International Journal of Computers and Informatics (IJCI)
Vol. (3), No. (5)

IJCI

May 2024

المجلة الدولية للحاسبات والمعلوماتية
الإصدار (3)، العدد (5)

Integrated Protection Mechanisms for Mitigating Microarchitectural Attacks in Cloud Computing

Abdullah Albalawi
Department of Computer Science, Shaqra University, Kingdom of Saudi Arabia
aalbalawi@su.edu.sa

Marran Aldossari
Department of Computer Science, Shaqra University, Kingdom of Saudi Arabia
maldossari@su.edu.sa

Abstract

By utilising the multi-tenancy characteristic, cloud computing promises to reduce expenses through less spending on hardware, infrastructure, and software. Even with all of its advantages, multi-tenancy poses hazards for cloud computing. Without suitable cloud security solutions, security concerns might end up being the main factor delaying adoption. Additionally, multi-tenancy enabled by virtualisation, which is one of the key elements of a cloud, creates significant security vulnerabilities and does not provide adequate isolation between various instances running on the same physical system. The three strategies we suggest to secure shared virtualised systems against microarchitectural attacks are presented in this research as a comprehensive solution. This includes experiments for combining the three approaches and assessing them in potential operational contexts. The assessment techniques have used several host systems to assess the system overhead, CPU usage, and protection accuracy. The studies we have conducted on both Debian 10 and Ubuntu 18.04 LTS physical servers utilising the KVM hypervisor demonstrate that our comprehensive protection can identify attacks with about 97% accuracy, and depending on how many mechanisms were used in the various experimental scenario settings, the proportion of CPU consumption has varied significantly. The CPU usage rate in experiments with different scenarios has ranged from 27% to 68%, while the average system load over 5 minutes has ranged from 1.40 to 4.2. This shows our proposed mechanisms are subject to refinement and enhancement, especially in cases that require a high processing load. Note that if we had used servers with more computing power, the results would certainly have been better.

Keywords: Cloud Computing, Flush+Reload, Flush+Flush, Microarchitectural Attacks, Prime + Probe.

1. Introduction

Cloud computing technology substantially benefits businesses and organisations due to its cost efficiency, scalability, and flexibility [1]. According to Gartner [2], cloud computing is among the top ten most important and promising components of technology [3]. Multi-tenancy technology is an essential feature of a cloud. Cloud providers can maximise resource usage by dividing a shared virtualised infrastructure across several users, lowering costs [1]. Automatic resource allocation algorithms are used by cloud computing providers, establishing two or more VMs associated with unique clients sharing the same physical machine's resources [4]. However, sometimes, a malicious user may share access to the cloud's resources using allocation algorithms or VM placement policies to co-locate their VM with the target VM on the same physical server. In such a scenario, co-resident attacks and microarchitectural attacks may be used to violate confidentiality [5].

Despite all the benefits of multi-tenancy, it also introduces new vulnerabilities to cloud computing. Security concerns might become the main obstacle deterring adoption due to the lack of suitable comprehensive security solutions developed for the cloud [1], [6], [7], [8]. Even though there have been numerous methods proposed [9], [10], [11], [12], [13], [14], [15] to reduce the risks of this kind of attack, these methods have certain limitations related to how a mechanism to detect or protect the data operates, how many attacks can be detected using these mechanisms, what actions will be taken after spotting potentially suspicious behaviour, the accuracy of threat detection, and who is in control of detection and protection operations. We also believe that everyone who uses shared virtualised environments must secure data. It is therefore necessary to find integrated solutions that are trustworthy, high in accuracy, and have acceptable performance in order to design an environment with the fewest threats that could potentially compromise the security and privacy of all users of shared virtualised environments.

We address these limitations of current mitigation solutions by introducing a diverse and comprehensive detection and protection system that protects against cache side-channel and microarchitectural attacks. Our mechanism works at the level of VMs and host machines. The VM can provide self-protection by using the memory deduplication feature to monitor malicious activities targeting shared cryptographic libraries and programs, obfuscating the attack results. Also, the host protects the shared virtualised system by relying on hybrid analysis processes, namely dynamic analysis, to monitor suspicious activities by analysing hardware performance counters. It also uses static analysis to extract executable files from RAM images of the suspicious.

VM to be checked against implicit attack characteristics (opcodes) using reverse engineering tools. Then, the threat level of the VM is determined using a Softmax classification algorithm. The proposed mechanism periodically scans the disk images of VMs to ensure their integrity in terms of the presence of executable files that contain implicit attributes of microarchitectural attacks. Our approach has diverse lines of defence that are difficult for attackers to penetrate and bypass. It can also work in a shared virtualised system with acceptable performance and high accuracy.

The main contributions of our paper are as follows:

1) We present a holistic, integrated mechanism for protection against microarchitectural attacks from within the victim VM and the local host.
2) We introduce the design and implementation of the protection method.
3) We evaluate the method in multiple scenarios in terms of attack detection accuracy and performance characteristics.

The remainder of this paper is organised as follows. Section 2 provides the background to the work. In Section 3, we explain and analyse the problem. Section 4 describes the proposed protection method. Section 5 provides an overview of our experiments using the new methods. In Section 6, we discuss the evaluation of the implemented methods. Section 7 compares our methods to related works. Finally, Section 8 provides a brief conclusion and makes suggestions for future work.

2. Background

2.1 Microarchitectural Attacks:

This section reviews microarchitectural attacks that threaten shared virtualised systems due to the nature and structure of those systems. Microarchitectural attacks share certain properties, techniques, and attack environments.

2.1.1 Cache Side-channel Attack:

These attacks target the shared cache memory between users' VMs in virtualised systems, where the attacker analyses the timing information gained from retrieving data from the shared cache and the main memory [16]. When the CPU requires data to execute such instruction, the CPU can find the data in the cache or the main memory. If the data is retrieved from the cache, the CPU cycles (CPU clocks) would be low; however, if the data is not cached then it must be retrieved from the main memory, ensuring relatively larger CPU cycles are consumed to retrieve it. Then, the required data will temporarily remain in the cache memory to improve the system performance if the data is needed next time. Therefore, the attack technique exploits the time difference between retrieving the data from the cache (Cache hit) and the main memory (Cache miss) [17].

