2018 UKSim-AMSS 20th International Conference on Modelling & Simulation

ConfMVM: A Hardware-Assisted Model to Confine Malicious VMs

Zirak Allaf
zirak.allaf@port.ac.uk

Mo Adda
mo.adda@port.ac.uk

Alexander Gegov
alexander.gegov@port.ac.uk

School of Computing
University of Portsmouth
Portsmouth, UK.

***Abstract* — Vulnerabilities in both hardware and software have exposed them to the lack of managing programs securely in the computational environment, giving hackers the means to conduct side channel attacks with intention to steal sensitive information, including secret encryption keys. Current techniques enable attackers to exploit vulnerabilities at the micro-architecture level to build side channels. A typical example is the use of the Flush+Reload technique in the Meltdown attack [1]. This paper proposes the detection of malicious loop activities within the Flush+Reload programs through the introduction of a new classification technique. Most current detection models, approach the side channel attacks, by relying on the correlation between attacker and victim programs through the use of machine learning algorithms. This paper differs from such models. It solely analyse the malicious loop activities inside the Flush+Reload attack program and does not seek to synchronise victim and attacker programs. The model proposed has the ability to classify Flush+Reload attacks with a level of accuracy approaching 99% for native and 96% for cloud systems without increasing the cost of detection in a cloud systems above that in native systems.**

Keywords* — *Side-Channel, Flush+Reload, Prime+Probe, HPC, Machine Learning, Cloud Computing, Cryptography

I. INTRODUCTION

The use of the Internet and, in particular, cloud computing on a wide range of mobile and desktop devices is growing, with privacy outsourced to the providers of cloud service. The result exposes data to a variety of threats, making maintenance and security of data for attacks increasingly challenging. Cryptography is the most important element in maintaining the security of data in every state to which it passes while being transmitted, processed and stored. Hackers are attracted by the data's vulnerability and seek access to the data by using side channel attacks to steal cryptographic components including the secret keys. A range of sophisticated side channel attack techniques have been put forward. Should these techniques succeed, data protection would be unreliable to the point where end users would be likely to either limit the extent to which they used cloud services or abandon cloud services altogether. Recent research has shown side channel attacks to be in widespread use, allowing attackers to obtain every part of a cryptographic key in around one minute on native systems and three minutes in a cloud system [2], [3]. A good deal of publicity has been given to a number of hacks where large amounts of data were stolen by hackers. Cloud service providers claim encrypting the data will toughen the tasks of who have unauthorised access. These tasks will involve many years of brute force attack to identify the encryption key. That defence is invalid when the key can be stolen in less than three minutes, from within the system itself.

The objective of this research is the creation of a knowledge based detection system capable of leveraging hardware support to analyse and confine the activities in the process with the view to detect malicious processes running in user space. It would be necessary for such a system to have the potential to mitigate and eliminate threats against the security of the cryptographic algorithms. The system uses machine learning techniques to detect a Flush+Reload side channel attack occurring in user space without relying on the victim and the attacker programs synchronisation.

Section 2 of this paper discusses related work both on side channel attacks and on countermeasures against them. Section 3 illustrates the study's methodology, while section 4 discusses the results. A conclusion then ends the paper.

II. RELATED WORK

A number of studies have shown it is possible to carry out side channel attacks on CPU components and sensitive applications through the use of user credentials and compromising technologies.

A. Side Channel Attacks

The notion of leaking data was first discussed by Lampson [4] who illustrated the possibility that the weakness of operating systems in protecting memory contents could lead to leakage through hidden communication channels. The malicious process would encode hardware in order to uncover information about the target processes. Lampson [5] proposed a protection mechanism which relies on program confinements from leaking data between two processes. Kocher et al. [6] applied side channel attack to modern computer systems and processes and demonstrated how sensitive data including secret keys could be obtained from a variety of cryptographic methods. The exploitation of hardware contention begins with CPU components such as CPU Cache Memory. Studies have shown the CPU to be the resource most targeted by attackers.

