NIGHTs-WATCH: A Cache-based Side-channel Intrusion Detector Using Hardware Performance Counters

Maria Mushtaq, Ayaz Akram, Muhammad Khurram Bhatti, Maham Chaudhry, Vianney Lapotre, Guy Gogniat

NIGHTs-WATCH: A Cache-Based Side-Channel Intrusion Detector using Hardware Performance Counters

Maria Mushtaq, Ayaz Akram, Muhammad Khurram Bhatti, Maham Chaudhry
Vianney Lapotre, and Guy Gogniat

ABSTRACT

This paper presents a novel run-time detection mechanism, called NIGHTs-WATCH, for access-driven cache-based Side-Channel Attacks (SCAs). It comprises of multiple machine learning models, which use real-time data from hardware performance counters for detection. We perform experiments with two state-of-the-art SCAs (Flush+Reload and Flush+Flush) to demonstrate the detection capability and effectiveness of NIGHTs-WATCH. we provide experimental evaluation using realistic system load conditions and analyze results on detection accuracy, speed, system-wide performance overhead and confusion matrix for used models. Our results show detection accuracy of 99.51%, 99.50% and 99.44% for F+R attack in case of no load, average load and full conditions, respectively, with performance overhead of $< 2\%$ at the highest detection speed, i.e., within 1% completion of a single RSA encryption round. In case of Flush+Flush, our results show 99.97%, 98.74% and 95.20% detection accuracy for no load, average load and full conditions, respectively, with performance overhead of $< 2\%$ at the highest detection speed, i.e., within 12.5% completion of 400 AES encryption rounds needed to complete the attack. NIGHTs-WATCH shows considerably high detection efficiency under variable system load conditions.

1. INTRODUCTION

With technologies like Blockchain, Cyber-Physical Systems (CPS), Cloud Computing Infrastructure-as-a-Service (IaaS), and Internet-of-Things (IoTs) in practice, the age of digital disruption has a renewed challenge of security and privacy. It has become a first-case design constraint for many application domains involving third-party processing of private data. Malicious software and privacy intrusion techniques have existed since long and their proliferation has seen rise in recent years. In an effort to address this problem, researchers have devised numerous prevention and detection schemes, each focused on a certain aspect of the malicious behavior. While it is infeasible to prevent bad actors from attacking, schemes that target the detection of and prevention from malicious processes do exist.

Side-channel attacks are a specific class of attacks that mainly targets cryptographic implementations [1], [2], [3] for unauthorized retrieval of information. The threat of side channel leakage imposes a serious concern to data privacy as it can *break* the otherwise theoretically sound cryptographic algorithms at implementation-level. Modern-day processors do extensive sharing and de-duplication for performance benefits. Cache-based side channel attacks, a sub-class of SCAs, are able to retrieve information from systems by exploiting vulnerabilities in shared caches [4], [5]. Such attacks rely on the presence of assembly instructions to fully or partially manoeuvre the state of shared caches. For instance, in Intel CPUs, that feature is available through the *CLFLUSH* instruction. To intercept these attacks between different processes, we need to constitute a CPU architecture that restricts the usage of these instructions or disables memory optimization for such features. In both the cases, there is an overhead for performance and cost to develop such architectures. In the recent past, there are many successful implementations of cache-based SCAs. For instance, SCA targeting DES implementation has been demonstrated in [6], while authors in [3] present Evit+Time and Prime+Probe techniques on AES implementation. Recently, [1] presented a cache-based SCA devising a new technique called Flush+Reload (F+R), to retrieve private exponent key of GNU privacy guard's implementation on RSA. Researchers in [7] also used F+R to extract secret key in AES encryption. Later, using the same principle of F+R, authors in [8] proposed a stealthier technique called Flush+Flush (F+F) on AES crypto-system.

Numerous software and hardware based protection techniques have also been proposed against specific type of threat generated by SCAs [9], [10]. Such attacks can be prevented at various levels such as system-level, hardware-level, and application-level [9]. At the system level, physical and logical isolation approaches exist [11], [12]. At the hardware level, mitigation techniques are rather difficult as they are not applicable on commodity systems. Hardware solutions, nevertheless, suggest to have new secure caches, changes in prefetching policies and either randomization or complete removal of cache interference [13]. At the application level, the proposed countermeasure techniques tend to target the source of information leakage and mitigate it [14].

Despite valiant efforts, mitigation techniques against malicious software and side-channel attacks are not perfect. Mitigation techniques generally protect against specific vulnerability and an *all-weather* protection against such attacks is often costly in terms of performance. Moreover, evidence suggest that attacks are becoming sophisticated and stealthier. Therefore, detection techniques can be used as a *first line of defense* against such attacks. For detection-based prevention strategy to be effective, detection need to be highly accurate, incur minimum system overhead at run-time, and capable of early-stage detection of attacks, i.e., before they complete.

In this paper, we address the problem of accurate & early detection of cache-based SCAs at *run-time*. We present a novel detection mechanism that uses machine learning models. These models use data from Hardware Performance Counters (HPCs), in near real-time, representing the pattern of memory accesses generated by data-dependent crypto-

graphic operations being carried out by underlying hardware. Specifically, the main contributions of this paper are as follows:

1. We propose NIGHTs-WATCH, a novel run-time & early detection mechanism for access-driven cache-based SCAs, which uses three machine learning models and real-time data from hardware performance counters.
2. We demonstrate successful detection of state-of-the-art cache-based SCAs (Flush+Reload & Flush+Flush) under *realistic* system load conditions, i.e., under No Load, Average Load, and Full Load conditions, with considerably high detection accuracy, high speed, and minimal performance overheads. Testing under realistic load conditions improves its deployability.
3. We propose, to the best of our knowledge, a first-ever successful detection of Flush+Flush category of stealthier SCAs using NIGHTs-WATCH with an accuracy of up to 99.97% (best-case) and performance overhead of $< 2\%$ while relying on selected hardware events.
4. We provide experimental results and discussion on detection accuracy, system-wide performance overhead, detection speed, and confusion matrix for selected machine learning models in detail. Also, we discuss limitations related to the use of HPCs.