The attackers use timing information to launch attacks on the victim's VM using the cache hits and cache misses to measure the CPU cycles or the time to recover the cache memory's targeted addresses. In this attack, the attacker can break the isolation between VMs, uncover the victim's actions, obtain information about cryptographic operations, and then break the encryption key, such as by using timing information in the Table Lookup implementation of AES. In the following explanation, we discuss three primary methods that can be used to leverage cache memory and extract sensitive data.

- Prime + Probe: In this method, the attacker's VM fills the cache lines with data. The victim is then given time to carry out certain encryption processes. The attacker's VM then calculates the retrieval time for previously loaded data. The attacker will then be able to identify the cache lines utilised in the victim's encryption operations as they will know what data has been deleted from the cache memory. Neither shared libraries nor page deduplication are necessary for this method.
- Flush + Reload: The attacker takes several steps to execute this type of attack, taking advantage of shared resources and memory deduplication, as shown in Figure 1. The attack is carried out with the following steps: (1) In the beginning, the victim can use the shared program that contains a number of sensitive operations and functions that are loaded into the shared cache by simply entering one of them and executing one of the functions. (2) The attacker evicts these

physical addresses from the cache memory by using the flush command (clflush) to ensure that the addresses will be retrieved from the main memory if requested next time, as a trap for the victim to find out the data retrieved and stored in the cache memory, if the victim used one of these addresses. (3) The victim may use one or more of the sensitive program's functions, and as soon as the victim uses one of them, it will be restored to the cache memory. That means that the victim has actually fallen into the trap set by the attacker. (4) The attacker retrieves all the addresses that have been flushed while keeping track of how long it takes to retrieve each of these addresses using Time Stamp Counter (rdtsc). (5) The attacker analyses the results. If the retrieval time for any of the physical addresses is longer than the specified threshold, this means that none of them were used. However, if the retrieval time for any of them was less than the threshold, this means that it was used in the sensitive operations.

- Flush + Flush: In this method, the attacker's VM first flushes the required memory lines out of the cache. After that, it gives the victim a period to perform encryption operations. Next, the attacker's VM flushes the previous memory lines again and measures the flush instructions' execution time, by- passing direct cache accesses. This technique relies on shared libraries and memory deduplication.

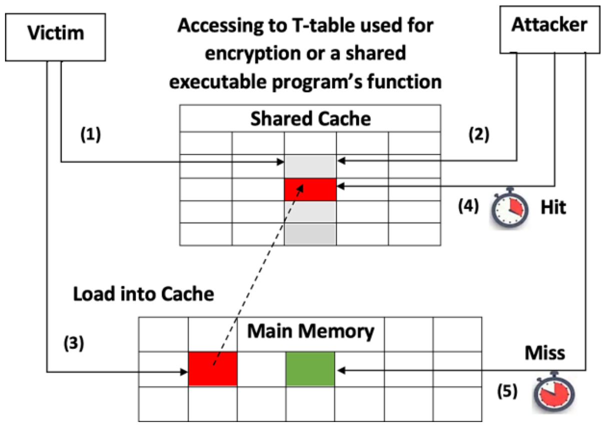

Victim
Accessing to T-table used for encryption or a shared executable program's function
Attacker
Shared Cache
(1)
(2)
(4)
Hit
Load into Cache
Main Memory
(3)
Miss
(5)

Figure (1): Cache Side-Channel Attack

2.1.2 Spectre Attack:

The Specter Attack exploits an essential optimisation technique adopted by modern CPUs called ’speculative execution’. Speculative execution occurs when a CPU retrieves data that will likely be required later, rather than waiting until it requires it. The attacker can observe the data from regions not allowed to be accessed in the memory, which leads to revealing the victim’s process. To perform the attack, first the attacker performs flush instructions to evict the desired cache lines and the target branch instruction address. The attacker often affects the CPU branch predictor using proper inputs for the conditional branch. After this, the attacker inputs an invalid value for the conditional branch to cause an incorrect prediction, thereby loading sensitive data into the shared cache. Finally, the attacker observes and keeps track of the access times of the cache lines. If certain cache lines have a short access time, the data is considered sensitive [18], [19].

2.1.3 Meltdown Attack:

Meltdown is a microarchitectural attack that abuses speculative execution features in modern CPUs to leak data that is stored in kernel memory. Meltdown is similar to a spectre attack except that it does not rely on branch prediction and aims to read the kernel memory from the userspace [18]. The attacker achieves a meltdown attack in the following manner. First, the attacker runs a code to read a byte (secret value) from privileged memory to implement a faulting instruction that will throw an exception of a segmentation fault. Although it throws the segmentation fault exception, the byte (secret value) is in the cache after the attacker multiplies the byte, the cache page size, and uses it as an index into the allocated memory block. After this, the attacker iterates through and observes the time taken to read, thus revealing confidential data [20].

2.1.4 Rowhammer Attack:

The rowhammer attack approach exploits the electrical interaction of the DRAM rows with each other, causing them to leak part of the charge when continuously accessing adjacent rows. The attacker takes advantage of this point by repeatedly accessing a DRAM row until it causes the bit to flip from one to zero or vice versa. The attacker takes the following steps to carry out this attack. First, the attacker selects a DRAM row next to the DRAM row to be flipped. After this, the attacker repeatedly accesses the DRAM row to affect the adjacent rows, thus leaking their charge. Finally, the attacker evicts the accessed DRAM out of the cache to guarantee subsequent access to the DRAM row [21].

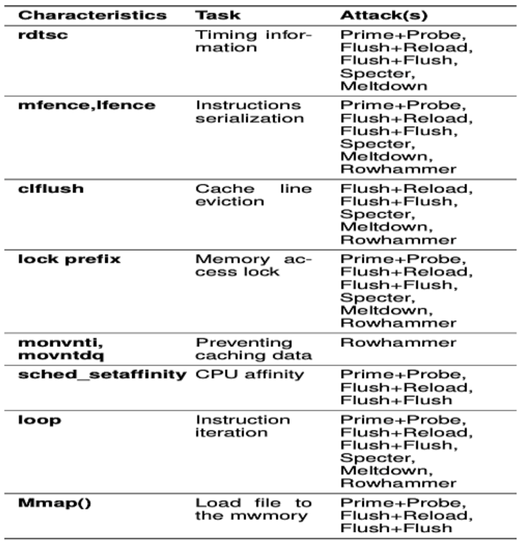
2.2 Microarchitectural Attacks Characteristics:

The implicit characteristics of microarchitectural attacks explain how these attacks have been designed and how they work. Figure 2 shows the characteristics of microarchitectural side-channel attacks (opcode). As described by Irazo- qui et al. [22] [23], the code and programs of microarchitectural side-channel attacks contain implicit characteristics and instructions that may distinguish them to some extent, which leads to them being revealed when analysing the attacks' codes. Table 1 shows a set of characteristics (opcodes) of microarchitectural attacks and their functions, as well as the microarchitectural attacks that use them.