The CPU cache is the main source of information for side channel attacks. In earlier days, L1 and L2 caches were targeted by attacks [7], [8], [9] deploying as communication channels between attack and victim processes residing on the same core. Core migration challenged this method through alternation of process assignment [9]. Faster, higher bit-rate attacks than L1 were explored. Attentions were also given to unified cache L3 across processor cores because of its higher resolution with shorter times for recovering sensitive information [10]. Rowhammer [11] attacks have exploited main memory DRAM as a channel for data leakage with a much higher bandwidth.

Main Memory has been exploited as a bridge across processes to transfer large amount of data. After practising side channel attack on the CPU cache, the researchers have found the feasibility of transferring large amount of data through DRAM. Gruss et al. [12] used the Flush+Reload technique through exploitation of prefetching address translation [13] to map virtual into physical addresses. The paper [14] proposed a high-speed covert channel of up to 2Mb/s using a side channel attack across the CPU, without memory sharing.

B. Mitigation and Solutions Based Hardware and Software

The developments in side channel attacks have been met by similar developments. Countermeasures have been put forward as a way of counteracting and mitigating side channel attacks' negative impact in real systems.

Earlier research has demonstrated the achievement of high resolution [10] and very fast [3] side channel attacks through Flush+Reload, which has the potential to exploit the system's page sharing characteristics. This is especially acute cross core in cloud systems [9], through LLC. These demonstrations of potential vulnerability presaged software developers and cloud providers to disable page sharing features which were previously their systems' default settings. The work presented in [15] suggested the kernel space solution CACHEBAR to give concrete protection to pages shared between cross VMs in PaaS. The drawback of this approach is performance impairment, and this is particularly true in cloud systems. In the drive to free space that can be leased to more tenants, cloud providers aim to reclaim the maximum possible amount of memory, and this creates temporal localities (data and/or resources being reused in a small period of time) that impacts on the system negatively. According to the work in [3], memory page vulnerabilities when using S$A against LLC cache in large page settings might enable AES secret keys to be extracted.

Cleemput et al. [16] suggested that compilers could be used against side channel attacks if execution time were made uniform by transforming code in the in AES algorithm. This study, however, suffers from hardware requirements, code complexity, portability, and performance issues. Crane et al. [17] suggested injecting noise into program execution to achieve control-flow diversity. This study gave solutions to limitations discussed in [16]. However, their solutions are specific to the application concerned, cannot be generalised and degrade the system performance. Countermeasures proposed by other researchers include proposal for a Sanctum protection model that flushes the L1 cache while the host OS performs context switching [18]. The paper in [19] suggested that a Prime+Probe attack could be defeated by flushing L1D/I to avoid data dependency.

On the other hand, a number of Profiling-Based approaches have been used for side channel attack detection. In [20] a statistical analysis to identify cache attacks, observing CPU cycles to monitor accessed and non-accessed cache (miss/hit) attacks was suggested. Briongos et al. [21] proposed monitoring for Flush+Reload attacks against the AES algorithm, and the study looked at the clflush instruction of multiple cache lines in the core of the attack. Detecting attacks in that case were mainly driven by the collection of CPU cycles. Their findings are is not efficient in for real-time systems, as unexpected workload may trigger false positives. Machine learning has been proposed as a way to reduce the number of false positives and thereby increasing the detection efficiency. However this require the involvement of more than one feature (e.g. CPU cycle).

Hardware-based detection techniques makes use of PMU to obtain a greater level of detail so that features can be extracted that support higher resolution detections. Zhang et al. [20] suggested CloudRadar, which seeks to detect signatures and anomalies. Besides, side channel attacks, this cloudRadar seeks to detect other forms of attacks such as denial of service against CPU caches. HPCs also make it possible to use machine learning algorithms for the extraction of patterns that have not yet been heavily explored. The results are efficient, highly accurate and reliable. Payer [22] suggested HexPads as a way of detecting side channel attacks by exploiting PMU through the use of sensitive events. Thresholds were set for every processor in the system by means of the perf tool, which makes use of the system proc file to obtain information about every running process. The work in [23], demonstrated that the attackers may hide the PID of the attack, thus evading detection when HexPads misses the attacker's PID and fails to monitor their activities. Instead of monitoring PID, the proposed system monitors activities in the processor core and this would prevent an attacker from avoiding being profiled. An additional limitation of HexPads is its ability to detect VMs. Our proposed technique in this paper detects both malicious VMs and native processes at no extra difference in cost.

