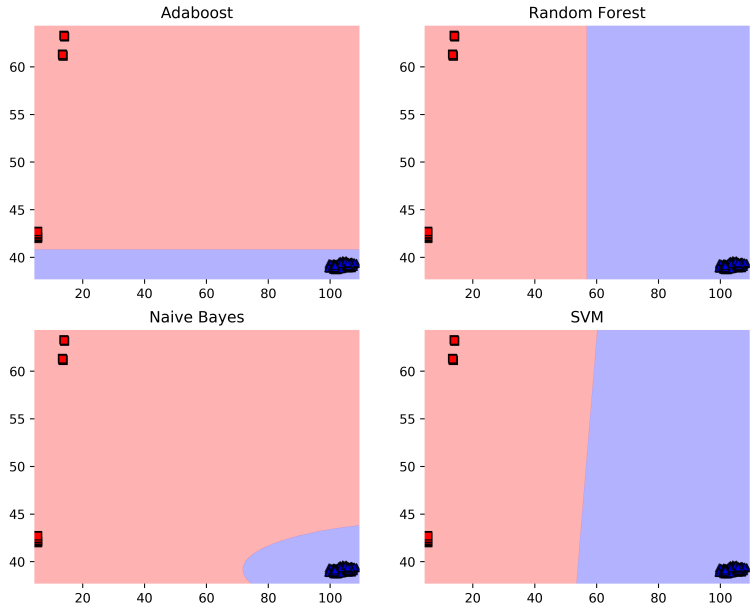


Fig. 5: Decision Boundary of different Classifiers on some of the side-channel-attacks

As we can see, based on selecting the classifier, for a new point the decision may change. We could get a clear picture of the behaviors of different types attacks once we consider the best classifier after comparing their accuracies.

The accuracies for different classifiers used in this step is mentioned in details in Results and Discussion section.

3.4 Third Phase: Correlation

The final and most important step of our proposed architecture is the correlation module. In this step, we try to correlate the appropriate hardware performance counter values to the secret key of the encryption algorithm. The appropriate hardware event can be obtained by observing the result of the classification step. If the behavior of the anomalous process is classified as a cache-based attack, then the execution trace of the hardware events responsible for those types of attacks needs to be correlated with the secret key for a successful attack.

First, we generate the execution trace of the encryption algorithm for a particular hardware event (say T_1). Next, we observe the execution trace of the anomalous process for the same hardware event (say T_2). These two traces are

nothing but two different temporal sequences. Since the side-channels-attack work by continuously monitoring the encryption algorithm, in this case, it can be said that T_1 is contained in T_2, but with some added noise of execution because of the attacker's own code.

We apply a widely used temporal sequence alignment problem, known as Fast Dynamic Time Warp (fast-DTW), as mentioned previously, to find the alignment cost between T_1 and T_2. A low value of alignment cost signifies the high amount of matching between these two sequences which in turn means a high correlation between secret key and the execution trace of the anomalous process. High correlation with the secret key helps to identify the true positiveness of the side-channel-attack process. A process having a low correlation value can be treated as a benign process, thereby reducing the false positives. Extensive experimentation determines the threshold of the alignment cost.

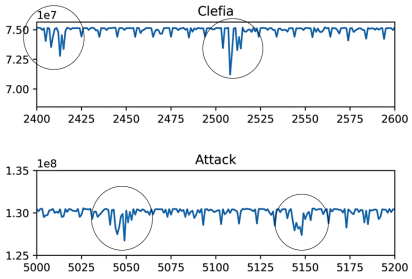
Case Study: IV We present here a simple case study of a side-channel-attack on Clefia [13], a Block Cipher developed by Sony. In this case study, we consider the following three situations -

- A side-channel-attack is executing in the system to compromise the secret key of Clefia, which is previously detected as an anomaly and one of the hardware events causing the anomaly is instruction counts.
- Same situation as above but in a multi-user environment, i.e., user_1 is executing Clefia with a secret key k_1, but the side-channel-attack is executing to compromise the secret key k_2 of user_2.
- A Firefox application detected as an anomaly in the first step, and the hardware event causing the abnormality is instruction count.