The attacker is likely to misuse unprivileged information and legal use instructions to launch microarchitectural side- channel attacks. This information can help the attacker to design and program attack scripts by utilising instructions that can evict the cache memory, measure time for retrieving data precisely, lock the memory bus, and bypass cache access, as shown in Figure 2. All these scripts are then compiled into executable files within shared virtualised environments to perform attacks. However, identifying these scripts is possible through disassembling the executable files of the attack and recognising the interior implicit characteristics and instructions related to how they have been built.

Microarchitectural attacks comprise certain characteristics that need to be incorporated in their design. Below we review these characteristics, as discussed in [22] and [23].

Table (1): Microarchitectural Attacks Characteristics



Characteristics	Task	Attack(s)
rdtsc	Timing information	Prime+Probe, Flush+Reload, Flush+Flush, Specter, Meltdown
mfence,lfence	Instructions serialization	Prime+Probe, Flush+Reload, Flush+Flush, Specter, Meltdown, Rowhammer
clflush	Cache line eviction	Flush+Reload, Flush+Flush, Specter, Meltdown, Rowhammer
lock prefix	Memory access lock	Prime+Probe, Flush+Reload, Flush+Flush, Specter, Meltdown, Rowhammer
monvnti, movntdq	Preventing caching data	Rowhammer
sched_setaffinity	CPU affinity	Prime+Probe, Flush+Reload, Flush+Flush
loop	Instruction iteration	Prime+Probe, Flush+Reload, Flush+Flush, Specter, Meltdown, Rowhammer
Mmap()	Load file to the mwmory	Prime+Probe, Flush+Reload, Flush+Flush

- High-Resolution Timers: As shown in Figure 2 (lines 7 and 12), a set of microarchitectural attacks rely on timing information for retrieving data from the cache, the RAM, the last level of cache, and the first and second level cache precisely. Hence, the use of an instruction is required, such as a Time Stamp Counter (rdtsc), that records the timing information efficiently and has adequate precision to distinguish between data retrieval times.
- Memory Barriers: As shown in Figure 2 (lines 5, 6, 8, and 11), memory barriers include two types of instructions: mfence, and lfence. The attacker may use these instructions to serialise all store and load activities before mfence and lfence instructions in the program instruction stream. In other words, these instructions can be used to suspend out-of-order execution and collect precise timing information for retrieving data from the cache and RAM. mfence and lfence instructions can also be included in an attack's script [24], [25].
- Cache Evictions: As shown in Figure 2 (line 14), the attacker is able to exploit eviction instructions toevict the required cache line out of the cache using the Clflush instruction as a trap for the victim to retrieve data from the RAM, waiting for a while until the victim retrieves this cache line. Therefore, the attacker realises that the evicted and flushed cache line has been used by measuring the data retrieval time. The desired cache line is removed from the entire cache memory (all cache levels) using the Clflush instruction [26].
- Memory Access Lock: Attack scripts can also contain memory bus-locking instructions to ensure that the processor has exclusive ownership of the shared memory for the execution duration. The bus locking instruction consists of the Lock prefix and the following instructions: Lock prefix and the following instructions as ADC, ADD, AND, BTC, BTR, BTS, CMPXCHG, DEC, FADDL, INC, NEG, NOT, OR, SBB, SUB, XADD, XOR. However, the XCHG in-

struction does not require the Lock prefix [27].

- Non-temporal instructions allow the processor to not write data into the cache, thus directly retrieving data from the memory when requested. These instructions include monvnti and movntdq instructions [28], [29].
- CPU Affinity Assignment: Almost all microarchitectural attacks require CPU affinity to achieve cores- idency on the same CPU core to share a specific cache level with a target process. Therefore, the at- tack scripts may have function calls that accomplish CPU āffinities, such as sched setaffinity [23].
- Instruction Iteration: Practically, in microarchitectural attacks, the attacker needs to repeat some of the instructions mentioned above to execute an attack successfully. Hence, some of these instructions may be placed inside aLoop.
- Mmap () Function: Attackers use the mmap () function to load the target program into memory as a Read Only file. Then the memory deduplication feature scans and removes the replica files to be one copy shared between users; this is a critical requirement for the Flush+Reload and Flush+Flush attacks.

```
int probe(char *adrs) {
  volatile unsigned long time;

  asm __volatile__ (
    "  mfence             \n"
    "  lfence             \n"
    "  rdtsc              \n"
    "  lfence             \n"
    "  movl %%eax, %%esi  \n"
    "  movl (%1), %%eax   \n"
    "  lfence             \n"
    "  rdtsc              \n"
    "  subl %%esi, %%eax  \n"
    "  clflush 0(%1)      \n"
    : "=a" (time)
    : "c" (adrs)
    :  "%esi", "%edx");
  return time < threshold;
}
```

Figure (2): Attack Characteristics snippet from [30]

3. Problem Definition

Cloud computing relies on sharing computing resources over a network to reduce the cost of infrastructure. One form of sharing involves using a shared pool of applications and programs that may be sensitive, such as sharing cryptographic libraries using memory deduplication, which is a memory-saving feature used to optimise memory utilisation and allow an increase in the number of VMs on the same host.

