Detecting Cache-Based Side Channel Attacks in IaaS using Enhanced Algorithm

R. Vanathi, SP. Chokkalingam

Abstract: The Connectivity of the information among people throughout the world is made possible through computers and smart devices connected over the internet. The economic related transactions also happen over the network which needs a secured transaction medium. Therefore, lots of intrusion detection and prevention systems are implemented in order to reduce the impact of the attack. But every year the impact of attack over the shared VMs is being dramatically increased. The economic transactions occur with the help of web applications and they are divided into browser-side and server-side components. One of the major services provided by the cloud environment is Infrastructure-as-a-Service in which the virtual machines are used to provide the shared services to the multiple users. Though the VMs are secured by implementing the various security algorithms, one of the attacks, Side-channel attack, uses the leaked information acquired from the implementation of hardware component. Cache-based side channel attack is the serious attack, which tries to steal the sensitive information like credit card details, password, medical related details, etc., by establishing various algorithms like PRIME+PROBE, FLUSH+RELOAD, FLUSH+FLUSH, etc.,. The VM does speculative execution for improving the CPU performance, thus resulting in a scenario which allows the user to access the sensitive data on the cache line. So in this paper the environment is set up with 5 various scenarios with the combinations consisting of attack, no-attack, Full load, Average load and no-load. The Hardware Performance Counters (HPC) is used along with Intel CMT to monitor and distinguish the attacker VM, thus increasing the detection accuracy and reducing the system overhead.

Keywords: VM, HPC, CMT, Spectre, LLC

I. INTRODUCTION

Most of the web applications deal with sensitive data transactions and it is not a great shock to find that the sensitive information's are leaked from a high-profile web applications which includes details of investments, medications, tax, individual details, credit card details, etc., even with HTTPS protection[24]; Medical and Public Health areas usually deal with patient records. These data may contain anomalies due to several reasons such as abnormal patient condition or instrumentation errors or recording errors. Therefore anomaly detection improvement is one of the important which concentrates on improving the error detection accuracy rates.

The major cloud computing services are Software-as-a-Service, Platform-as-a-Service and Infrastructure-as-a-Service.

Among the various attacks, the major attack is DDoS attack, which concentrates on overloading the servers unnecessarily by acquiring the control over the vulnerable VM. In order to gain access to the VM, the attacker will try all the possibilities for spoofing the secret information about the victim. In worst case, the attacker will disguise like a legitimate user after gaining the complete control over the victim VM and will generate the attack very easily. As the DDoS concentrated much in overloading the servers, thus reducing the CPU performance, the Infrastructure-as-a-Service has a greater impact by this attack. To compromise the VM, one of major attacks that are used is side-channel attack. Cache line leakage is one of the major side-channel attacks, which is not based on the error in the implementation of the algorithm, whereas it depends on the vulnerabilities in the system implementation.

In the evaluation of the cache based side channel attack, the environmental set up consisting of 5 scenarios have considered as 1. Attack, Intensive-load; 2. No-attack, no-intensive load; 3.attack, no intensive-load; 4.no-attack, intensive-load; 5. attack, average-load. In all these 5 scenarios the data set consisting of events ranging from 0-7000 was given as input. There are maximum chances for the dataset to have the anomaly in it; therefore they are normalized using the Gaussian distribution algorithm to normalize the dataset. The cloud environment is established with VMs and multiple level caches with LLC being shared among the VMs. The cache hit and cache miss will vary for the 5 scenarios based on the attack and no-attack. The Hardware Performance Counters (HPC) and Intel CMT were used to monitor the attacks generated on the VM. This paper mainly focuses on improving the attack detection accuracy for the various scenarios [17] by including the classification algorithm. There are various methods to establish the attack, here in this paper Flush+Reload attack is considered due to its severity of the attack. In this paper the side channel attacks types are discussed in section II and the defence methods are compared in section III and the conclusion is discussed in section IV.

II. CACHE DESIGN WITH L1, L2 AND LLC

The caches are either hardware or software component that is used to speed up the storage and retrieval of data. It is RAM (Random Access Memory) used by the CPU to either retrieve data from the previous computation or else to copy the data. Cache is a small memory unit which is used to increase the speed of CPU by storing most frequently used data in the cache.

Revised Manuscript Received on July 13, 2019.

R. Vanathi, Research Scholar, Saveetha Institute of Medical and Technical Sciences, Chennai, TN. Department of Computer Science and Engineering, Hindustan Institute of Technology and Sciences, Chennai, TN

SP. Chokkalingam, Department of Information Technology, Saveetha Institute of Medical and Technical Sciences, Chennai, TN

Cache hit and miss is the general two terms, if the required data is found in the cache it is called as cache hit else it is known as cache miss. Cache is mainly used to speed up the performance of the computer compared to the process of retrieving the data from the slow storage medium.