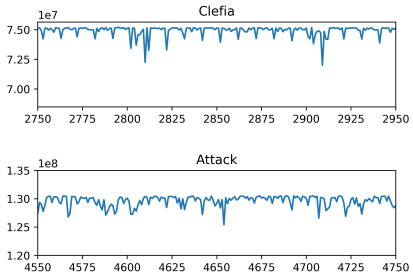
In all the three cases, we applied our correlation module by observing the instruction execution traces (this event was among one of the anomalies as reported by the classifier since the classifier was already trained with behaviors of possible attacks on Clefia) with a fixed time interval. The results of minimum cost alignments, with window size = 1, are presented in Figure 6.

The alignment cost for Figure 6a is the lowest, which is 2734.785 precisely considering the Manhattan Distance as a distance measure. The alignment cost for Figure 6b and Figure6c are 4159.338 and 16771.428 respectively. This alignment cost perfectly reflects the correlation of the process traces with the secret key. For a predefined threshold value, the first scenario will be correctly treated as a case of side-channel-attack by our detection system.

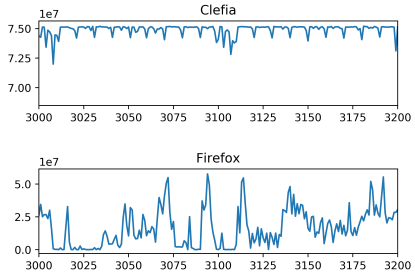
For an unknown "intelligent" attack belonging to the category of cache-based attack, or branch prediction based attack, or a DRAM based attack, though its behavior is not trained to the classifier, it will be successfully detected based on its abnormal hardware event accesses (since the classifier is already trained with attacks of the same category though not the same). Once the category is determined, the correlation module can be used to measure the amount of correlation with its hardware event trace with the secret key of encryption, and thus can be successfully detected as an attack process.



(a) Correlation for first situation



(b) Correlation for second situation



(c) Correlation for third situation

Fig. 6: Correlation with key for different situations

4 Results and Discussion

In this section, we focus on the performance and qualitative evaluation of our detection mechanism.

4.1 Experimental Setup

We perform all the experiments in two different environments to get a generalized performance measure of the detection scheme.

1. Setup 1: Optiplex 9020 Desktop equipped with Intel Core i5-4570 CPU with 3.20GHz clock frequency, 4GB RAM running Ubuntu 16.04.1 having Kernel Linux 4.8.0-49-generic.
2. Setup 2: GPU Server powered by Intel Xeon E5-2630 v3 CPU with the 2.40GHz clock frequency and 256GB RAM.

4.2 Accuracy of different phases

Here we discuss the detection accuracies of each of the phase individually.

Anomaly Detection Phase In this phase to model the normal behavior of the anomaly detection module, we select most common Linux commands and utilities [37]. We measure the True Positive (TP) (an anomaly is correctly identified), False Positive (FP) (a regular process is identified as an anomaly), True Negative (TN) (a regular process doesn't get identified as an anomaly) and False Negative (FN) (an anomaly detected as regular process) from our anomaly detection module. In ideal scenario TP, TN should be 100% and FP, FN should be 0%. We also calculated the accuracy of the system as:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{1}$$

We consider all the attacks mentioned in the preliminaries along with some micro-architecture intensive sample programs which are not attacks (having high cache access or high branch-misses) but a simple benign process to test whether our detection module can capture any anomalies in the system.

We considered two profiling granularity of 1ms and 10ms, and for each frequency, we executed the detection module 100 times with randomly checking the normal process, attack process and sample micro-architecture intensive programs in each run and report the accuracy of the system on both the experimental setup in Table 1 and Table 2 respectively.

Table 1: Sampling granularity of 1ms

Setup 1			Setup 2		
	Positive	Negative		Positive	Negative
True	35	65	True	47	48
False	0	0	False	5	0