Rest of the paper is organized as follows. Section 2 presents necessary background & related work on cache-based SCAs, mitigation and detection mechanisms. Section 3 presents NIGHTs-WATCH in detail. Section 4 provides experimental evaluation and discussion. Section 5 concludes this paper.

2. BACKGROUND AND RELATED WORK

This section provides necessary background on cache-based SCAs, their mitigation techniques, and the support for hardware events using HPCs.

2.1 Cache-based SCAs and Mitigation

Side-channel attacks target micro-architecture of platforms and can collapse the strongest of crypto-systems like AES, RSA, and ECC etc. The pattern of memory accesses and timing generated by data-dependent cryptographic operations is a major source of information revelation in these attacks. Over the last decade, numerous cache-based SCAs have been proposed, followed by their mitigation techniques. Both Instruction and Data Caches are indeed important sources of vulnerabilities for micro-architectural attacks [15]. Some examples of such attacks are Prime & Probe, Flush+Reload, Flush+Flush and Evict & Time, [1], [3], [8].

Some practical solutions to mitigate such SCAs include; disabling hardware threading [16], auditing [17], cache flushing [18], cache coloring [19], scheduling-based obfuscation [20], and hardware cache partitioning [21]. These techniques are often designed to offer protection against a specific information leakage channel and incur significant system overhead by blowing up the code size and cutting down the performance by many-folds. A common protection approach is to ensure that the memory accesses generated by the cryptographic operations are not dependent on the data being processed. Cache partitioning is another popular solution as a countermeasure against cache-based SCAs [14]. These strategies, however, introduce significant performance degradation because of cache reservation. Since, applying mitigation techniques in all cases is expensive, therefore, detection methods can help applying countermeasure on need-basis.

2.2 Selected Cache-based SCAs

We have selected two state-of-the-art access-driven cache-based SCAs to demonstrate the effectiveness of NIGHTs-WATCH. We elaborate these attacks for the reader here.

2.2.1 Flush+Reload Attack on RSA

Flush+Reload attack, proposed in [1] on RSA cryptosystem, fundamentally exploits the security vulnerability introduced due to demand-fetch policy of caches by measuring the timing difference when data is loaded in response to a cache *hit* compared to when it is loaded in response to a cache *miss*. F+R attack relies mainly on two factors: 1) the ability of processes to *flush* any cache line by virtual address on the target architecture and 2) the existence of shared virtual memory (shared libraries). A malicious process shares memory with a *victim* process while executing on Intel's *x*86 architecture having inclusive caches. One round of F+R attack consists of three distinct phases: *Flush*, *Wait*, and *Reload*. In the first phase, attacker, while targeting shared Last-Level Cache (LLC), first evicts a shared cache line of interest by either using *eviction through contention* mechanism or dedicated instructions. Authors in [1] have used Intel's *clflush* instruction for this purpose. In the second phase, attacker waits for a prefixed duration and allows victim to execute. In the last phase, attacker reloads the same cache line it had evicted earlier and measures its loading time. A slow reload would indicate that attacker is the only process touching that particular cache line, thus every reload will cause a cache miss and will take more time to fetch instruction from main memory. A faster reload, however, indicates that the victim process also touched the cache line of interest while attacker was in its wait phase, thus a cache hit for attacker and it takes less time to fetch instruction from cache. The attack exploits inclusive property of Intel's caches to remove instructions from all levels while targeting LLC.

2.2.2 Flush+Flush Attack on AES

Flush+Flush attack, proposed in [8], is also based on similar approach as F+R attack, except that it is considerably stealthier. In contrast to other cache attacks, F+F does not perform any memory accesses, which means it does not allow any additional cache misses and minimal cache hits due to attacker process. It is dependent on precise cache timing only, which makes it more stealthy and efficient attack. Proposed detection methods to-date are not able to detect such stealthy attack techniques. F+F uses the same hardware and software vulnerabilities exploited in F+R except the fact that F+F attacks OpenSSL T-Table-based AES Implementation. F+F works across cores and in virtualized environments. This is also a three phase attack. In the first phase, attack executes an infinite loop executing *clflush* instruction to flush a shared cache line. In the second phase, the attacker process measures the execution time of clflush instruction itself every time it is being applied on a shared cache line. In the third phase, attacker uses the measured time in second phase to determine whether the that shared memory line, being flushed in first phase, had been cached or not by the victim process. Since the attacker itself is not loading any memory line into the cache, therefore, if any other process fetches shared cache line into the cache, the attacker process will come to know about it and in every loop it will simply flush the memory line again.

Authors in [8] have claimed to detect their own attack using hardware events in linux *Perf_event_open* syscall interface. They documented that their attack is not detectable using 24 hardware events available with Linux syscall interface.

2.3 Hardware Performance Counters (HPCs)

Hardware Performance Counters (HPCs) are special hardware registers used primarily for the performance monitoring & subsequent tuning, debugging, and identifying bottlenecks in program execution. HPC registers reveal run-time behavioral information of software using specific hardware events such as; cache references, cache misses & hits, branch miss-predictions, retired instructions, and CPU cycles etc. These events use dedicated hardware, therefore, they can be accessed very fast without affecting target software. Moreover, no source code modification is required to access them. Modern processors, such as Intel family [22], offer hundreds of events that can be tracked. These events vary in numbers, types, and methods in which they can be used across various architectures. One limitation, however, is that these HPC registers are limited due to which only a very small number of events can be monitored concurrently. HPCs have been used in recent research work [23], [24] for dynamic profiling and intrusion detection.

