Malicious Loop Detection Using Support Vector Machine

Zirak Allaf
School of Computing
University of Portsmouth
Portsmouth, UK
Email: zirak.allaf@port.ac.uk

Mo Adda
School of Computing
University of Portsmouth
Portsmouth, UK
Email: mo.adda@port.ac.uk

Alexander Gegov
School of Computing
University of Portsmouth
Portsmouth, UK
Email:alexander.gegov@port.ac.uk

Abstract—Existing Side-channel attack techniques, such as meltdown attacks, show that attackers can exploit the microarchitecture and OS vulnerabilities to achieve their goals. In this paper, we present the development of our real-time system for detecting side-channel attacks. Unlike previous works, our proposed detection system does not rely on synchronisation between the attackers and victims. Instead, it uses processors' performance indicators to capture malicious Flush+Reload activities with an accuracy of up to 99%. Moreover, the detection activities can be achieved with minimum time delay in both native and cloud systems with a low overhead performance approximately less than 1% in the host system.

1. Introduction

In Cloud computing environments, data protection becomes a great challenging factor. Although crypto-algorithms have emerged as promising approaches to protect data, hardware and software vulnerabilities attract hackers to target cryptographic components and steal their private keys through side channel attacks. The side channel attack is the action of stealing sensitive information by exploiting vulnerabilities in hardware/software. The main attack characteristics can be identified firstly by relying on hardware contentions - cache misses to observe victim' activities, secondly by performing operations on the system without privileges, and finally by monitoring the processors' cycles [1] and the power consumption [2] as metrics to measure latency.

Flush+Reload [3], Prime+Prob [4] are the two common attack techniques in performing side-channel attacks.Flush+Reload technique observes the victim by monitoring specific cache line(s) in cache memory by measuring the access time, if the cache memory buffers data from main memory in the specified cache line(s), the access time is faster, which indicates that the victim has recently accessed the data. Whereas Prime+Prob monitors the victim's activities by filling the cache memory and waiting for the victim to evict the attacker's data, in this case, the attacker can easily determine the evicted cache line, which is replaced by the victim's data. In this paper, we focus on Flush+Reloadattacks, which relies on hardware contention vulnerabilities. Hardware contentions occurred when multiple threads operating on the same data in the cache memory and thus the attacker utilised to break the memory isolation in real-time across concurrent programs. Recent studies showed the successful attacks occurred in security settings - hardware settings [5] and disabling OS features like page sharing[6] and KASLR [7], securing software implementation constant programming [8], hardware implementation for sensitive data SGX [9], and compiler optimisation [10].In this paper, the Support Vector Machine algorithm is proposed which can leverage hardware features to analyse process activities at the processor core to detect malicious processes which perform side-channel attacks in the user space. The detection system is capable of capturing malicious activities from side channel attacks against cache memories, which may hold sensitive data such as secret keys. The proposed detection system utilises a profiling technique, which captures processor core level activities and feeds it to the machine learning algorithms to build a classification model. The detection system does not require any synchronisation between the victim and attacker programs to detect side channel attacks in the system. There are several factors negatively affecting the synchronisation approaches, few maybe be named as heavy workloads leverage [11]. The rest of this paper is organised as follows. Section 2presents a brief review of the related work. In section 3, the necessary background is presented. Section 4 elaborates on the proposed detection system. Section 5 demonstrates the simulation results with detailed analysis. Finally, a summary of the work is given in section 6.

2. Related Works

Side channel attacks have been studied in the laboratory and in real systems over the past twenty years. They have been practised against on-board resources such as CPU computational units, cache and main memories. The majority of these attacks have been concerned with cloud systems, with an emphasis on IaaS (Infrastructure as a Service), in which the physical resources of the same machine are logically isolated across VMs. As the cloud

grows in popularity, cloud providers become concerned with some attacks that threaten their privacy and resources.

Recent studies have shown that Operating Systems are vulnerable to side channel attacks. CPU designers as well as software companies have responded with more robust hardware designs and data fetching mechanisms in order to alleviate the attacks on sensitive data.