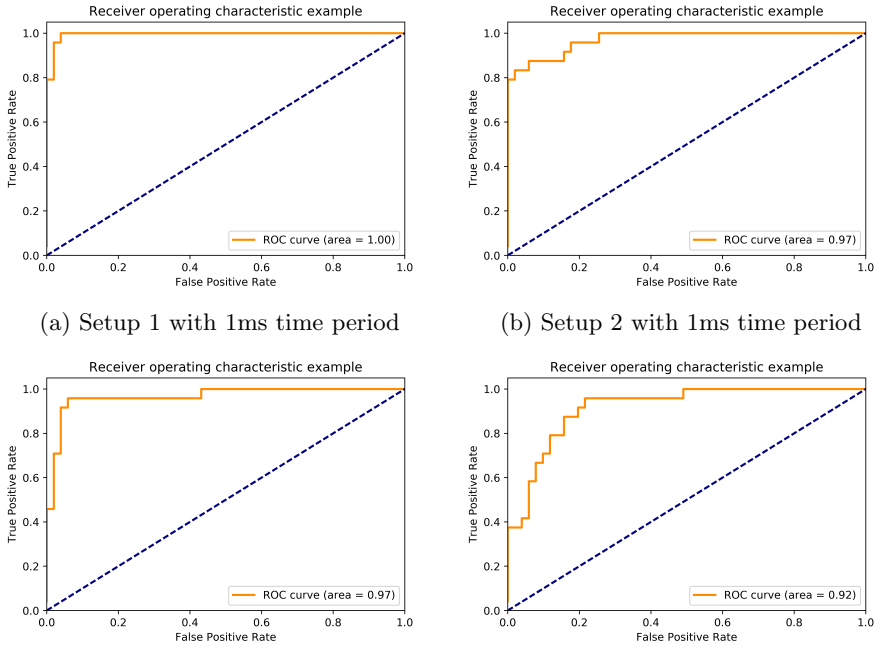
Table 2: Sampling granularity of 10ms

Setup 1			Setup 2		
	Positive	Negative		Positive	Negative
True	51	46	True	35	57
False	3	0	False	7	1

Accuracies in Table 1 are 100% and 97% and in Table 2 are 97% and 92% respectively.

The corresponding ROC Curves for all the above four cases are shown in Figure 7.

From the above tables and figures, we can easily say that the anomaly detection module performs well in real-time for both the setups. We verified our detection technique with different sampling granularities, and also observed that anomaly detection method works better for 1ms profiling frequency.



(a) Setup 1 with 1ms time period

(b) Setup 2 with 1ms time period

(c) Setup 1 with 10ms time period

(d) Setup 2 with 10ms time period

Fig. 7: Receiver Operating Characteristic Curves

Classification Phase We have considered three different categories of attacks in the previous sections - cache-based, branch-based and DRAM based. We train each classifier, as mentioned previously, with the behavior of these attacks to correctly model the generalized exploitation of hardware events. Like the previous phase, in this phase also we consider some micro-architecture intensive sample code to check whether the classifier correctly classifies the abnormality in the hardware events.

The classification algorithms were tested by cross-validation method on the dataset of behaviors of the side-channel-attacks mentioned in the preliminaries section. The accuracies of different classifiers for both the setups and for two different sampling frequencies are mentioned in Table 3 and Table 4.

Table 3: Classification Accuracy for Setup 1

	Naïve Bayes	MLP	SVM	Adaboost	Random Forest
1ms	81.01%	82.1%	91.0%	99.1%	99.2%
10ms	78.3%	81.4%	89.57%	99.0%	98.7%