Allaf et al. [24] noted that side channel attacks rely on cache behaviour based on the attacker/target correlation. Alam et al. [25] suggested using machine learning to extract the patent of the attack using synchronisation to correlate attack and target, but no correlation is required by the proposed technique since it allows the system to detect malicious loop activities occurring inside the Flush+Reload program. The computational cost in the host system is thereby saved.

III. METHODOLOGY

This work will present the use of a supervised method in classifying the Flush+Reload side channel attack and will then compare native system and cloud system results to show the level of accuracy attained by the classifier in efficient detection of side channel attacks for both native and VM processes.

A. K-Nearest Neighbour (KNN)

The instance-based algorithm *k*-NN is a simple nonparametric classification algorithm that has been around for a long time [26]. It may be used in any classification task using discrete data but the classification of unseen data-sets and regression tasks to predict a continuous label relying on datasets based on time series. Each tested data class is predicted by measuring test data items' similarity. The classification process conducted on the test data-set realised for each class on *k* closest neighbours. Any set of sample data points can be classified according to its neighbours' majority vote. *k*NN makes use of a search engine based on measurement distance functions to find from the dataset the closest data items. *k*-NN has been studied for a considerable period of time and a number of distance measures have been used, with the most popular being: Euclidean; Manhattan; Hamming; and Minkowski. This study has made use of the Manhattan measure to find the best *k* instance for the classification tasks in the training data-set.

$$D_H = \sum_{i=1}^{k} |x - y| \tag{1}$$

Optimal *k* values are found on the basis: larger values mean better classification. Since this approach is not reliable, this study uses the cross-validation (CV) [27] to determine how optimised the *k* value is. Cross-validation divides datasets into a number of predefined data-sets before feeding them independently to *k*-NN during training and testing tasks. The optimal *k* value is selected by the search engine from a number of independent predefined data-sets.

The *k*-NN algorithm measures distances between data items in the data-set. This is why *k*-NN has been chosen; the choice of data set rests on data sample similarities with stress on features that are near neighbours. The features chosen for this experiment include L1, L2 and LLC cache misses, because Flush+Reload attacks work by flushing a specific memory address from all levels of cache – that is, L1, L2 and LLC – and, after a very short sleep, accessing the memory address that was flushed. Three consecutive cache access points are needed; the memory access instruction generates an identical number of hardware events, which in this case are cache misses, for each cache level. *k*-NN is looking for data items with the smallest distance between them, and so identifies efficiently the malicious loop inside the Flush+Reload program.

B. Method Evaluation

When the classifier has predicted unseen data sets, its accuracy is evaluated to test how sensitive and precise it is, and how specific it is in identifying malicious activity. Those three characteristics – sensitivity, specificity and precision – are plotted along a Receiver Operating Characteristic (ROC) curve. The ROC was put forward by [28], [29] to enable the performance metrics of a predictive model to be examined in visual form by drawing line graphs connecting sensitivity and specificity. A point on the curve will signify a ratio between 0 and 1. Since the halfway point on this curve represents a random guess, the diagonal connects the points (0.5,0.5). Anything above that diagonal will be more accurate than a random guess, and the actual position enables its accuracy to be characterised on a continuum from good to excellent, with the very best performance closest to the top right corner. Anything below the diagonal is likely to be even less accurate than a random guess.

C. Performance Monitoring Unit (PMU):

Programmers need help in debugging and locating the bottlenecks in their programs and PMU provides this assistance with run-time feedback. There is one PMU in each processor core of current multicore processors, and each has the task of capturing core activities. Each activity is recorded as an event and it happens when use of a component of the CPU calls for a hardware action. Among these events are cache accesses to L1, L2 and L3, and branch predictions. There are is detailed Intel documentations on events and how they can be used in chapter 18 and 19 [30]. For this paper, PMU has been used deployed to profile attacker activities in the host operating system using the kernel module.

D. Benchmarks

Benchmark suites of programs given by communities and companies with agreements to be representatives and assess the relative performance of a system. They measure performance of a piece of code, an application or a system. The Standard Performance Evaluation Corporation (SPEC) CPU2006 Benchmark suite comprises a number of programs each representing a specific program type. For example, bzip2 represents compression programs. SPEC can be used as a workload to detect and measure system performance degradation. SPEC [31] is the benchmark used most widely for system performance measurement. In this study, we utilise SPEC CPU2006 as a workload on the host system in regard to side channel attack with Flush+Reload in order to stress the computational environment and to evaluate the precision of the detection model.