Earlier research has demonstrated the achievement of high resolution [3] and very fast [14] side channel attacks through a Flush+Reload attack, which has the potential to exploit the systems' page sharing characteristics. The content is a shared library of the same applications on the machine, such as an AES shared library in relation to SSL implementation in Linux. In the early stages of the cloud, the cloud providers aimed to reclaim the maximum possible amount of memory by utilising shared pages. However, the disclosure of the page sharing vulnerability has caused software industries and cloud providers to disable the page sharing feature, which was previously the systems' default setting. To mitigate side channel attacks against page sharing, the papers [20], [21] proposed the kernel space solution CACHEBAR to provide concrete protection to shared pages across VMs in PaaS. The drawback of this proposal is the OS modifications and the performance impairment, particularly in cloud systems.

A number of studies carried out over the past ten years have examined many hardware-based vulnerabilities that have permitted side channel attacks. Most often, the hardware resource targeted by the attackers are CPU caches. Microprocessor designers have made physical changes to reduce the impact of such attacks. A study by Percival [22] into L1 cache data leakage attacks suggested that microprocessor manufacturers should disable cache sharing between threads and the core to prevent any data from being evicted from the cache lines. However, by disabling cache sharing across concurrent programs, this leads to significant degradation in system performance.

In general, when the side channel attack uses hardware resources such as CPU cache memory, it relies on memory contentions in the repetitive manner, which leads to unintentional contentions. This can be easily detected by utilising a synchronisation approach. The attack processes can be detected by relying on the data collected by the victim [23]. In this context, Allaf el at. [19] studied a comparative approaches of multiple machine learning algorithms, using SPEC CPU2006 with int and fp applications, in order to stress the CPU cache memory. It was noticed that heavy workloads have a negative effect on the detection accuracy of the three machine learning algorithms applied DT, PCAANN and KNN. All algorithms performed well when no workload was running. However, with int applications, such as gcc and bzips, the accuracy degrades and it got even worse with fp applications.

3. Background

3.1. Multi-core Platforms

The mainstream microprocessors support a large number of inter-connected cores with complex memory systems within the CPU die. Each processor core has private cache L1 and L2 with one inclusive Last Level Cache (LLC) or L3 cache across the processor cores. Any communication between processor cores and other sources in the machine must travers through processor cache memories. Thus, the high frequent hardware contentions occur in all cache levels, particularly in LLC. The main sources that each processor core needs is the main memory. Microprocessor industries have provided flexibility in using CPUs by modifying the hardware settings. For instance, OS can run under process or thread mode. In process mode, two processes cannot share private caches, whereas in thread mode threads can share private L1 and L2 cache.

3.2. Real-time Scheduling

Scheduling is one of the core OS services to support and mange hardware resources across running programs. The main goal of the scheduler is to minimise power consumption [24], which is used by the resources, and offer the optimal performance by minimum stalls [25] to provide the optimal dynamic adoption Dynamic voltage and frequency scaling (DVFS)[1] [26]. Thus, OS designers and researchers intend to propose optimal scheduling algorithms to aid bottlenecks and reduce power consumption in order to utilise highest possible speed that a CPU can offer. The focus in scheduling studies is how to virtualise and share resources across processes. On the other hand, side channel attacks come into account to distort the beauty of schedulers by misusing the shared resources [27], [28].

3.3. Malicious Loop Phase Modelling

The main part of a Flush+Reload attack program body is a malicious loop (ML), which monitors secret elements from AES look-up table when the victim's processes access them. ML flushes out the memory addresses in which the look-up table is stored by utilising Clflush instruction; and then access them continuously until the attacker retrieves the whole secret key. Each ML iteration consumes the hardware resources mainly L1, L2 and LLC caches and generating hardware events such as cache miss and hit. Any clflush instruction causes an equal number of cache misses at each cache level, when the next access is achieved. This leads to the to generate an organised set of chach misses per cache layer.

1. is the adjustment process of power and speed settings on various processors in computing device

3.4. Threat Model

The mechanism of the Flush+Reload exploits hardware and OS vulnerabilities by utilising intentional hardware contentions with a victim's processes. The attacker and victim can synchronise in shared environments in which hardware resources, such as CPU caches, are fairly shared across running applications. The attacker and victim on this case run an AES algorithm to encrypt plain texts. The AES algorithm is taken from the crypto.so-shared library in the OpenSSL package and it is installed in the host OS Ubuntu 14.04. The attacker is a malicious program in the host, which analyses the hardware cache contentions to deduce the AES secret keys.