2.4 Related Work on Detection

This section summarizes the state-of-the-art on side-channel intrusion detection mechanisms (with and without machine learning approaches) and discusses their advantages and limitations. In recent years, some research work such as [8], [24], [25], [26], [27], [28], [29], [30] has been done to detect side channels by implementing user-level processes to observe the execution of other processes or by providing light-weight patches in operating system. These solutions do not modify underlying hardware. Authors in [25], [28] present mechanisms related to signature-based anomaly detection and pattern recognition for known attacks. Authors in [30] propose to detect both known and unknown attacks by monitoring normal and abnormal behaviors of target processes. They claim to have lower chances of observing false positives compared to other approaches due to the fact that their approach is capable of detecting abnormal behavior deviation from normal one in unknown attacks as well. [30] also introduced a method to deal with false positives and false negatives in cloud environments, but it does not provide proof of concept for side-channel attacks and memory intensive programs together, which are hard to obtain in rigorous security demanding applications. Authors in [28] investigated detection of both known and unknown attacks using machine learning approaches, but they did not address the difficulty provoked by false negatives.

Researchers in [27] investigated a three-step detection method to deal with above problems in side-channels and branch prediction-based attacks. This approach helped to remove false positives in the system and detects branch prediction and DRAM attacks. However, the mechanism is not demonstrated to be efficient in the presence of realistic system load conditions, i.e., any background noise to make it more suitable to work in real systems and in cloud environments. Moreover, the technique has significant detection overheads in this system and needs to be more accurate about attack classes. In 2016, authors in [29] demonstrated an anomaly-based detection approach on *Heartbleed* vulnerability. This work explains the vulnerability of buffer over-reads only on malwares. This technique was able to detect data-oriented attacks at run-time in the user space. This approach was able to correctly detect 92% of two-class and 70% for one-class support vector machine model. However, there is no evidence of this detection mechanism working under normal system load conditions. Furthermore, it has some limitations for detecting unknown data-oriented attacks. It only tackles one type of vulnerability (Heartbleed) as a test case in OpenSSL cryptographic software library with one SVM algorithm. Hence, more sophisticated approaches can be proposed to detect a large class of side-channel attacks.

Authors in [26] propose to use hardware-supported lower-level micro-architectural features for detection of anomalies due to malware. This approach uses unsupervised learning on micro-architectural features taken from HPCs. Authors demonstrated that sequence of events in program execution, which motivates attacks, can be easily retrieved using HPCs. These features were later used to detect lower-level irregularities caused by malware. In another relevant research work presented in [24], authors proposed a technique called ConFirm, which is an inexpensive method to detect threatful alterations in the firmware of embedded control systems. ConFirm measures the low-level hardware events, which are performed all along the firmware execution. The detection accuracy and performance of ConFirm has been checked on different firmwares like ARM and PowerPC. As explained in Section 2.2.2, F+F attack is stealthy and to-date, it is considered to be undetectable using published detection mechanisms. A novel contribution of the NIGHTs-WATCH is that it is capable of successfully detecting F+F attack with an accuracy of 99.97% while relying on existing HPCs.

3. NIGHTs-WATCH

This section presents the proposed NIGHTs-WATCH detection mechanism. Intrusion detection is a problem of identifying data patterns that do not confirm with the expected (normal) system behavior. Detection mechanisms therefore apply a huge amount of effort in learning the expected system behavior first. NIGHTs-WATCH does this learning by profiling target crypto-systems (RSA & AES in our case) using carefully selected hardware events described in Section 3.2. Since detection mechanisms can only approximate the system behavior, they can be inaccurate and lead to false positives or false negatives at run-time. Moreover, they can also slowdown program execution due to detection overhead. We use all these parameters as evaluation metrics for NIGHTs-WATCH. We first describe our system model.

3.1 System Model

We demonstrate the effectiveness of proposed detection mechanism on Intel's core $i7-4770$ CPU running on Linux Ubuntu 16.04.1 with kernel architecture... Our threat model comprises of access-driven side-channel attacks that cause information leakage across complete cache hierarchy (i.e., L1 through LLC) in Intel's x86 architecture. We consider *same-core* as well as *cross-core* SCAs for detection. Moreover, we consider that the Operating System is not compromised. There are many high-level software libraries/APIs that help

extract information from HPCs such as; PerfMon, OProfile, Perf tool, Intel Vtune Analyzer, and PAPI. We use PAPI (Performance APIpplication Programming Interface) library [31] to access HPCs on Intel's Core *i*7 machines.

3.2 Selection of Hardware Events as Features

The PAPI library offers 100+ events for Intel's Core *i*7 systems. These events provide per-process, per-CPU, and system-wide statistical profiling of the running processes. Since NIGHTs-WATCH targets access-driven cache-based micro-architectural SCAs, we considered only those hardware events which are more likely to be affected by these attacks. In order to select the most relevant hardware events, we did experiments on a larger set of 12 most relevant events, presented in Table 1, in order to observe the impact of target computational loads,i.e., crypto-operations and attacks.

Table 1: Selected events related to cache-based SCAs

Scope of Event	Hardware Event as Feature	Feature ID
L1 Caches	Data Cache Misses	L1-DCM
	Instruction Cache Misses	L1-ICM
	Total Cache Misses	L1-TCM
L2 Caches	Instruction Cache Accesses	L2-ICA
	Instruction Cache Misses	L2-ICM
	Total Cache Accesses	L2-TCA
	Total Cache Misses	L2-TCM
L3-Caches	Instruction Cache Accesses	L3-ICA
	Total Cache Accesses	L3-TCA
	Total Cache Misses	L3-TCM
System-wide	Total CPU Cycles	TOT_CYC
	Branch Miss-Predictions	BR_MSP

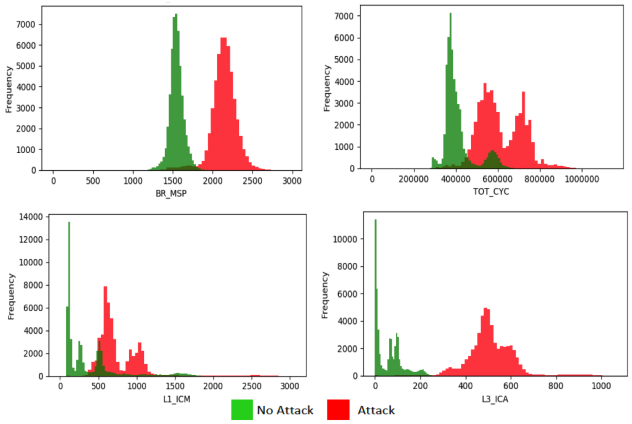

Figure 1: Experimental results of selected hardware events.