Despite the economic benefits of sharing computing resources, it is known to give rise to security risks if the resources are shared with malicious users. In such a case, the malicious users can exploit the shared resources on the same physical machine as a covert channel to launch microarchitectural attacks. In some cases, such side-channel attacks are known to be able to crack most encryption algorithms, leading to confidentiality violations [5], [17], [30], [31], [32]. There are a number of countermeasures [9], [10], [12], [13], [14], [23], [33], [34] designed to mitigate microarchitectural attacks. Due to microarchitectural attacks relying on sharing computing resources, such as cache levels and instructions, to obtain the time difference of accessing data from cache and main memory, most existing defence methods are proposed based on eliminating imbalance, partitioning caches, avoiding colocation, or detecting malicious activities. However, they are also known to have significant shortcomings. First, when applied to cloud computing, significant changes are required to the computing infrastructure, which may hinder adoption by cloud providers. Also, some of these methods may cause system performance degradation and high overhead. They also have high false rates (either false positive or negative) in terms of detecting malicious activities. In addition, they need more diversity and comprehensiveness of protection against various types of microarchitectural attacks. Moreover, they do not provide systematic procedures to exclude a malicious VM after detection. Therefore, it is necessary to design a protection system that integrates diverse lines of defence with acceptable performance and high accuracy, making penetration difficult and leading to bypassing of attackers. While achieving all this, a protection system must maintain the fundamental nature of shared virtualised systems and improve security controls at the same time as enhancing performance attributes. It is also essential to conserve the economic advantages of shared virtualised systems while reducing side-channel attacks and microarchitectural attack threats, as well as providing mechanisms to monitor VMs' activities on the shared host. It is also necessary to design forensic workstations compatible with shared virtualised systems to analyse executable files of suspicious VMs and exclude any malicious VMs.

4. Methodology

The methodology used consists of a combination of three integrated methods that operate in shared virtualised systems. The first method [35] monitors and protects sensitive shared program functions (cryptographic libraries and shared executable files) within a VM. It also uses the memory deduplication feature to acquire attack readings and then analyses them using the logistic regression model. It can detect suspicious activities that sensitive shared programs are exposed to during execution of sensitive cryptographic operations in shared virtualised systems. Additionally, it can obfuscate the results of attacks obtained by the attacker. The method supports VMs to detect attacks by knowing the attacks' readings, thus providing self-protection for VMs.

Assume that two VMs run in a shared virtualised system supporting the memory deduplication feature. One of the VMs is malicious and the other is a target or victim VM. The two VMs share the last level of cache on the same host and also share the same cryptographic libraries and executable files as a result of using memory deduplication, which deletes all replicas of executable files and retains only one shared copy between them to save memory capacity. As shown in Figure 3, the VMs can access the shared memory and perform a flush-based cache attack in the system. The attacker conducts the flush-based attack (steps 1 and 6 in Figure 3).

- The attacker defines the desired memory addresses related to the shared executable file's target functions and flushes them out of the cache using the clflush instruction (the attacker may need to repeat flushing of the exact addresses multiple times to ensure the attack's success). The flushed functions' addresses are intended to be retrieved from the main memory when the victim requests and executes these functions.
- Next, the attacker waits for the victim to perform encryption operations or execute sensitive data-related functions. Then the attacker reloads the flushed functions' addresses and measures the access time (using the rdtsc instruction) to determine whether or not the victim has requested and executed those functions.

The proposed protection mechanism includes the steps numbered 2, 3, 4 and 5 in Figure 3).

- It obtains the shared functions' addresses of executable files and cryptographic libraries to be mon- itored and shielded from flush-based cache attacks.
- It recovers the monitored functions into the cache memory while measuring each function's recovery time over each specified period. As a result, the functions will be reloaded and the detection mechanism will discover the flush instructions. The measurement uses rdtsc instructions that provide a high- resolution time stamp counter. It sets an iteration sample for the monitored functions to measure the recovery time frequently. It is then measured against the system's threshold to detect whether flush in- structions have been conducted on the functions. As the detection mechanism accesses the addresses specified continuously to be monitored, the attack results are obfuscated, thus the attacker will record cache hits

for all the addresses monitored, even if the victim does not use them.

- It records the number of flushes for each monitoring function and then analyses them using logistical regression.
- It then warns the user in case of attack.

The second method [36] is a mechanism for detecting and protecting against microarchitectural attacks inside the host. The technique is a dynamic and static analysis hybrid, as shown in Figure 4. The dynamic analysis monitors VMs' activities in a virtualised system by obtaining readings from hardware performance counters relevant to the shared cache at runtime. Then, the activities of the VMs are classified as benign or suspicious after analysing the readings using a logistic regression model. When any suspicious activity is detected, the static analysis runs. The static analysis accesses the suspicious VM and extracts executable files from a disk and RAM images. It then examines whether these files contain opcodes of microarchitectural attacks. Based on the results, the threat level of these files is determined using a neural network classification model.

The third method [44] is based on a combination of static analysis with the ClamAV application. The method runs periodically and VMs are randomly selected for testing. There are two types of scans: a microarchitectural attacks scan and an antivirus scan, as shown in Figure 5. The mechanism accesses the VM, extracts executable files, checks if they contain the implicit characteristics of a microarchitectural attack, and then analyses the results using a logistic regression model to detect whether there are any malicious files and whether they contain viruses. The scan is divided into two parts based on duration: a fast scan and a full scan. This method periodically scans a shared virtualised system to eliminate attack files and viruses and identify the malicious VM. The methods operate in tandem to provide adequate protection for a shared system.