4. Detection System

Detection system is responsible for detecting side channel attacks, namely Flush+Reload. The model utilises supervised machine learning algorithms to classify the attack activities which are achieved by the attacker program in user space. The detection model continuously observes program execution attributes on active processor cores from any ML activities through communication channels and collect data samples. The samples, which are collected from the ML jobs, are aggregated by mean functions. Then the data is feed to the classifier to extract the attacks pattern. The classifier triggers an alarm to indicate the presence of side channel attack in the host system.

4.1. Experiment Setup

The experiments were carried out on a HP Proliant DL360 G7 with Intel Xeon X5650 2.66GHz processor and 16 GB of RAM, running Ubuntu 14.04 operating system.
The SPEC CPU2006 benchmark has been used for testing purposes.

4.2. Data Collection

In this study, the main data collection source is HPCs which is available in modern CPUs. In a typical Intel microprocessor, HPCs support monitoring of hundreds of CPU-related events. These events characterise program execution behaviours. However, these events are not equally beneficial to address a specific problem. For instance, some of the events might visualise the Flush+Reload execution attributes, such as L1, L2 and LLC misses, which are more sensible.

As HPCs do not require kernel module to be accessed, a kernel module is implemented to capture L1, L2 and LLC misses per each processor core and send them to the preprocessing procedure to apply average function with aggregation to find the groups belonging to the ML tasks. Finally, the aggregated data is fed to the SVM algorithm to build an efficient classifier and extract Flush+Reload activities among other workloads. SEPC CPU2006 benchmark suits has been used to stress the CPU while the data collection is performed in order to test the classifier's performance with noisy data sets.

5. Feature Selection

This section describes how to choose program execution attributes to detect the Flush+Reload attack activities in Real Time Systems (RTS) by utilising HPCs. Selecting the most relevant events to ML has an efficient affect in detecting and identifying the attack activities.

In this study, feature selection plays the key role in detecting side channel attack by profiling processor cores in RTS, in which program execution attributes are captured during their assignment to the processor cores, because no information is provided about processes, which are assigned to the processor cores such as PID. Proper features, which represent the execution attributes, support classification algorithms to extract Flush+Reload attack activities with high accuracy. Thus, it is crucial to select features, which make the distinction between attack programs and other workloads in the system. In previous work [29], [30], the feature selection relies on the facts relevant to the synchronisation between victim and attacker programs and features are selected based on the data dependency indicating the corelation between victim and attacker programs. In this paper, the feature selection does not rely on synchronisation and data dependencies; instead, it focuses on the unintentional memory contentions by the attacker program.

This study focuses on the ML Flush+Reload program, which is the core part of the program that efficiently explores the vulnerabilities. The main task of each iteration in ML is composed of clflush instruction followed by mov instruction. The clflush instruction removes the data from the hierarchical caches (L1, L2 and LLC) at a specific memory address, whereas the mov instruction accesses the flushed memory addresses from main memory. The access to the flushed memory addresses requires N misses for each cache level. We assume the N misses - L1, L2 and LLC - occur while the jobs of ML is assigned to one of the active processor core. Consequently, a very strong co-relation among L1, L2 and LLC caches can be noticed. This is the key intuition in the proposed framework to detect and identify the attacker at the core level observations.

However, clflush instruction itself might be used by the operating system when the memory management unit is required. This makes it possible for the noise to incur in the observations. Furthermore, noise may also interfere with observations if clflush instruction is used in user space by another program causing clflush instructions that were not

initiated by the attack program to be visible in the observations. Despite of the occurrence of the noise, it is possible to identify the malicious loop with great accuracy because of one of its particular characteristics. It has a repetition of not less than 25000 iterations being the minimum number of operations required to retrieve every bit making up the whole key in native systems.