Figure 1 shows experimental results of selected hardware events that measure Branch Miss-Predictions (BR_MSP), Total execution Cycles (TOT_CYC), L1 Instruction Cache Misses (L1-ICM), and L3 Instruction Cache Accesses (L3-ICA) for 15,000 encryption rounds of RSA crypto-system. The figure shows frequency of samples on y-axis and magnitude of measured events on x-axis. Results shown in green represent normal behavior of RSA encryption rounds running under No Attack while results in red show RSA under F+R Attack. Figure 1 clearly shows that the magnitude of events significantly increases under attack conditions compared to normal behavior. Thus the events are affected by attack and reveal interesting information, which could be used by the ML for run-time detection.

3.3 Selected Machine Learning Models

NIGHTs-WATCH includes three different ML models: Linear Discriminant Analysis (LDA), Logistic Regression (LR) and Support Vector Machine (SVM) [32]. LDA is commonly used for dimensionality reduction of data and pattern classification in applications offering somewhat linear distribution. The LR uses logistic functions for carving linear relationships between dependent and non-dependent variables. LR is mostly used for classification of binary dependent variables. SVM finds a hyper-plane that separates the given data into different classes.

3.4 Methodology

Figure 2 presents an abstract view of the proposed methodology involving LDA, LR, and SVM models. NIGHTs-WATCH bases its detection on following 3 distinctive phases.

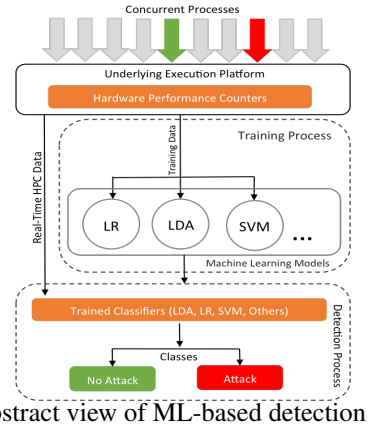

Figure 2: Abstract view of ML-based detection mechanism.

3.4.1 *Phase-I: Training of ML Models*

In the first phase, we train the machine learning models being used by NIGHTs-WATCH by profiling both RSA and AES crypto-systems, separately, under *Attack* and *No Attack* scenarios and variable load conditions. We collect significant samples of low-level data (roughly 1-Million) at a predetermined granularity by profiling the performance counters using PAPI library. We use appropriate hardware events (listed in Section 3.2) for profiling, which are responsible for differentiating between a victim/benign process and an attack process for both RSA and AES crypto-systems, separately. With the training data set of 1-million samples collected, we combine samples from all execution scenarios and train our ML models on these labeled data. We perform cross validation using $K-$fold cross validation technique for all models on the training data before applying them for the run-time profiling and detection.

3.4.2 *Phase-II: Run-time Profiling*

In the second phase, NIGHTs-WATCH performs run-time sample collection from hardware events. Here, sampling granularity plays an important role, as it exhibits a direct impact on the performance of the victim process and its shared library compared to normal execution. Moreover, it also effects detection speed and accuracy, which are crucial for real-time intrusion detection. NIGHTs-WATCH offers fine-grain

and coarse-grain profiling modes for run-time sample collection from the hardware events. These modes allow trade-off between performance impact and the detection speed. In fine-grain profiling mode, it collects samples at a much higher frequency to offer early detection at increased performance cost for the system, while in the coarse-grain profiling mode, it takes samples at relatively low frequency, which takes a little longer to detect intrusion, but incurs much less performance cost. In both cases, it can successfully detect SCAs much earlier than the attack completion.

3.4.3 Phase-III: Classification & Detection

In the last phase, NIGHTs-WATCH uses run-time data collected in previous phase for classification and subsequent detection of anomaly. The data from second phase are passed on to the ML models/classifiers in real-time at a prefixed frequency, which serve as trained anomaly detectors. Based on these run-time data, each model classifies samples into *Attack* and *No Attack* categories to report intrusion detection. Detection accuracy of each classifier varies depending on how well it is trained. Detection speed and performance impact, however, is subject to the sampling frequency selected by the NIGHTs-WATCH as discussed in Section 3.4.2.

4. EXPERIMENTS AND DISCUSSION

We create two experimental case studies to demonstrate detection of F+R and F+F attacks. In each case study, we evaluate the performance of our ML models under realistic system load conditions. To do so, we vary the system load from *No Load* (NL), *Average Load* (AL), to *Full Load* (FL) conditions by using selected SPEC benchmarks [33] that offer memory-intensive computations such as; *gobmk*, *mcf*, *omnetpp*, and *xalancbmk*, to run in the background as both attacks would be targeting/affecting caches and they would create realistic execution scenarios for evaluation. A NL condition involves only Victim and Attacker processes running, an AL condition involves at least two SPEC benchmarks running along with Victim & Attacker processes, and a FL condition involves at least four SPEC benchmarks running along with Victim & Attacker processes.

4.1 Case Study-I: Detecting Flush+Reload

Our first case study presents experimental results on the detection of F+R attack presented in Section 2.2.1.