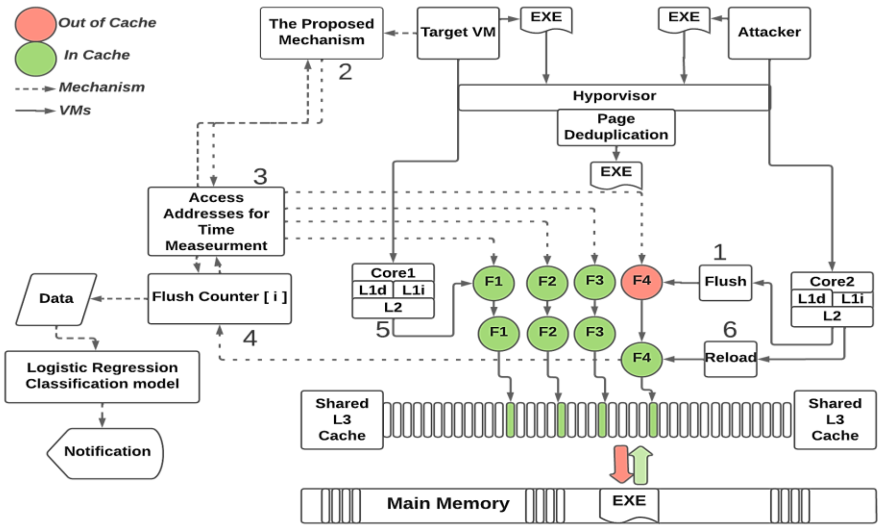


Figure (3): Flush-based Attacks Detection Method

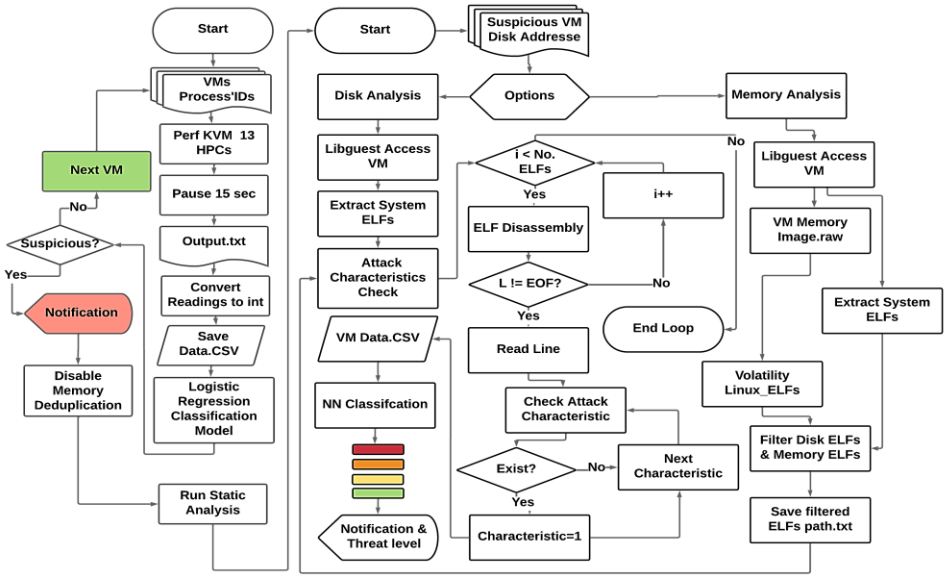


Figure (4): The Dynamic and Static Analysis Protection Method

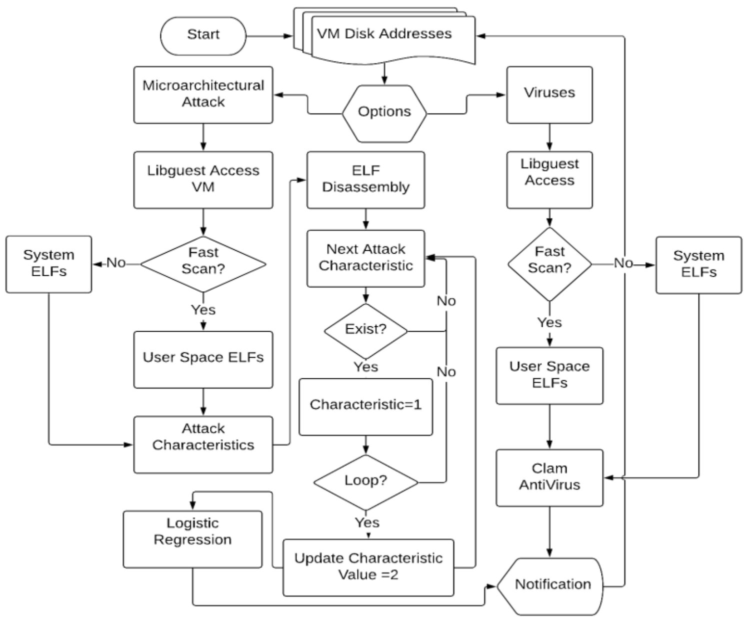


Figure (5): The Periodically Cleansing Method

5. Experimental Setup

We created shared virtualised systems and executed experiments with the QEMU-KVM hypervisor that runs KSM as a memory-saving deduplication feature. As hosts, we used Ubuntu 18.04.5 LTS, which uses an Intel Core i5- 4200M CPU, and Debian 10, which uses an Intel Core i5-4200U CPU. We then created two VMs on the same host. The VMs were running Ubuntu 18.04.5 LTS OS, with one VM as an attacker and the other as a victim. We installed several essential tools inside the VMs. For instance, the GDB (GNU Project debugger) tool facilitates finding the functions' addresses of shared executable files, thus enabling monitoring of functions. We also installed AVML (acquire volatile memory for Linux) to capture the RAM status regularly. Moreover, we installed the Linux Perf, the Libguestfs Tool, the Linux Objdump Disassembler, Radare2, Volatility Tools, and ClamAV inside the host. We conducted the attacks using the Mastik Tool designed by Yarom et al. [37]. We conducted experiments using this environment to evaluate the system's overall performance and record the system load and CPU overhead. The mechanism integrates four mechanisms that may significantly impact a system. Thus, we decided to conduct experiments and record the effect.

6. Experimental Results and Evaluation

The experiments were conducted to assess the load on the system and the overhead of the CPU. Therefore, the experiments were performed with different scenarios, as follows:

- Execution of an experiment on dynamic analysis mechanisms while we were conducting microarchi- tectural attacks. Only mechanisms for monitoring the activities of VMs were included in this experiment. The mechanism of the first method was combined with the mechanism of dynamic analysis in the second method, and the overhead of the processor was measured.
- Execution of an experiment on the mechanisms of static analysis. The static analysis of microarchitec- tural attacks in the second method was combined with the static analysis of microarchitectural attacks in the third method.
- Measuring the load on the system and the CPU's overhead while operating all mechanisms. The dynamic and static analyses were implemented together.

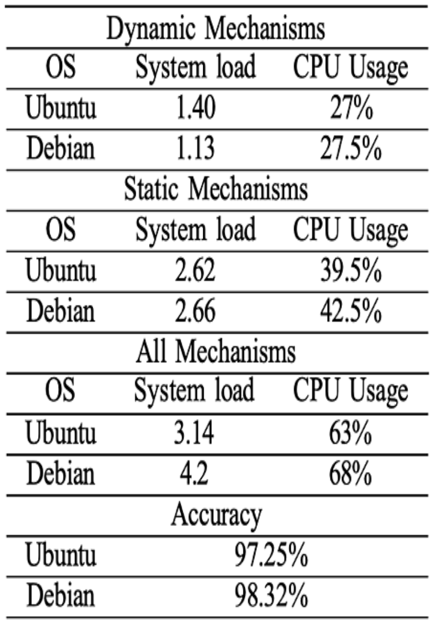
The load on the system and the overhead for the CPU were measured using the Linux Top tool, where the system load rate was recorded every 5 minutes, as shown in Table 2. Also, the detection accuracy of the proposed mechanisms was averaged based on the detection accuracy experiments of all three methods.