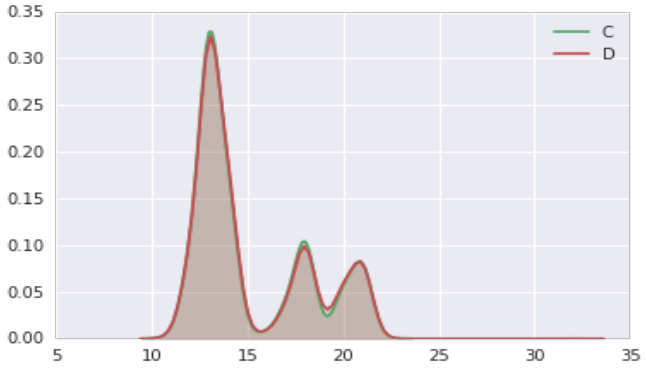

Figure 1. L1 and L3 cache misses distribution of the attacker's program in host system

Reflecting on the activities, it is assumed that the access needs N misses for each cache level without any hits. So, N misses and zero hit will occur at each cache level when the ML is assigned to the processor core. Consequently, a very strong relation among L1, L2 and LLC caches can be noticed. The series of equal number of cache misses makes the distinction between Flush+Reload program and other workloads in the system. This is the key intuition in selecting the most relevant events to Flush+Reload program. The Table 1 represents the use of four programmable events. The distribution of L2 and LLC cache misses of the attack in the host system is shown in Figure 1, where the area beneath the red line shows the LLC misses distribution and the area under the green line is L2 cache misses distribution. They are almost congruent with each other, showing that they are issuing almost the same cache miss rates.

PMCs	Annotation Events E_i	LLC Misses
Programmable	E_2	L2 RQSTS.ALL CODE RD
	E_3	L2 RQSTS.DEMAND DATA RD HIT
	E_4	L2 RQSTS.ALL DEMAND DATA RD
	E_5	Inst Retired.Any
Fixed	E_6	CPU CLK UNHALTED.CORE
	E_7	CPU CLK UNHALTED.REF TSC

TABLE 1. RELEVANT EVENTS TO SIDE CHANNEL ATTACK

As shown in Figure 1, LLC is still constant when the attack program runs with SPEC workloads, but there is a slight change in L2 cache misses. The workload variation is shown in Figure 1, where the area inside the red line is the original cache misses, and the area inside the green line represents noise filtered out by $E2$ event. 5.1. Methodology

5.2. Support Vector Machine

Support Vector Machine (SVM) is an instance based algorithm, which was introduced by Boser et al. [31] for classification purposes. SVM also can be used for regression problems [32]. The SVM algorithm has numerous application in various fields such as medical imaging, and image processing, financial analysis and web applications. The focus in this paper is on the binary classification problem. SVM attempt to separate two classes by discovering an optimal decision boundary called hyperplane. The point inside the boundaries are called support vectors. The SVM algorithm takes training data-set $S = (x_1,y_1),\ldots,(x_n,y_n)$, where $x \in X$ *and* y Y in binary classification $x \in R^n$ and $y \in 1,1$. The hyperplane separates data points from S can be defined by:

$$y_i = f(x_i,\alpha) \; for \; 1 \le i \le n \tag{1}$$

where α are parameters of the function f and n is the number of instances in S. The function f is expressed by

$$f(x,\{w,b\}) = sign(w * x + b) \tag{2}$$

The hyperplane should be maximised between the instances of two classes by $\frac{2}{||w||}$. However, in the noisy data-sets, outliers are introduced to the data-sets. As we mentioned that clflush instruction might be utilised by the OS introducing noise to the data-sets. As the result, the classifier misclassify the outliers and the performance of the classifier decreases. Therefore, polynomial function is utilised to solve the problem by introducing a slack variable ξ to identify the outliers. This is defined by the following expression:

$$y_i(w * x_i + b) \ge 1 - \xi_i for \; \xi_i \ge 0 \; and \; 1 \le i \le n \tag{3}$$

The new hyperplane can be defined by the following equation:

$$min \;\; \frac{1}{2}||w_2 + C\sum_{i=1}^{k}\xi_i \tag{4}$$

where C is the soft margin parameter which has the impact on the classifier's performance.

5.3. Model Evaluations

After building the SVM classifier, we need to make sure that the model is efficiently applied in the unseen dataset.

Therefore, there are a number of metrics that can be used to assess the model. In this study, only the SVM algorithm has been used for classification problems. As the binary classification is used to classify normal and attack activities in the system, we will describe the metrics that have been utilised in this thesis. The evaluation metrics for classification models rely on confusion matrix, which contains information about the predicted classes produced by the classifier models and the actual classes from the original data-sets as shown in Table 2. When True Positive (TP) is the case where the classifier correctly recognises the positive samples in the data-set. False Positive (FP) in

TABLE 2. CONFUSION MATRIX

	Prediction	
	Positive (P)	Negative (N)
Positive (P)	TP	FP
Negative (N)	TN	FN

this case, the classifier miss-classifies the positive classes as negative. True Negative (TN) represents the total number of the negative classes detected by the classifier correctly. False Negative (FN) when the classifier miss-classifies the Negative samples as Positive. Based on the confusion matrix, we can derive the following metrics, which are used to analyse the performance of the SVM classifier.