4.1.1 Detection Accuracy

Detection accuracy is the foremost performance indicator in our evaluation metric for NIGHTs-WATCH ML Models. Results in Tables 2 through 4 show our experimental results for LDA, LR, and SVM models, respectively. Under No Load (NL) conditions, the accuracy of NIGHTs-WATCH is very consistently high across models, i.e., up to 99.51% for both LDA and LR models whereas up to 98.82% for SVM model. Average Load (AL) conditions seem not to bother the accuracy of LDA and LR models as it still remains the same, but for SVM model, accuracy reduces to 90%. Under Full Load (FL) conditions, all three ML models achieve an accuracy of 95%-above. Almost all existing state-of-the-art detection mechanisms measure detection accuracy under NL conditions while our results demonstrate a very high detection accuracy for F+R under variable and more realistic system load conditions. Results on high accuracy of ML models can be cross-validated by looking at the run-time behavior of features under different load conditions as shown in Figures 4 & 5. All features exhibit better-defined differences under NL condition, while only one feature (TOT_CYC) exhibits partially indiscernible behavior under FL condition. Such distribution of features leads the models to more accurate detection in general.

Table 2: Results using LDA model for F+R attack detection

System Condition	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
No Load	99.51	0.98	99.60	0.40	0.94
Average Load	99.50	0.98	98.42	1.58	
Full Load	99.44	0.98	87.76	12.24	

Table 3: Results using LR model for F+R attack detection

System Condition	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
No Load	99.51	0.98	100	0	1.63
Average Load	99.50	0.98	98.82	1.18	
Full Load	99.47	0.98	92.28	7.72	

Table 4: Results using SVM model for F+R attack detection

System Condition	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
No Load	98.82	0.98	33.72	66.28	1.29
Average Load	90.01	0.98	1.70	98.30	
Full Load	95.79	0.98	76.29	23.71	

4.1.2 Detection Speed

Intrusion detection speed is another very important performance indicator for run-time detection mechanisms. It is a trade-off between timely intrusion detection and detection overhead. F+R is a single encryption round attack. Therefore, in case of NIGHTs-WATCH, we define detection speed as a percentage of bits being encrypted *before* it can successfully detect an attack. For RSA performing 1024−bit encryption under F+R attack, the percentage of secret key bits being encrypted before the ML models can raise flag will reflect their detection speed. Authors of various attacks [3], [34] claim that, theoretically, it is enough to retrieve up to 50% secret key, while the rest can be attained through reverse-engineering. Therefore, a safe bound on detection speed can be *detection as late as* up to 50% of the secret key is being retrieved by F+R attack. Our experimental results reveal that all three ML models used by the NIGHTs-WATCH were able to detect F+R attack on 1024−bits RSA encryption in less than 2% of its completion, i.e., within the first 20−bits being encrypted by the victim under attack! Tables 2 through 4 refer to detection speed of individual models. This speed is achieved at the highest profiling granularity of NIGHTs-WATCH at which it samples hardware events after every 10−bits being encrypted. Each model cross-validates its detection results on at least two consecutive True Positives before raising the flag. We have tested our models with a coarse-grain sampling of every 50− and 100−bits being encrypted. Resulting accuracy remains the same while system overhead reduces further, which we discuss in Section 4.1.4.

4.1.3 Confusion Matrix

Confusion matrix gives a summary of prediction results on a classification problem by showing the number of correct

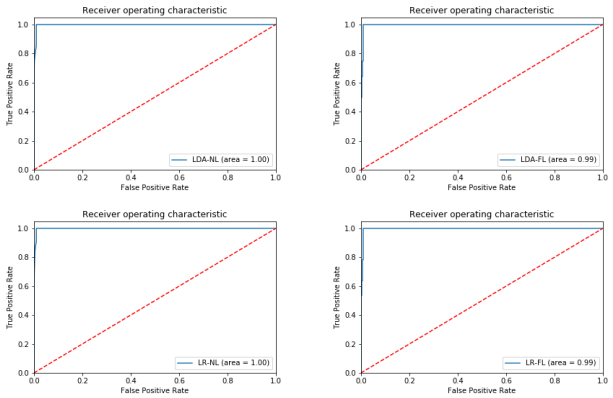

Figure 3: Receiver Operating Characteristic (ROC) Curves for LDA & LR Models while detecting F+R Attack.

and incorrect predictions broken down by each class. Confusion Matrix compliments the understanding of detection accuracy for our models. We analyze each ML model separately to elaborate its Confusion Matrix. Tables 2 shows, for all load conditions, the LDA model remains highly accurate (up to 99.51%). A closer look at the miss-classifications performed by model (0.49%) reveals that majority of them are False Positives (FP). Table 3 shows similar results for LR model with even lesser number of False Negatives (FN). For SVM model, Table 4 shows that, though the accuracy remains above 90%, the number of FPs and FNs vary significantly under different load conditions. No Load and Average Load conditions incur more FNs compared to Full Load condition. SVM model in this case did not prove as effective as LDA or LR models. Figures 3 show ROC curves for LDA & LR models illustrating their diagnostic ability for F+R attack under variable load conditions. ROC curves represent the ratio between the *attack* cases classified as *attack* (True Positive) to the *no attack* cases misclassified as *attack* (False Positive) for LDA & LR classification models. The values for Area Under the Curve (AUC) provide specific performance of each model in Figure 3. Results in ROC curves show high rate of True Positives, which reflect better accuracy and low rate of False Positives. These results are motivated by the fact that features under both NL & FL conditions offer significant separation for better classification as shown in Figures 4 & 5.

4.1.4 Performance Overhead

Performance overhead plays an important role in deployability of detection mechanisms in real systems. As mentioned in Section 4.1.2, detection granularity, which determines how aggressively the detector profiles hardware events and makes subsequent decisions, impacts performance overhead of detection mechanisms. Our results, presented in Tables 2-4, show only $1-2\%$ performance overhead of profiling and detection combined for LDA, LR, and SVM models. These results imply that the perceivable performance overhead of NIGHTs-WATCH using any of these models is very low. We measured these results at the highest sampling frequency for hardware events. With reduced sampling frequency, this overhead will be minimal.