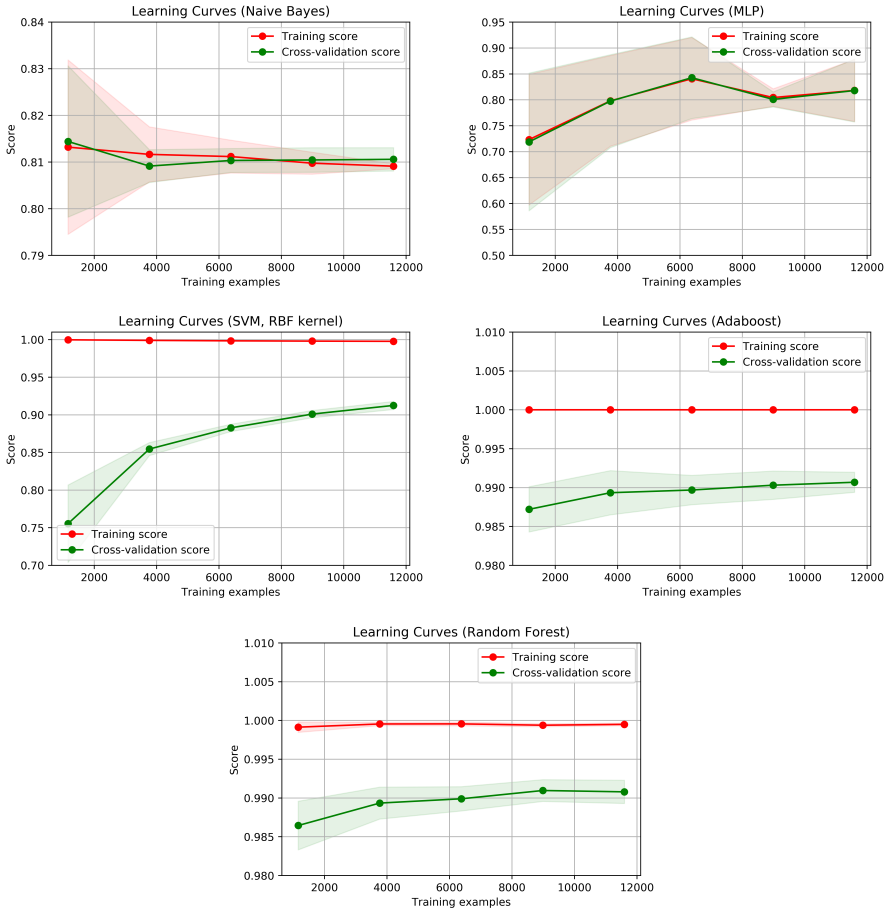


Fig. 8: Learning Curves for the classifiers in Setup 1 with 1ms granular sampling

The learning curves for the classifiers in Setup 1 with 1ms of granular profiling are shown in Figure 8. The red line indicates the training score for the classifier, whereas, the green line signifies the cross-validation score. We can easily see that Random Forest and Adaboost outperform the other classifiers for our problem. Also, we can observe that, with lower profiling granularity, accuracies of the classifiers increase, it is because the more granular the data, the more distinctive behavior can be seen in the attacks.

The learning curves for the classifiers in Setup 2 with 1ms of granular profiling are shown in Figure 9. The red line indicates the training score for the classifier, whereas, the green line signifies the cross-validation score. Here also we can easily see that Random Forest and Adaboost outperform the other classifiers. Also, we can observe that, with lower profiling granularity, accuracies of the classifiers increase.

Table 4: Classification Accuracy for Setup 2

	Naïve Bayes	MLP	SVM	Adaboost	Random Forest
1ms	81.02%	62%	91%	99.1%	99.3%
10ms	80.7%	59.4%	90.1%	99.2%	99.2%

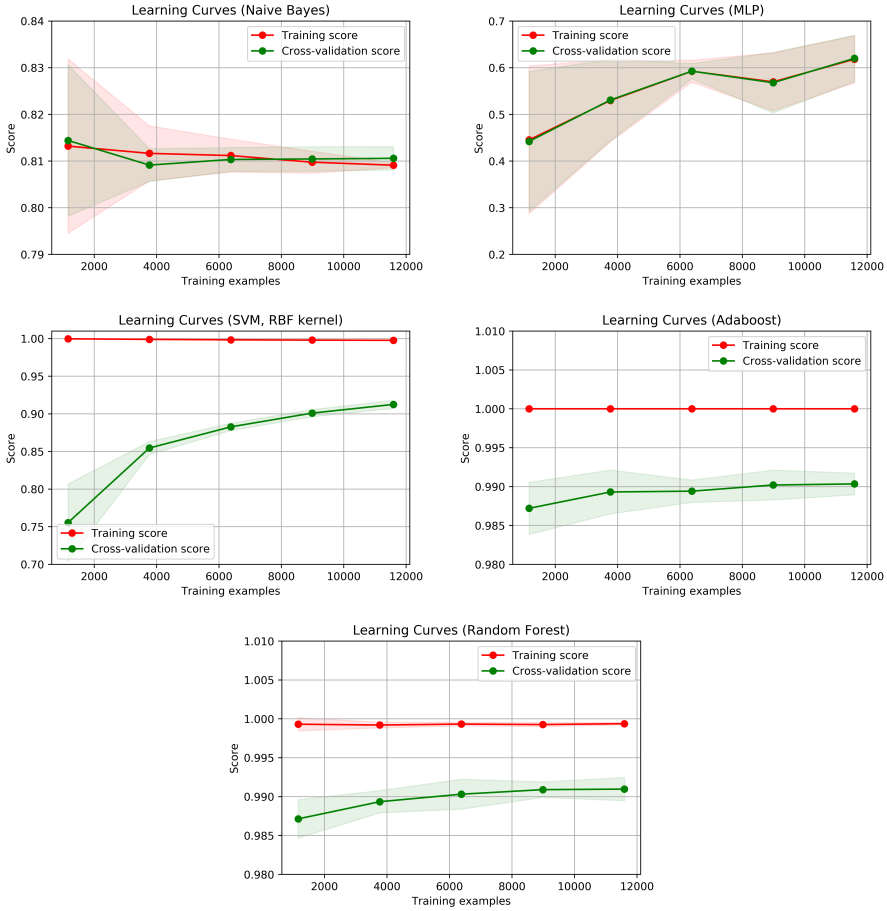


Fig. 9: Learning Curves for the classifiers in Setup 2 with 1ms granular sampling

So, observing the accuracies of different classifiers in two different setups with two different sampling frequencies we choose to use Adaboost or Random Forest as a classifier in our detection methodology.