E. Threat Model and Assumptions

The CPU is a multi-core. Each core has private caches L1 and L2 together with a single Last Level Cache (LLC) that is shared with other cores. This study examines side channel attacks that mostly exploit LLC occurring both in native and cloud systems. The assumptions are in native systems the attackers are users and in cloud systems, the attackers are most likely the Virtual Machines (VM). This is particularly the case in the Infrastructure as a Service (IaaS), where guest VMs operate in a virtualised environment that enables the sharing of resources such as CPU caches.

According to [2], [10], the RDTSC instruction is used to attack the timing channel, since it can be accessed through user space and does not require any privileges. This enables the attacker then observes the target program's use of shared resources. For side-channel attacks, this study focuses on Flush+Reload, a memory cache side channel popular in today's computer systems.

F. Experiments Setup

This section describes the software and hardware components for the experiments involving online and offline observations with a view to detect attacks.

1) Hardware Specifications: The experiment was conducted on HP Proliant DL360 G7 with Intel Xeon X5650 2.66GHz processor with 16 GB RAM running Ubuntu 14.04. The various tests used SPEC cpu2006.

2) Data Collection: Data collection for this experiment is by HPCs (Hardware Performance Counters) that profile the execution attributes of programs for each processor core. Events are sampled that would occur in a Flush+Reload attack to see whether such an attack is in fact in process. Using the available physical PMC counters is best; there are seven of them in this experiment, and so the PMC has monitored L1, L2 and LLC cache misses in three fixed function counters and four programmable counters. Profiling automatically counts the number of the selected events for all programs assigned to the select processor core. Each sample lasts 0.02 µs, the time needed for a single iteration of the malicious loop in a Flush+Reload attack. The kernel module takes from the profiling loop 4,000 samples to see whether the attacker program could have been running more than once.

3) Data Aggregation: Aggregating mean values allows the data-set to be fed into machine learning algorithms. Aggregation slices the raw data-set into a series of subsets, each containing n samples, to generate a new time series execution data-set. n samples are combined using the mean function, after which their average is taken to give a sample representing the malicious process. Malicious activities appear in the raw data as consecutive samples in the form of time series, but grouped samples cannot be handled in one step by k-NN, and so the aggregation mean function makes possible the feeding of the data-set into the k-NN algorithm so that k-NN can perform the classification.

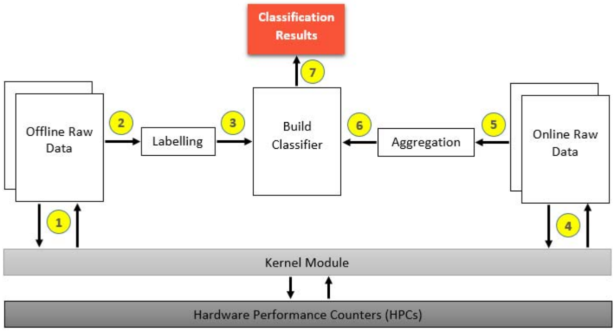

Fig. 1. Detection system overview

G. Detection System Overview

Figure 3 illustrates the detection systems, of which the general concept is described in this section. Step 1, the detection system communicates with the kernel module for the collection of data from the PMC counters; the PMC counters can only be written to with the kernel's permission. The kernel module profiles processor cores as raw data and provide both off-line and online data. Offline data is labelled, in step 2, to reflect attack activities and prepare training and test data-sets. It is then fed them to the k-NN algorithm to build a classifier, in step 3, using a cross validation method. The online data, on the other hand, is collected in step 4. Then, in step 5, the data must be pre-processed to produce a new data-set from the consecutive attack activities using the aggregation mean function. In step 6, the new data-set is then fed to the classifier to predict attack activities in the system in step 7.

IV. RESULT ANALYSIS AND DISCUSSION

This section addresses the success of *k*-NN classifier in detecting side channel attacks in native and in cloud systems. It goes on to discuss how the selected features of the *k*-NN algorithm's search engine finds in the dataset indicators the presence of side channel attacks, if any.