4.2 Case Study-II: Detecting Flush+Flush

Our second case study presents experiments & results on the detection of F+F attack. Current state-of-the-art reports

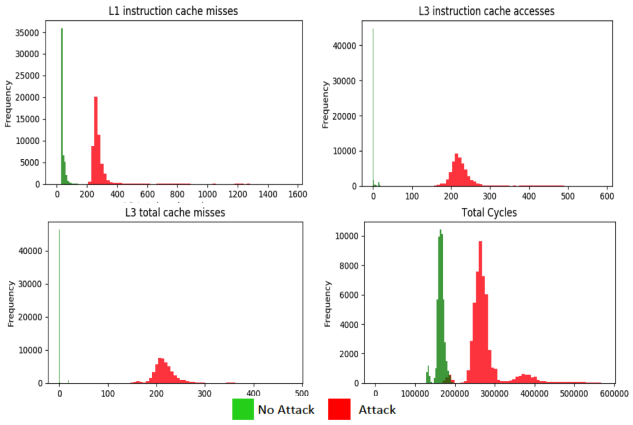

Figure 4: Selected ML Features under No load conditions for RSA encryption: With & Without F+R Attack.

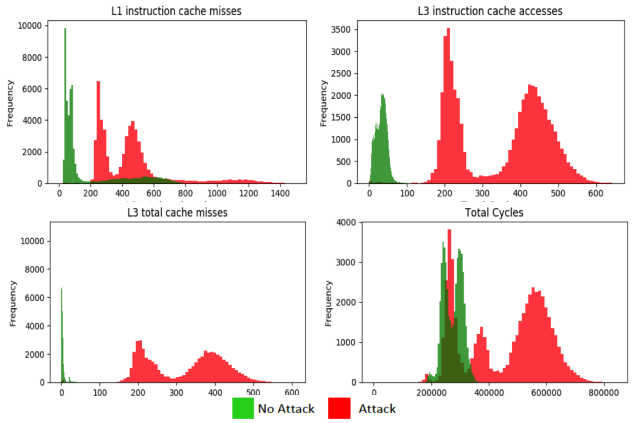

Figure 5: Selected ML Features under Full load conditions for RSA encryption: With & Without F+R Attack.

that any detection mechanism for F+F attack does not exist. It is a fast and high-resolution cache-based SCA that also targets LLC with the exception that it is stealthier than F+R attack. This stealthiness comes from the fact that F+F does not make any memory accesses, contrary to any other cache attack. Thus, *itself* as a running process, it causes no cache misses and the number of cache hits is reduced to a minimum due to the constant cache flushes. Authors in [35] claim that the spy process in this case cannot be detected based on cache hits and misses, or using state-of-the-art detection mechanisms. The F+F attack does cause, however, more cache misses and memory accesses for its victim process due to constant and high speed flushing. We build our argument around the the fact that detection mechanisms should indicate the *presence* or *absence* of intrusion from the victim's perspective. Thus, specifically identifying which particular process happens to be malicious is often not a requirement to apply protection. If, and when, an intrusion is detected on the victim process, the OS can take preventive measures such as completely isolated or critical section execution of victim process and/or relocation of co-located VMs etc. To the best of our knowledge, in this work, we are the first to practically demonstrate that F+F attack is successfully detectable using hardware performance counters and machine learning models. We achieve considerably high detection accuracy on F+F

attack. Next, we provide our results.

4.2.1 Detection Accuracy

Tables 5 through 7 show the detection accuracy achieved by all ML models under different load conditions. LDA model is able to show very high accuracy under all load conditions, i.e., 99.97%, 98.74% and 95.20% for NL, AL and FL conditions, respectively. The other two models perform well under NL, but suffer from greater inaccuracy under FL conditions. These results indicate that F+F, although a stealthier attack, can be detected accurately by NIGHTs-WATCH. In order to explain higher inaccuracy shown by LR and SVM models under FL conditions, we can observe the behavior of architectural features used by ML models as shown in Figure 6 and 7 for NL and FL conditions, respectively. Under NL condition, most of the features show distinguishable behavior while under attack and no-attack scenarios. However, in case of FL condition, it is evident that all the features start to overlap under attack and no-attack scenarios as shown in Figure 7. This behavior is in contrast to F+R attack where there were at-least few features that showed distinguishable behavior under FL as shown in Figure 5. This behavior of overlapping features makes it harder for ML models to properly discern attack scenario from no-attack scenario. However, it is very interesting to see that the LDA model is still able to show very good accuracy in case of FL conditions (95.20%).

4.2.2 Detection Speed

As indicated in the work of Gruss et al. [8], F+F attack on AES requires at least 350-400 encryption rounds to reliably guess the upper 4 bits of a secret key byte. Therefore, this value sets an upper bound on detection speed in the sense that a detection would be useful if it is done anytime before 400 encryption rounds are performed by AES while F+F attack is in progress. We provide the detection speed of ML models in terms of number of encryption rounds by which the attack is detected by the ML model taken as a percentage of 400 encryption rounds. For example, 25% detection speed means that the attack was detected by the first 100 encryption rounds. Tables 5 to 7 show worst-case detection speed for all ML models used by NIGHTs-WATCH. F+F attack is always detected by at most first 100 encryption rounds in all cases. In the best-case scenario, we achieve detection speed as high as within first 50 encryption rounds. The number of extra cache misses incurred during each AES encryption round while it is under F+F attack is not significant. Thus, high sampling frequency does not help. It can become difficult for ML models to discern attack and no-attack scenarios if hardware events are sampled per encryption round or within each encryption round (as values of hardware counters would not be very distinguishable). That's why hardware events are sampled at lower granularity (at least after each 50 encryption rounds) in case of F+F attack. This explains why detection speed for F+F is lower compared to F+R in terms of number of encryption rounds performed by the encryption algorithm.