1) Recall/Sensitivity (True positive Rate) corresponds to the proportion of normal activities that are positive samples.

$$Recall = \frac{TP}{TP + FN} \quad (5)$$

2) Specificity (False Positive Rate) corresponds to the FN samples which are incorrectly classified as positive.

$$Specificity = \frac{FP}{FP + TN} \quad (6)$$

5.4. Experimental Design

The experimental procedure can be summarised in to two phases. In the first phase, the data-sets are collected from the kernel module, which are collected in the host system. The aggregation and mean functions are applied to the data sets; 10 different runs of six-fold cross validation (CV) were executed. In CV, each new data set is constructed from different data points for both training and testing; all data points contribute in the learners' building stage. For each iteration of CV, 70% of the original data sets were used for training and the rest were for testing.

In the second phase, SVM algorithm has been used. With each CV iteration, the new training data set is fed to the SVM algorithm to build a classifier and then the new testing data set is used to evaluate the classifier.

5.5. Experimental Results and Analysis

In this section, the results of the experiments are shown for SVM algorithm and visualised by (ROC) Area, under

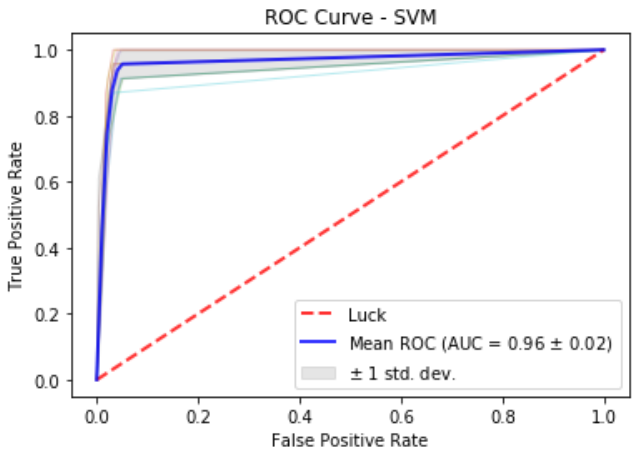

Figure 2. ROC-AUC for SVM in native system

curve (AUC) ROC-AUC. The figures depict the classifiers performance in discriminating between two process activities, which are normal, and attack.

In ROC-AUC figures, the classifiers outputs is represented as ROC curves, which represent the sensitivity (recall) and specificity calculations at incremental thresholds between zero and one across 6 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 (recall) and the X axis plots False Positive Rates (specificity). Each fold is an individual ROC and is the light blue line. It represents detection quality. The solid blue line is the calculated mean. The ideal representation is when the ROC curves has x = 0 and y = 1. This indicate that the classifiers classify normal and attack classes in unseen samples with 100

Figure 2 shows the ROC metric that evaluates the SVM classifier's ability to detect the ML activities among normal workloads, SPEC CPU 2006 benchmark applications, 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 the AUC of ROC. The model identifies ML with very high accuracy (AUC=0.99 for an

average of 10 folds, with a zero confidence interval). The noise incurred by L1 and L2 cache memories arises from the additional clflush instruction. However, the outliers are eliminated by the use of polynomial functions to predefine the hyperplane as defined by the Equation 4. In this equation, the slack variable ξ is able to identify the execution of *clflush* instruction by the host OS. Furthermore, the average function with aggregation contributes in increasing the performance of the classifier; in this case, less outliers will be introduced to the data sets.

5.6. Performance

In these experiments, the SPEC CPU2006 benchmark was running for about 10 hours with the detection model.
The detection model was running continuously in user space. The results showed that the detection model has a very low impact on the performance of the host system; even in the worst case, the performance overhead has less than 0.03 effect on the system operation.