A. Analysis

Figure 2 and 3 illustrate the use of Receiver Operating Characteristics (ROC) metrics to quantify the classifier's accuracy in discriminating between two process activities that are benign and are normal workloads, and malicious loop activities. The classifier's output is represented as ROC curves, which represent the sensitivity and specificity calculations at incremental thresholds between zero and one across 10 folds when the same dataset is randomly shuffled, resulting in each fold having a different spread of the data. The *Y* axis plots the classifier output's True Positives Rates and the *X* axis plots False Positive Rates. Each fold is an individual ROC and is the light blue line. It represents detection quality. The solid blue line is the calculated mean of 10 ROC curves. The classifier's stability is shown by the transparent blue area representing confidence intervals.

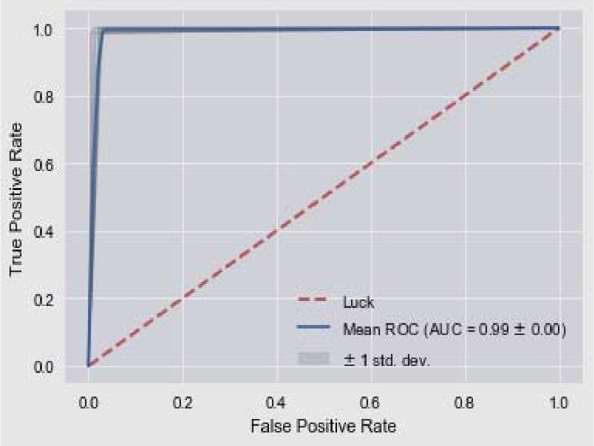

Fig. 2. The distribution of ROC curves in native system.

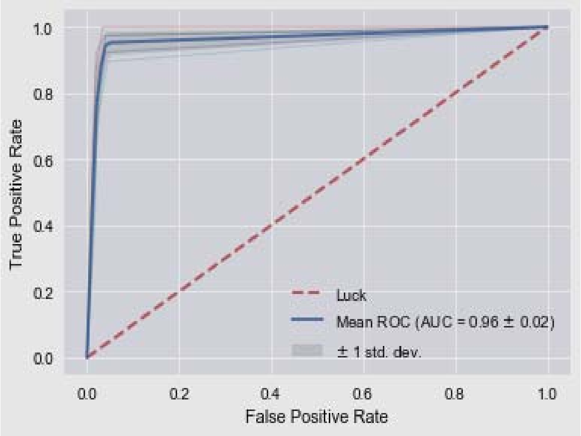

Fig. 3. The distribution of ROC curves in cloud system.

Three highly relevant features have been chosen to illustrate the leverage of *k*-NN in detecting malicious loop activities inside a Flush+Reload attack. They were selected from hundreds of events supported by the CPU because the clflush instruction followed by a mov instruction to a memory address generates the same number of cache miss events as the host OS. Using machine learning detects a malicious activity in a shorter time than an attacker needs to establish covert communication channels.

Figure 2 shows the ROC metric that evaluates the classifier's ability to detect malicious loop activities among normal workloads in the host system. Success in observing program execution attributes and classifying processes as malicious or benign as a measure of the risk of existing side channel attack in the system is shown as estimated by AUC of ROC. The model identifies the malicious loop in a native system with very high accuracy (AUC=0.99 the average of 10 folds, with zero confident interval). In the cloud, however, the same algorithm trained on a data-set that captured VM activities was less accurate at predicting malicious activities from among other workloads (AUC=0.96 Confident interval=0.02). The classifier is therefore 3% less efficient at identifying malicious loop activities in the cloud than in a native system. This is the result of the noise in L1 and L2 cache memories arising from the additional translation layer imposed by Structure as a Service (SaaS), a layer which for security reasons the hypervisor hides across VMs. *K*-NN's reliability arises because the execution time series is sliced into proper window-size segments, confining the malicious loop's time-sliced execution inside a Flush+Reload attack.