Based on the results shown in Figure 6, it was found that there is an increase in CPU usage and system load. However, the reason for this may be either the limited computing power of the systems used, as we utilised an Intel Core i5 processor for both hosts, or the complexity of the code. The results will be better if a server with suitable computing capabilities is utilised in both cases. Moreover, the mechanism relies on dynamic analysis to determine whether any suspicious behaviours require static analysis to ensure the presence of attack files inside a suspicious VM. This condition may reduce overhead and improve performance. Also, the static analysis that is used to detect files of microarchitectural attacks scans VMs for long-term periods, rationalising the reliance on static analysis despite its importance in protecting a shared virtualised system.

Figure 7 represents the normal distribution of system load and processor usage. The figure also shows the probability density of load and processor usage. Normal distribution was calculated by calculating the mean and standard deviation, and then the normal distribution was calculated using Microsoft Excel. We can also calculate the normal distribution after calculating the mean μ and standard deviation σ using the following equation:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

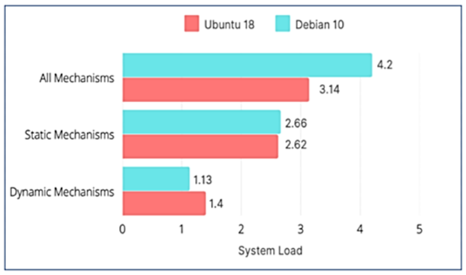
Table (2): System Overhead



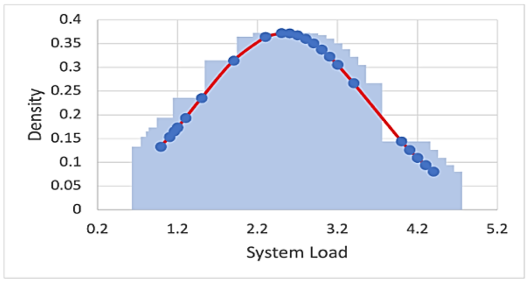
Dynamic Mechanisms		
OS	System load	CPU Usage
Ubuntu	1.40	27%
Debian	1.13	27.5%
Static Mechanisms		
OS	System load	CPU Usage
Ubuntu	2.62	39.5%
Debian	2.66	42.5%
All Mechanisms		
OS	System load	CPU Usage
Ubuntu	3.14	63%
Debian	4.2	68%
Accuracy		
Ubuntu	97.25%	
Debian	98.32%	

7. Related Work

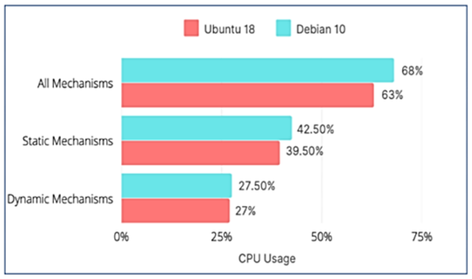
Various prior works have focused on detecting and preventing microarchitectural attacks. This section reviews some of these previous studies. It clarifies numerous problems and limitations related to the studies that motivated us to focus on these limitations and consider them in our work. Irazoqui et al. [23] introduced MASCAT, a technique for detecting microarchitectural attacks using static analysis of the executable files associated with attacks. MASCAT detects microarchitectural attack instructions (opcodes) hidden inside executable files. MASCAT is similar to an antivirus appliance for scanning files, particularly before uploading and downloading software to an app store. However, MASCAT has several areas for improvement that may prevent it from being adopted as an appropriate solution for shared virtualised systems, such as a high rate of false positives. Additionally, it results in substantial system overhead. Furthermore, it is limited to scanning executable files on a local device. It is sometimes not used to monitor the VM's disk and RAM in real-time to detect and protect against attacks in shared virtualised systems.



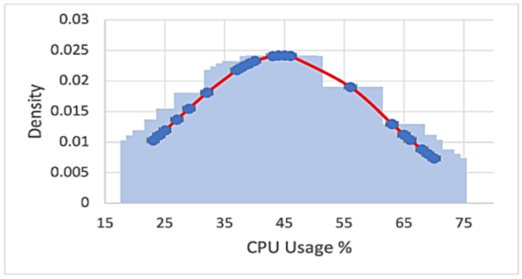
(a) System Load



(a) Normal Distribution of System Load



(b) CPU Usage



(b) Normal Distribution of CPU Usage

Figure 6: Systems Overhead

Figure 7: Normal Distribution of Systems Overhead

Additionally, Zhang et al. [14] presented CloudRadar, a detection mechanism used to significantly decrease cache- based side-channel attacks in cloud systems. It uses a combination of a signature and anomaly-based detection method supported by a hardware performance counter such as perf used in the Linux Kernel. CloudRadar uses a database to store signatures for use in comparison to identify suspicious behaviour. However, CloudRadar is unable to identify attacks that exhibit only minor changes from existing attacks because CloudRadar depends on signature-based detection, so it can only specify a specific pattern of attack program behaviour. If there is even a slight change in the attack program's behaviour pattern, the attack will be undetected, meaning that this method only detects attacks known with a specific pattern.

Cho et al. [11] proposed a detection mechanism based on machine learning for monitoring cache side-channel attacks' activities, classifying these attacks according to hardware performance counters' results obtained in real-time. The approach relies on the Intel Performance Counter Monitor (Intel PCM) to get readings from five counters related to cache performance and provides an accurate detection rate. However, it requires remarkable adjustments to the Intel PCM tool. Furthermore, the method of Cho et al. assumes that the attacker's VM must have privileged access mode to the resources, making the mechanism capable of monitoring the attacker's activities.

Also, Chiappetta et al. [10] proposed various detection approaches that detect Flush + Reload attacks' activities in real-time by relying on the hardware performance events counter. Several of these approaches are based on machine learning algorithms. Nevertheless, the approaches are restricted in terms of implementation, meaning the approaches need to be expanded to cover more than one type of cache side-channel attack other than Flush + Reload attacks.