6. Conclusion

This paper has proposed a new system detection of side-channel attacks using the bagging technique. The paper also puts forward a new profiling technique to capture the program execution attributes at the core level. Thus the attacker cannot escape from the profiling in both native and cloud systems. 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 systems with performance of up to 99\% 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. The future work will be considering the design of a model that will include detection of other side-channel attacks such as Prime+Probe.

References

[1] D. J. Bernstein, "Cache-timing attacks on aes," 2005.

[2] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Annual International Cryptology Conference*. Springer, 1999, pp. 388–397.

[3] 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.

[4] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: the case of aes," in *Topics in Cryptology–CT-RSA 2006*. Springer, 2006, pp. 1–20.

[5] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+ flush: A fast and stealthy cache attack," *arXiv preprint arXiv:1511.04594*, 2015.

[6] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 2015, pp. 605–622.

[7] O. Aciic¸mez, "Yet another microarchitectural attack:: exploiting icache," in *Proceedings of the 2007 ACM workshop on Computer security architecture*. ACM, 2007, pp. 11–18.

[8] 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.

[9] Y. Xu, M. Bailey, F. Jahanian, K. Joshi, M. Hiltunen, and R. Schlichting, "An exploration of l2 cache covert channels in virtualized environments," in *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*. ACM, 2011, pp. 29–40.

[10] 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.

[11] T. S. Messerges, E. A. Dabbish, and R. H. Sloan, "Investigations of power analysis attacks on smartcards." *Smartcard*, vol. 99, pp. 151–161, 1999.

[12] S. Weiser, R. Spreitzer, and L. Bodner, "Single trace attack against rsa key generation in intel sgx ssl," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*. ACM, 2018, pp. 575–586.

[13] T. Kim, M. Peinado, and G. Mainar-Ruiz, "Stealthmem: system-level protection against cache-based side channel attacks in the cloud," in *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, 2012, pp. 189–204.

[14] 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.

[15] D. Gruss, M. Lipp, M. Schwarz, R. Fellner, C. Maurice, and S. Mangard, "Kaslr is dead: long live kaslr," in *International Symposium on Engineering Secure Software and Systems*. Springer, 2017, pp. 161–176.

[16] T. Pornin, "Bearssl: A smaller ssl/tls library," 2016. [Online]. Available: https://bearssl.org/

[17] F. Brasser, U. Muller, A. Dmitrienko, K. Kostiainen, S. Capkun,¨ and A.-R. Sadeghi, "Software grand exposure: Sgx cache attacks are practical," *arXiv preprint arXiv:1702.07521*, 2017.

[18] 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.

[19] 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.

[20] C. Maurice, C. Neumann, O. Heen, and A. Francillon, "C5: crosscores cache covert channel," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2015, pp. 46–64.

[21] 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.

[22] C. Percival, "Cache missing for fun and profit," 2005.

[23] Y. Kulah, B. Dincer, C. Yilmaz, and E. Savas, "Spydetector: An approach for detecting side-channel attacks at runtime," *International Journal of Information Security*, pp. 1–30, 2018.

[24] Z. Zhang and J. M. Chang, “A cool scheduler for multi-core systems exploiting program phases,” *IEEE Transactions on Computers*, vol. 63, no. 5, pp. 1061–1073, 2014.

[25] T. Sherwood, E. Perelman, G. Hamerly, S. Sair, and B. Calder, “Discovering and exploiting program phases,” *IEEE micro*, vol. 23, no. 6, pp. 84–93, 2003.

[26] G. L. Valentini, W. Lassonde, S. U. Khan, N. Min-Allah, S. A. Madani, J. Li, L. Zhang, L. Wang, N. Ghani, J. Kolodziej *et al.*, “An overview of energy efficiency techniques in cluster computing systems,” *Cluster Computing*, vol. 16, no. 1, pp. 3–15, 2013.

[27] 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.

[28] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, “Spectre attacks: Exploiting speculative execution,” *arXiv preprint arXiv:1801.01203*, 2018.

[29] 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.

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

[31] B. E. Boser, I. M. Guyon, and V. N. Vapnik, “A training algorithm for optimal margin classifiers,” in *Proceedings of the fifth annual workshop on Computational learning theory*. ACM, 1992, pp. 144–152.

[32] V. Cherkassky and Y. Ma, “Practical selection of svm parameters and noise estimation for svm regression,” *Neural networks*, vol. 17, no. 1, pp. 113–126, 2004.