B. Discussion

The results demonstrate the ability of the k-NN algorithm to build a classifier that is very accurate in identifying malicious loop activities used by the Flush+Reload attack in both native and cloud systems. Flush+Reload frequently repeats the same task, which is organised by executing clflush instruction to a specific memory address of interest and then executing mov to the same address. When it receives the memory address from which to read the contents, mov must retrieved them from memory pages because the previous clflush rendered the content in hierarchical cache memory at that address invalid, and Invalid contents are updated from main memory leading to a sequence of hardware events. Cache misses at L1, L2 and LLC are the events selected as executing two consecutive instructions produces an equal number of L1, L2 and LLC cache misses. This sets the attack program apart from other workloads as shown in SPEC benchmark suite including two integer applications (bzip2 and gcc) and two floating applications (bwaves and dealII). It is this particularity that enables the k-NN algorithm to build a model identifying the malicious loop in the computational environment with high accuracy. The aggregation mean function corroborates the classifier's reliability by slicing the data-set into a sequence of windows of equal size to be searched for samples of the malicious loop. The malicious loop is then seen to be repeating the same task of flushing and accessing the same memory address.

K-NN is a distance-based algorithm using the search engine to perform classification by finding the closest samples in the data-set. These three features (L1, L2 and LLC) have the shortest distance which, in native systems, is zero. In cloud systems, on the other hand, the noise in L1 and L2 caches slightly reduces the classifier's accuracy.

Another advantage found in these results is that the profiling can record native and cloud-based activities with no difference in cost. The same classifier does not require training with different data-sets, and the same data-set can be used to train the classifier to detect malicious processes either in originate in a native or a VM process – but there will be a 3% degradation in the classifier's accuracy in cloud settings due to the noise.

V. CONCLUSIONS

This paper has proposed detection of side channel attacks through classification by using Hardware Performance Counter (HPCs) to record hardware events in the host system. What is proposed to be monitored is events relevant to a Flush+Reload attack, and these improve the accuracy of detection significantly. The paper also put forward a new profiling technique to record activities in processor cores to which processes are assigned. The proposed detection model is not only reliant on the relationship between target and attacker programs, but it can also detect a malicious VM performing side channel attacks with no additional cost compare to the detection of a native process. The ROC curve is used to evaluate the efficiency of the proposed classifier for the detection of side channel attacks. The classifier detects side channel attacks in both native and cloud settings with accuracy of up to 99% and 96% respectively under SPEC CPU2006 workloads.

However, the proposed method cannot detect techniques such as Prime+Probe due to the behaviour of the malicious loop inside the program. Also if the function is smaller than the number of malicious loop samples, the new data-set will be noisier and influence the classification model, causing the attacker to be wrongly classified.

The future work will consider the accuracy improvement of the classifier in malicious process detection for cloud systems. Further work will be devoted to the design of a model that can detect other side channel attacks and improves the aggregation to avoid the possibility of missing some forms of malicious loop activities.

REFERENCES

[1] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, “Meltdown,” arXiv preprint arXiv:1801.01207, 2018.

[2] G. Irazoqui, M. S. Inci, T. Eisenbarth, and B. Sunar, “Wait a minute! a fast, cross-vm attack on aes,” in International Workshop on Recent Advances in Intrusion Detection. Springer, 2014, pp. 299–319.

[3] G. Irazoqui, T. Eisenbarth, and B. Sunar, “S $ a: A shared cache attack that works across cores and defies vm sandboxing–and its application to aes,” in 2015 IEEE Symposium on Security and Privacy. IEEE, 2015, pp. 591–604.

[4] B. W. Lampson, “Dynamic protection structures,” in Proceedings of the November 18-20, 1969, fall joint computer conference. ACM, 1969, pp. 27–38.

[5] ——, “A note on the confinement problem,” Communications of the ACM, vol. 16, no. 10, pp. 613–615, 1973.

[6] P. C. Kocher, “Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems,” in Annual International Cryptology Conference. Springer, 1996, pp. 104–113.

[7] C. Percival, “Cache missing for fun and profit,” 2005.

[8] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, “Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds,” in *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009, pp. 199–212.

[9] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, “Cross-vm side channels and their use to extract private keys,” in Proceedings of the 2012 ACM conference on Computer and communications security. ACM, 2012, pp. 305–316.

[10] Y. Yarom and K. Falkner, “Flush+ reload: a high resolution, low noise, l3 cache side-channel attack,” in 23rd USENIX Security Symposium (USENIX Security 14), 2014, pp. 719–732.