Table 5: Results using LDA model for F+F attack detection

System Condition	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
No Load	99.97	25	75	25	1.18
Average Load	98.74	25	89.26	10.74	
Full Load	95.20	12.5	95.43	4.57	

4.2.3 Confusion Matrix

Table 6: Results using LR model for F+F attack detection

System Condition	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
No Load	91.73	12.5	0	100	1.103
Average Load	83.09	25	84.32	15.68	
Full Load	75.86	25	98.39	1.61	

Table 7: Results using SVM model for F+F attack detection

System Condition	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
No Load	97.42	12.5	0	100	0.79
Average Load	70.64	12.5	94.56	5.44	
Full Load	63.16	12.5	98.14	1.86	

Similar to F+R attack detection, our ML models exhibit miss-classification in this case as well. Tables 5 to 7 show breakdown of miss-classifications in terms of False Positives (FP) and False Negatives (FN). In most of the cases, the majority of miss classifications are False Positives. The only two exceptions are LR and SVM models under NL conditions. For both cases, all miss classifications are False Negatives shown in Tables 6 and 7. Figure 8 shows ROC curves for LDA and LR models illustrating their diagnostic ability for F+F attack. The ROC curves in this case show significant variations under Full Load conditions. These ROC curves validate our findings related detection speed and accuracy.

4.2.4 Performance Overhead

The profiling and detection overhead incurred by all three algorithms is very small and ranges from 1.18%, 1.103% and 0.79% for LDA, LR and SVM models, respectively, as shown in table 5 to 7. This overhead can be further reduced if hardware events are sampled at lower frequency. However, sampling at lower frequency can result into delayed detection.

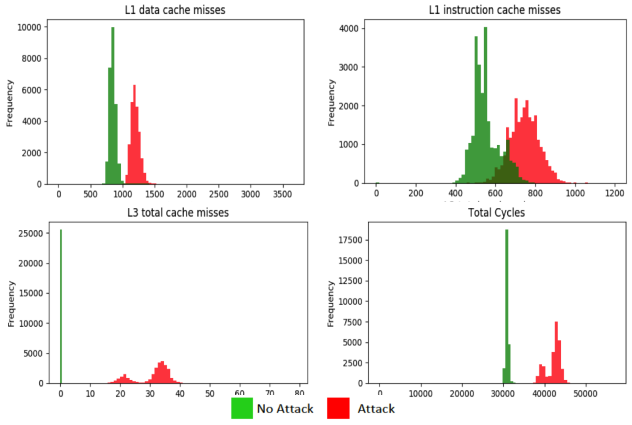

Figure 6: Selected hardware events under NL conditions for AES encryption: With & Without F+F Attack.

5. CONCLUSIONS AND FUTURE WORK

This paper presents a novel detection mechanism, called NIGHTs-WATCH, for access-driven cache-based side-channel attacks on modern processors. NIGHTs-WATCH uses multiple ML models for detection purpose. These models use real-time data from hardware performance counters to determine cache-based intrusions on RSA and AES crypto-systems. The paper presents experiments using two state-of-the-art access-driven cache-based SCAs to demonstrate the detection capability and effectiveness of NIGHTs-WATCH. We provide detailed analysis of results on detection accuracy, speed, system-wide performance overhead and confusion matrix for used models. One of the strengths of this work is that we provide experimental evaluation using realistic system load conditions. We use standard SPEC benchmarks to create No Load, Average Load, and Full Load conditions to train and

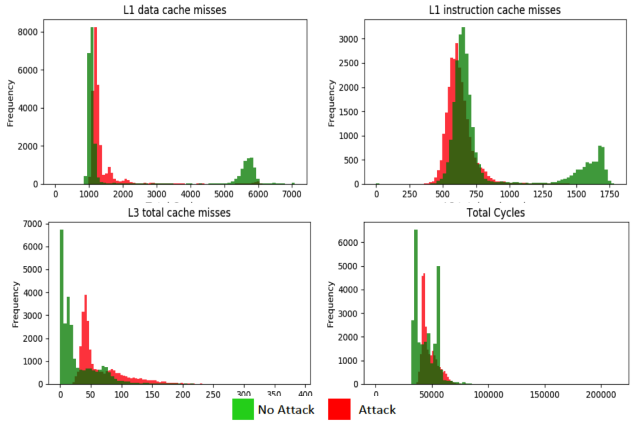

Figure 7: Selected hardware events under FL conditions for AES encryption: With & Without F+F Attack.

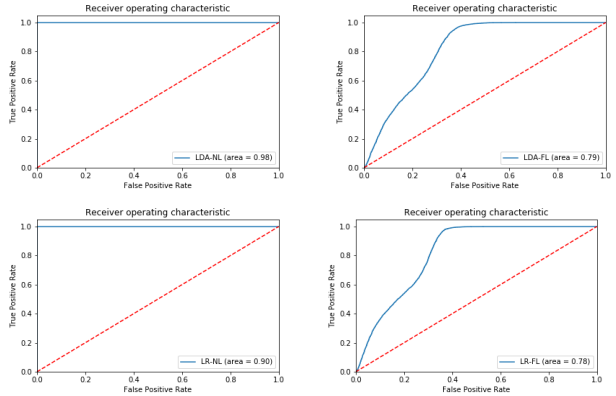

Figure 8: Receiver Operating Characteristic (ROC) Curves for LDA and LR Models while detecting Flush+Flush Attack.

evaluate ML models. NIGHTs-WATCH shows considerably high detection efficiency under variable system load conditions.

Our results show detection accuracy of 99.51%, 99.50% and 99.44% for F+R attack in case of NL, AL and FL conditions, respectively, with performance overhead of < 2% at the highest detection speed, i.e., within 1% completion of a single RSA encryption round. In case of Flush+Flush (stealthier) attack, our results show 99.97%, 98.74% and 95.20% detection accuracy for NL, AL and FL conditions, respectively, with performance overhead of < 2% at the highest detection speed, i.e., within 12.5% completion of 400 AES encryption rounds needed to complete the attack.