Correlation Phase In this phase, the accuracy depends on the window size as selected for the fastDTW algorithm to find optimal alignment between two

time-series sequences of hardware events. Here we run the experiment as mentioned in *Case Study: IV* for the two experimental setups and with two sampling frequencies along with some micro-architecture intensive random benign sample programs which behave similarly as that of the attack. For a predefined threshold value, we conduct the experiment for different window length and report the accuracy of the module in Table 5. The accuracy is calculated according to equation (1).

Table 5: Accuracy of Correlation Module for different window size

Setup 1			
	w=1	w=3	w=5
1ms	76%	79%	83%
10ms	63%	75%	82%

Setup 2			
	w=1	w=3	w=5
1ms	57%	67%	74%
10ms	62%	63%	69%

We can observe from Table 5 that accuracy of the correlation module is best in Setup 1 when profiling at 1ms time interval for window size = 5 and with the same granularity and window size it is also best in Setup 2. The low accuracy in the second setup is because of the presence of background noise in the system.

5 Future Work

The possible future work in this direction would be to retrain the classifier after each successful attack detection to increase the accuracy and accommodate new types of attack. The accuracy of detection against the training overhead would be our next research work.

Future work on this needs to focus on improving the accuracy of the correlation based module in the presence of large background noise, to make it more feasible to real-time implementation.

Another future direction could be to implement the proposed detection method in Cloud environment where the possibility of existence of side channel attacks is very high.

6 Conclusion

This paper proposed a Machine Learning based three phase approach to detect different micro-architectural side-channel-attacks. The main working of this method is to profile hardware events at lower granularity and process it with machine learning algorithms. This approach can also deal with false positives and unknown attack with a novel technique which correlates its execution trace with the secret encryption key, using a time-series approach. The results established the fact with superiority.

References

1. Bonneau, J., Mironov, I.: Cache-collision timing attacks against AES. In: Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings. (2006) 201–215
2. Aciiçmez, O., Koç, Ç.K., Seifert, J.: Predicting secret keys via branch prediction. In: Topics in Cryptology - CT-RSA 2007, The Cryptographers' Track at the RSA Conference 2007, San Francisco, CA, USA, February 5-9, 2007, Proceedings. (2007) 225–242
3. Uhsadel, L., Georges, A., Verbauwhede, I.: Exploiting hardware performance counters. In: Fifth International Workshop on Fault Diagnosis and Tolerance in Cryptography, 2008, FDTC 2008, Washington, DC, USA, 10 August 2008. (2008) 59–67
4. Bhattacharya, S., Mukhopadhyay, D.: Who watches the watchmen?: Utilizing performance monitors for compromising keys of RSA on intel platforms. In: Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings. (2015) 248–266
5. Qian Ge, Yuval Yarom, F.L., Heiser, G.: Contemporary processors are leaky - and there's nothing you can do about it (feb 2017)
6. Liu, F., Lee, R.B.: Random fill cache architecture. In: 47th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2014, Cambridge, United Kingdom, December 13-17, 2014. (2014) 203–215
7. Domnitser, L., Jaleel, A., Loew, J., Abu-Ghazaleh, N.B., Ponomarev, D.: Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks. TACO **8**(4) (2012) 35:1–35:21
8. Martin, R., Demme, J., Sethumadhavan, S.: Timewarp: Rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks. In: Proceedings of the 39th Annual International Symposium on Computer Architecture. ISCA '12, Washington, DC, USA, IEEE Computer Society (2012) 118–129
9. Chiappetta, M., Savas, E., Yilmaz, C.: Real time detection of cache-based side-channel attacks using hardware performance counters. Applied Soft Computing **49** (2016) 1162 – 1174
10. Demme, J., Maycock, M., Schmitz, J., Tang, A., Waksman, A., Sethumadhavan, S., Stolfo, S.: On the feasibility of online malware detection with performance counters. In: Proceedings of the 40th Annual International Symposium on Computer Architecture. ISCA '13, New York, NY, USA, ACM (2013) 559–570
11. Zhang, T., Zhang, Y., Lee, R.B.: Cloudradar: A real-time side-channel attack detection system in clouds. In: Research in Attacks, Intrusions, and Defenses - 19th International Symposium, RAID 2016, Paris, France, September 19-21, 2016, Proceedings. (2016) 118–140
12. Bernstein, D.J.: Cache-timing attacks on aes. Technical report (2005)
13. Rebeiro, C., Mukhopadhyay, D., Takahashi, J., Fukunaga, T. In: Cache Timing Attacks on Clefia. Springer Berlin Heidelberg, Berlin, Heidelberg (2009) 104–118
14. Wikipedia: Rowhammer wikipedia page, https://en.wikipedia.org/wiki/Rowhammer (2016)
15. Kim, Y., Daly, R., Kim, J., Fallin, C., Lee, J., Lee, D., Wilkerson, C., Lai, K., Mutlu, O.: Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In: ACM/IEEE 41st International Symposium on Computer Architecture, ISCA 2014, Minneapolis, MN, USA, June 14-18, 2014. (2014) 361–372