One study [9] designed an attack detection based on the Gaussian anomaly detection algorithm. The proposed approach employs Intel Cache Monitoring Technology (Intel CMT) to obtain real-time hardware performance counter readings. This detecting technique generates accurate findings in a limited number of circumstances but is adversely influenced by background noise [13].

Another study [38] introduced a technique based on examining the opcodes of executable files obtained from VMs, particularly the VMs' RAM image, using VM introspection tools. After that, it classifies the files using classification models to determine which files are benign and which are malicious. However, the technique is unable to detect side- channel attacks.

Several studies [33], [34] have proposed mechanisms that add noise to distract the time difference between the cache hit and the cache miss in shared systems, or they eliminate fine-grained timers using fuzzy timers instead of high-resolution clocks to deal with this risk. However, implementing these solutions is not attractive because it requires modifications to the hypervisor. Also, they are not feasible for applications that require fine-grained timing in- formation [39]. Additional solutions have been proposed to reduce the probability of sharing the same resources between the victim and the attacker by designing VM placement policies [40]. The fundamental concept is to restrict the number of servers allowed for each account to use, hence reducing the attacker’s exposure to target VMs. This policy increases the possibility of co-locating VMs associated with the same user account, making it challenging to complete co-residence with the target VM. However, this policy has apparent limitations related to workload balance and power consumption [41]. Reducing the probability of sharing the same resources can be achieved by dividing the cache into several zones and assigning one for each VM, thus leading to partial isolation of VMs [42]. This approach may effectively isolate caches between distinct processes performing sensitive functions [43]. However, it limits the number of VMs that use a shared cache on the same host, and it requires significant changes in the current cloud model to be adopted effectively [41].

In brief, some of these methods need to be more com prehensive for an adequate number of cache side-channel attacks. Some of them are less attractive because they re- quire significant changes in the infrastructure of the shared virtualised system, and several of them also need to develop their results because they produce a high number of false- negative and false-positive results. Our work concentrates on protecting and detecting sufficient numbers of significant microarchitectural attacks with accurate detection results and satisfactory system performance.

8. Conclusion

Cloud computing relies on sharing resources between users of the same physical machine to reduce costs by optimising and increasing utilisation. However, sharing these resources with malicious users may lead to confidentiality violations through co-residency attacks. These attacks may exploit sharing resources, such as cache memory, to reveal a legitimate user's recent activities. Multiple techniques and factors can successfully be exploited to perform side-channel and other microarchitectural attacks. Therefore, there is still a risk despite all the benefits of sharing resources on the same physical machine. If this security risk is not properly and adequately mitigated, it could be the primary concern that obstructs cloud adoption. This paper has introduced the use of the three approaches to protect shared virtualised systems. These approaches provide self- protection for the VM on which they are used by monitoring activities within shared virtualised systems, determining the threat level of suspicious VMs, and providing periodic scanning of the virtualised system against microarchitectural attacks and viruses.

We have proposed developing three methods to provide comprehensive and holistic protection for shared virtualised systems against microarchitectural attacks. The first method detects cache attacks using memory deduplication and a logistic regression model. The second method detects and pro tects shared virtualised systems against cache side-channel attacks by integrating a dynamic and static analysis and identifying the threat level of a particular VM by using machine learning algorithms. The final method periodically cleanses shared virtualised systems against microarchitectural attacks and viruses by analysing implicit attributes of executable files using a logistic regression algorithm comprised of ClamAV.

9. Acknowledgements

We would like to extend our sincere gratitude to Shaqra University for their unwavering support throughout the research and preparation of this publication. The resources and academic environment provided by the university have played an integral role in shaping the outcome of this work. We are thankful for the opportunity to contribute to the scholarly community, and we recognize the invaluable contribution of Shaqra University in making this endeavour possible.

References

[1] H. Takabi, J. B. Joshi, and G.-J. Ahn, "Security and privacy challenges in cloud computing environments," IEEE Security & Privacy, vol. 8, no. 6, pp. 24–31, 2010.

[2] "Gartner identifies the top 10 strategic tech- nology trends for 2020." [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/

2019-10-21-gartner-identifies-the-top-10-strategic-technology-trends-for-2020

[3] K. Hashizume, D. G. Rosado, E. Ferna´ndez-Medina, and E. B. Fernandez, "An analysis of security issues for cloud computing," Journal of internet services and applications, vol. 4, no. 1, pp. 1– 13, 2013.

[4] H. Aljahdali, P. Townend, and J. Xu, "Enhancing multi-tenancy security in the cloud iaas model over public deployment," in 2013 IEEE Seventh International Symposium on Service-Oriented System Engineering. IEEE, 2013, pp. 385–390.

[5] S. Saxena, G. Sanyal, S. Srivastava, and R. Amin, "Preventing from cross-vm side-channel attack using new replacement method," Wire- less Personal Communications, vol. 97, no. 3, pp. 4827–4854, 2017.

[6] H. AlJahdali, A. Albatli, P. Garraghan, P. Townend, L. Lau, and J. Xu, "Multi-tenancy in cloud computing," in 2014 IEEE 8th International Symposium on Service Oriented System Engineering. IEEE, 2014,

pp. 344–351.

[7] A. Albalawi, V. Vassilakis, and R. Calinescu, "Side-channel attacks and countermeasures in cloud services and infrastructures," in NOMS 2022-2022 IEEE/IFIP Network Operations and Management Sympo- sium. IEEE, 2022, pp. 1–4.

[8] A. Donevski, S. Ristov, and M. Gusev, "Security assessment of virtual machines in open source clouds," in 2013 36th International Conven- tion on Information and Communication Technology, Electronics and Microelectronics (MIPRO). IEEE, 2013, pp. 1094–1099.

[9] M.-M. Bazm, T. Sautereau, M. Lacoste, M. Sudholt, and J.-M. Menaud, "Cache-based side-channel attacks detection through intel cache monitoring technology and hardware performance counters," in 3rd Int. Conf. on Fog and Mobile Edge Computing (FMEC), 2018, pp. 7–12.

[10] M. Chiappetta, E. Savas, and C. Yilmaz, "Real time detection of cache-based side-channel attacks using hardware performance coun- ters," Applied Soft Computing, vol. 49, pp. 1162–1174, 2016.

[11] J. Cho, T. Kim, S. Kim, M. Im, T. Kim, and Y. Shin, "Real-time detection for cache side channel attack using performance counter monitor," Applied Sciences, vol. 10, no. 3, p. 984, 2020.