We also experimented with tree-based ML models like Random Forest (RF) in this work, which shows good detection accuracy for both attacks. Although, tree-based models achieve good accuracy, their implementation complexity makes it harder to use them in real-time attack detection mechanisms like NIGHTs-WATCH. Moreover, in order to prove portability, we performed experiments on different hardware such as Intel's core $i3-2120$ CPU running on Linux Ubuntu $4.4.0-116-generic$ at 3.30-MHz. Our results show consistency. Due to space constraints, however, we have omitted results on other ML models and hardware platforms. Timely detection of intrusion using actual variations in process's execution, measured directly from hardware (particularly caches), can help the operating system to take preventive measures such as halt or killing of identifiable malicious process, completely isolated or critical section-first execution of victim process, or relocation of co-located VMs etc. In future, we intend to integrate more ML models in NIGHTs-WATCH and apply it on other cache-based SCAs.

6. REFERENCES

[1] Y. Yarom and K. Falkner, "Flush+reload: A high resolution, low noise, l3 cache side-channel attack," in *USENIX Security 14*, pp. 719–732.

[2] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+flush: A fast and stealthy cache attack," in *DIMVA*, pp. 279–299, 2016.

[3] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: The case of aes," *CT-RSA*, pp. 1–20, 2006.

[4] P. K. et al, "Spectre attacks: Exploiting speculative execution," 2018.

[5] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown,"

[6] T. Tsunoo, Yukiyasuand Saito, T. Suzaki, M. Shigeri, and H. Miyauchi, "Cryptanalysis of des implemented on computers with cache," *CHES*, pp. 62–76, 2003.

[7] B. Gülmezoğlu, M. S. İnci, G. Irazoqui, T. Eisenbarth, and B. Sunar, "A faster and more realistic flush+reload attack on aes," COSADE.

[8] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+flush: A fast and stealthy cache attack," in *DIMVA*, pp. 279–299, 2016.

[9] Q. Ge, Y. Yarom, D. Cock, and G. Heiser, "A survey of microarchitectural timing attacks and countermeasures on contemporary hardware," *IACR Crypt. ePrint Arch.*, p. 613, 2016.

[10] S. A. et al, "Cross-vm cache-based side channel attacks and proposed prevention mechanisms: A survey," *Journal of Network and Computer Applications*, pp. 259 – 279, 2017.

[11] X. Jin, H. Chen, X. Wang, Z. Wang, X. Wen, Y. Luo, and X. Li, "A simple cache partitioning approach in a virtualized environment," in *IEEE ISPA*, pp. 519–524, Aug 2009.

[12] H. Raj, R. Nathuji, A. Singh, and P. England, "Resource management for isolation enhanced cloud services," in *CCSW*, pp. 77–84, 2009.

[13] F. Liu and R. B. Lee, "Random fill cache architecture," in *MICRO*.

[14] T. K. et al, "Stealthmem: System-level protection against cache-based side channel attacks in the cloud," in *USENIX Security 12*, pp. 11–11.

[15] O. Aciiçmez, "Yet another microarchitectural attack:: Exploiting i-cache," in *ACM CSAW*, pp. 11–18, 2007.

[16] M. A. et al, "Security best practices for developing windows azure applications," *Microsoft Corp*, p. 1, 2010.

[17] Y. Tan, J. Wei, and W. Guo, "The micro-architectural support countermeasures against the branch prediction analysis attack," in *IEEE TrustCom*, pp. 276–283, 2014.

[18] Y. Zhang and M. K. Reiter, "Düppel: Retrofitting commodity operating systems to mitigate cache side channels in the cloud," in *ACM CCS*, pp. 827–838, 2013.

[19] M. . Godfrey and M. Zulkernine, "Preventing cache-based side-channel attacks in a cloud environment," *IEEE Transactions on Cloud Computing*, vol. 2, pp. 395–408, Oct 2014.

[20] F. Liu, L. Ren, and H. Bai, "Mitigating cross-vm side channel attack on multiple tenants cloud platform.," *JCP*, pp. 1005–1013, 2014.

[21] Z. Wang and R. B. Lee, "New cache designs for thwarting software cache-based side channel attacks," in *ISCA*, pp. 494–505, 2007.

[22] Intel, "Intel 64 and ia-32 architectures developer's manual," 2013.

[23] E. W. L. Leng, M. Zwolinski, and B. Halak, "Hardware performance counters for system reliability monitoring," in *IEEE IVSW*, pp. 76–81.

[24] X. Wang and R. Karri, "Reusing hardware performance counters to detect and identify kernel control-flow modifying rootkits," *IEEE TCAD*, vol. 35, pp. 485–498, March 2016.

[25] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. J. Stolfo, "On the feasibility of online malware detection with performance counters," in *ISCA*, 2013.

[26] A. Tang, S. Sethumadhavan, and S. J. Stolfo, "Unsupervised anomaly based malware detection using hardware features," *CoRR*, 2014.

[27] M. A. et al, "Performance counters to rescue: A machine learning based safeguard against micro-architectural side-channel-attacks," Crypt. ePrint Arch., 2017. `https://eprint.iacr.org/2017/564`.

[28] M. Chiappetta, E. Savas, and C. Yilmaz, "Real time detection of cache-based side-channel attacks using hardware performance counters," *Appl. Soft Comput.*, vol. 49, pp. 1162–1174, Dec. 2016.

[29] G. Torres and C. Liu, "Can data-only exploits be detected at runtime using hardware events?: A case study of the heartbleed vulnerability," in *HASP*, pp. 2:1–2:7, 2016.

[30] T. Zhang, Y. Zhang, and R. B. Lee, "Cloudradar: A real-time side-channel attack detection system in clouds," in *RAID 2016*.

[31] "Performance application programming interface." `http://icl.cs.utk.edu/papi/`, 2018.

[32] C. M. Bishop, *Pattern Recognition and Machine Learning*

(Information Science and Statistics). 2006.

[33] "https://www.spec.org/benchmarks.html," 2018.

[34] Y. Yarom, D. Genkin, and N. Heninger, "Cachebleed: a timing attack on openssl constant-time rsa," *Journal of Crypt. Engg. 2017*.

[35] D. Gruss, C. Maurice, A. Fogh, M. Lipp, and S. Mangard, "Prefetch side-channel attacks: Bypassing smap and kernel aslr," in *ACM CCS*.