16. Huang, R., Yang, H., Chao, M.C., Lin, S.: Alternate hammering test for application-specific drams and an industrial case study. In: The 49th Annual Design Automation Conference 2012, DAC '12, San Francisco, CA, USA, June 3-7, 2012. (2012) 1012–1017
17. Kim, D., Nair, P.J., Qureshi, M.K.: Architectural support for mitigating row hammering in DRAM memories. Computer Architecture Letters **14**(1) (2015) 9–12
18. Wang, X., Karri, R.: Numchecker: detecting kernel control-flow modifying rootkits by using hardware performance counters. In: The 50th Annual Design Automation Conference 2013, DAC '13, Austin, TX, USA, May 29 - June 07, 2013. (2013) 79:1–79:7
19. Wang, X., Konstantinou, C., Maniatakos, M., Karri, R.: Confirm: Detecting firmware modifications in embedded systems using hardware performance counters. In: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2015, Austin, TX, USA, November 2-6, 2015. (2015) 544–551
20. Wang, X., Karri, R.: Reusing hardware performance counters to detect and identify kernel control-flow modifying rootkits. IEEE Trans. on CAD of Integrated Circuits and Systems **35**(3) (2016) 485–498
21. Malone, C., Zahran, M., Karri, R.: Are hardware performance counters a cost effective way for integrity checking of programs. In: Proceedings of the sixth ACM workshop on Scalable trusted computing, STC@CCS 2011, Chicago, Illinois, USA, October 17, 2011. (2011) 71–76
22. Performance Application Programming Interface: (2016)
23. OProfile: (2015)
24. perf: Linux profiling with performance counters: (2015)
25. Intel 64 & IA-32 Architectures Software Developer's Manual Volume 3 (3A, 3B, 3C & 3D): System Programming Guide: (2010)
26. Meinshausen, N., Bhlmann, P.: Stability selection. Journal of the Royal Statistical Society: Series B (Statistical Methodology) **72**(4) (2010) 417–473
27. Chou, Y.L.: Statistical Analysis. Holt International (1975)
28. Theodoridis, S., Koutroumbas, K.: Pattern Recognition. Elsevier Academic Press (2009)
29. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey. ACM Comput. Surv. **41**(3) (July 2009) 15:1–15:58
30. Schölkopf, B., Platt, J.C., Shawe-Taylor, J.C., Smola, A.J., Williamson, R.C.: Estimating the support of a high-dimensional distribution. Neural Comput. **13**(7) (July 2001) 1443–1471
31. Wolpert, D.H.: The lack of a priori distinctions between learning algorithms. Neural Computation **8**(7) (Oct 1996) 1341–1390
32. Ho, T.K.: Random decision forests. In: Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1. ICDAR '95, Washington, DC, USA, IEEE Computer Society (1995) 278–
33. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. J. Comput. Syst. Sci. **55**(1) (August 1997) 119–139
34. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Neurocomputing: Foundations of research. MIT Press, Cambridge, MA, USA (1988) 696–699
35. Cortes, C., Vapnik, V.: Support-vector networks. Mach. Learn. **20**(3) (September 1995) 273–297
36. Salvador, S., Chan, P.: Toward accurate dynamic time warping in linear time and space. Intell. Data Anal. **11**(5) (October 2007) 561–580

37. Natarajan, R.: 50 most frequently used unix / linux commands (with examples) (nov 2010)