[12] B. Gulmezoglu, A. Moghimi, T. Eisenbarth, and B. Sunar, "For- tuneteller: Predicting microarchitectural attacks via unsupervised deep learning," arXiv preprint arXiv:1907.03651, 2019.

[13] M. Mushtaq, A. Akram, M. K. Bhatti, R. N. B. Rais, V. Lapotre, and

G. Gogniat, "Run-time detection of prime+ probe side-channel attack on aes encryption algorithm," in Global Information Infrastructure and Networking Symp. (GIIS), 2018, pp. 1–5.

[14] T. Zhang, Y. Zhang, and R. B. Lee, "Cloudradar: A real-time side- channel attack detection system in clouds," in Int. Symp. on Research in Attacks, Intrusions, and Defenses, 2016, pp. 118–140.

[15] H. Wang, H. Sayadi, S. Rafatirad, A. Sasan, and H. Homayoun, "Scarf: Detecting side-channel attacks at real-time using low-level hardware features," in IEEE 26th Int. Symp. on On-Line Testing and Robust System Design (IOLTS), 2020, pp. 1–6.

[16] S. Anwar, Z. Inayat, M. F. Zolkipli, J. M. Zain, A. Gani, N. B. Anuar, M. K. Khan, and V. Chang, "Cross-vm cache-based side channel attacks and proposed prevention mechanisms: A survey," Journal of Network and Computer Applications, vol. 93, pp. 259–279, 2017.

[17] 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.

[18] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Ham- burg, M. Lipp, S. Mangard, T. Prescher et al., "Spectre attacks: Ex- ploiting speculative execution," in 2019 IEEE Symposium on Security and Privacy (SP). IEEE, 2019, pp. 1–19.

[19] C. Tang, Z. Liu, C. Ma, J. Ge, and C. Tu, "Secflush: A hard- ware/software collaborative design for real-time detection and defense against flush-based cache attacks," in International Conference on Information and Communications Security. Springer, 2019, pp. 251– 268.

[20] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh,

J. Horn, S. Mangard, P. Kocher, D. Genkin et al., "Meltdown: Reading kernel memory from user space," in 27th {USENIX} Security Symposium ({USENIX} Security 18), 2018, pp. 973–990.

[21] S. Bhattacharya and D. Mukhopadhyay, "Curious case of rowhammer: flipping secret exponent bits using timing analysis," in International Conference on Cryptographic Hardware and Embedded Systems. Springer, 2016, pp. 602–624.

[22] G. Irazoqui, T. Eisenbarth, and B. Sunar, "Mascat: Stopping microar- chitectural attacks before execution." IACR Cryptol. ePrint Arch., vol. 2016, p. 1196, 2016.

[23] ——, "Mascat: preventing microarchitectural attacks before distribu- tion," in Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy, 2018, pp. 377–388.

[24] "MFENCE - memory fence." [Online]. Available: https://www. felixcloutier.com/x86/mfence

[25] "LFENCE — load fence." [Online]. Available: https://www. felixcloutier.com/x86/lfence

[26] "CLFLUSH — flush cache line." [Online]. Available: https:

//www.felixcloutier.com/x86/clflush

[27] "LOCK — assert lock signal prefix." [Online]. Available: https:
//www.felixcloutier.com/x86/lock

[28] "MOVNTI — store doubleword using non-temporal hint." [Online].
Available: https://www.felixcloutier.com/x86/movnti

[29] "MOVNTDQ — store packed integers using non-temporal hint." [Online]. Available: https://www.felixcloutier.com/x86/movntdq

[30] 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.

[31] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+ flush: a fast and stealthy cache attack," in Int. Conf. on Detection of Intrusions and Malware, and Vulnerability Assessment, 2016, pp. 279–299.

[32] D. Philippe-Jankovic and T. A. Zia, "Breaking vm isolation-an in- depth look into the cross vm flush reload cache timing attack," Int. J. of Computer Science and Network Security (IJCSNS), vol. 17, no. 2,
p. 181, 2017.

[33] Y. Zhang and M. K. Reiter, "Du¨ppel: retrofitting commodity operating systems to mitigate cache side channels in the cloud," in Proceedings of the 2013 ACM SIGSAC conference on Computer & communica- tions security. ACM, 2013, pp. 827–838.

[34] B. C. Vattikonda, S. Das, and H. Shacham, "Eliminating fine grained timers in xen," in Proceedings of the 3rd ACM workshop on Cloud computing security workshop. ACM, 2011, pp. 41–46.

[35] A. Albalawi, V. Vassilakis, and R. Calinescu, "Memory deduplication as a protective factor in virtualized systems," in International Con- ference on Applied Cryptography and Network Security. Springer, 2021, pp. 301–317.

[36] A. Albalawi, V. G. Vassilakis, and R. Calinescu, "Protecting shared virtualized environments

against cache side-channel attacks," 2022.

[37] Y. Yarom, "Mastik: A micro-architectural side-channel toolkit," https: //cs.adelaide.edu.au/~yval/Mastik/.

[38] X. Wang, J. Zhang, and A. Zhang, "Machine-learning-based malware detection for virtual machine by analyzing opcode sequence," in Inter- National Conference on Brain Inspired Cognitive Systems. Springer, 2018, pp. 717–726.

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

[40] Y. Han, J. Chan, T. Alpcan, and C. Leckie, "Virtual machine alloca- tion policies against co-resident attacks in cloud computing," in 2014 IEEE International Conference on Communications (ICC). IEEE, 2014, pp. 786–792.

[41] ——, "Using virtual machine allocation policies to defend against co-resident attacks in cloud computing," IEEE Transactions on De- pendable and Secure Computing, vol. 14, no. 1, pp. 95–108, 2015.

[42] J. Shi, X. Song, H. Chen, and B. Zang, "Limiting cache-based side- channel in multi-tenant cloud using dynamic page coloring," in 2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W). IEEE, 2011, pp. 194–199.

[43] Y. Yarom, Q. Ge, F. Liu, R. B. Lee, and G. Heiser, "Mapping the intel last-level cache." IACR Cryptology ePrint Archive, vol. 2015, p. 905, 2015.

[44] Albalawi, A., Vassilakis, V., & Calinescu, R. (2022, April). Side-channel attacks and countermeasures in cloud services and infrastructures. In NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium (pp. 1-4). IEEE.