[11] D. Gruss, C. Maurice, and S. Mangard, “Rowhammer. js: A remote software-induced fault attack in javascript,” in Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, 2016, pp. 300– 321.

[12] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O’Connell, W. Schoechl, and Y. Yarom, “Another flip in the wall of rowhammer defenses,” arXiv preprint arXiv:1710.00551, 2017.

[13] D. Gruss, C. Maurice, A. Fogh, M. Lipp, and S. Mangard, “Prefetch side-channel attacks: Bypassing smap and kernel aslr,” in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2016, pp. 368–379.

[14] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, “Drama: Exploiting dram addressing for cross-cpu attacks.” in USENIX Security Symposium, 2016, pp. 565–581.

[15] Z. Zhou, M. K. Reiter, and Y. Zhang, “A software approach to defeating side channels in last-level caches,” arXiv preprint arXiv:1603.05615, 2016.

[16] J. V. Cleemput, B. Coppens, and B. De Sutter, “Compiler mitigations for time attacks on modern x86 processors,” ACM Transactions on Architecture and Code Optimization (TACO), vol. 8, no. 4, p. 23, 2012.

[17] S. Crane, A. Homescu, S. Brunthaler, P. Larsen, and M. Franz, “Thwarting cache side-channel attacks through dynamic software diversity.” in NDSS, 2015, pp. 8–11.

[18] V. Costan, I. A. Lebedev, and S. Devadas, “Sanctum: Minimal hardware extensions for strong software isolation.” in USENIX Security Symposium, 2016, pp. 857–874.

[19] Y. Zhang and M. K. Reiter, “Duppel: retrofitting commodity operating¨ systems to mitigate cache side channels in the cloud,” in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 827–838.

[20] T. Zhang, Y. Zhang, and R. B. Lee, “Cloudradar: A real-time sidechannel attack detection system in clouds,” in International Symposium on Research in Attacks, Intrusions, and Defenses. Springer, 2016, pp. 118–140.

[21] S. Briongos, P. Malagon, J. L. Risco-Mart´ ´ın, and J. M. Moya, “Modeling side-channel cache attacks on aes,” in Proceedings of the Summer Computer Simulation Conference. Society for Computer Simulation International, 2016, p. 37.

[22] M. Payer, “Hexpads: a platform to detect “stealth” attacks,” in International Symposium on Engineering Secure Software and Systems. Springer, 2016, pp. 138–154.

[23] X. Wang and R. Karri, “Numchecker: Detecting kernel control-flow modifying rootkits by using hardware performance counters,” in Proceedings of the 50th Annual Design Automation Conference. ACM, 2013, p. 79.

[24] Z. Allaf, M. Adda, and A. Gegov, “A comparison study on flush+ reload and prime+ probe attacks on aes using machine learning approaches,” in UK Workshop on Computational Intelligence. Springer, 2017, pp. 203–213.

[25] M. Alam, S. Bhattacharya, D. Mukhopadhyay, and S. Bhattacharya, “Performance counters to rescue: A machine learning based safeguard against micro-architectural side-channel-attacks,” 2017, https://eprint.iacr.org/2017/564.

[26] T. Cover and P. Hart, “Nearest neighbor pattern classification,” IEEE transactions on information theory, vol. 13, no. 1, pp. 21–27, 1967.

[27] S. Arlot, A. Celisse et al., “A survey of cross-validation procedures for model selection,” Statistics surveys, vol. 4, pp. 40–79, 2010.

[28] A. P. Bradley, “The use of the area under the roc curve in the evaluation of machine learning algorithms,” Pattern recognition, vol. 30, no. 7, pp. 1145–1159, 1997.

[29] T. Fawcett, “An introduction to roc analysis,” Pattern recognition letters, vol. 27, no. 8, pp. 861–874, 2006.

[30] I. Intel, “Software guard extensions programming reference, revision 2,” 2014.

[31] J. L. Henning, “Spec cpu2006 benchmark descriptions,” SIGARCH Comput. Archit. News, vol. 34, no. 4, pp. 1–17, Sep. 2006. [Online]. Available: http://doi.acm.org/10.1145/1186736.1186737