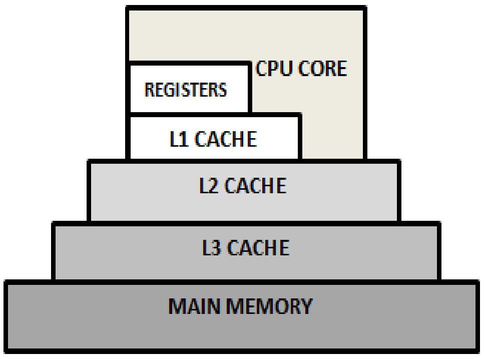


Fig. 1 Cache hierarchy up to L3 level of cache

Cache Addressing

L3 caches are physically addressed and whenever the data is transferred from the main memory to the CPU cache, an entry is created in the cache line (3 parts are Tag, Set and Offset). The cache addressing method is illustrated as the example, for 4-way set associative cache of 4096 bytes cache size. Every cache set holds 4 lines of 64 bytes each. Therefore 26 least significant 6 bits will exclusively find cache line. For 64 lines, cache is 4- way associative, 16 sets will be available. To discover a cache set, set index field of the address, $\log_2$Se successive bits starting from $\log_2$Cs bit are used. Where Set –Se, Cache line Size - Cs. If data is found in cache memory then it is retrieved from cache else from the main memory. While retrieving the data from the main memory a copy of this will be saved in cache memory, for future reference. The advantage of cache memory can be proved by differentiating the Average Access Time (AAT) of the memory pyramid with and without using cache. There are chances that there will be a cache miss often because the small cache size. So AAT of memory with cache includes "miss rate x miss penalty".

$AAT_{\text{memory without cache}}$= Hit time main memory

$AAT_{\text{memory without cache}}$ = Hit time $_{\text{cache memory}}$ + (Miss rate$_{\text{cache}}$ x miss penality$_{\text{time taken to shift to memory)}}$

AAT, for the memory with cache is less compared to the vice versa.

Evolution in Memory Latency

The main motive behind exhausting the cache is to progress the memory latency. There is a difference in the memory latency based on the type of the cache. The cache types are

1. Direct mapped caches
2. Fully associative caches
3. Set associative caches

The miss rate in the direct mapped caches is regularly more compared to the fully associative caches, though the cache size is same. On the other hand it is also not advisable to use the fully associative cache always as it will lead to more power consumption while searching the entire cache every time. The implementation of CAM hardware in fully associative cache is costly. This is a critical stage in deciding the cache design as both has their disadvantage in terms of time and power consumption.

In order to overcome latency problem in direct mapped caches, multiple levels of cache is introduced in which the L1 cache will be small and also will reside in a distance of 1-2 CPU clock cycle. The L2 and L3 cache will increase in size gradually resulting in lower miss rate. The miss rate in the various cache levels will vary according to the size ranging in the order, L1-L2-L3. So, the data will be searched in the L1 cache and if there is a cache miss, L2 and L3 caches will be searched one after another. So before accessing the main memory for every cache miss, the multiple levels of caches were checked, which in turn reduces the latency problem. The number of cache depends on the system architect.

Performance Improvements

As the technology is growing the memory is made smaller and feasible to accommodate even a very small chip. Nowadays, many of the cache level are up to 3 or 4. The AAT can be reduced by using multiple cache levels. An example is illustrated below, in which AAT of main memory and up to 3- levels of caches are calculated. Example: Main memory = 50nanosec, L1 = 1nanosec (10% miss rate), L2 = 5nanosec (1% miss rate), L3 = 10 nanosec (0.2% miss rate).

AAT (No cache) = 50nanosec

AAT (L1 cache+ Main Memory) = 1ns + (0.1 × 50ns) = 6ns

AAT (L1 cache + L2 cache + Main Memory) = 1ns + (0.1 × (5 + 0.01(50ns)) = 1.505ns

AAT (L1 cache + L2 cache + L3 cache + Main Memory) = 1ns + (0.1 × (5 + 0.01(10 + 0.002 × 50ns))) = 1.5001ns

In the above example it is proved that the AAT is improved in the order, Main memory – L1 –L2 – L3.

Write guidelines

Whenever the data is accessed from the cache, there is a possibility of data modification, which in turn has to be copied into Read Only Memory. To do this, any of the following two methods can be used:

Write Through - In this technique, the modified will be updated to the lowest-levels of memory hierarchy, thus ensuring, updated copy is stored safely throughout the hierarchy.

Write Back – In this technique, the modified will be updated to the lowest-levels of memory only when the modified data in the cache is evicted. Therefore it becomes a tedious task to update the hierarchy whenever the cache data is evicted. Therefore a flag (dirty bit) is attached to the cache block. The flag turns on whenever the cache data is modified and during the cache eviction the modified cache data alone will be copied into the lower-level hierarchy.

DDoS-The main motive of DDoS attack is to compromise a Server and in turn generate internet Bots that will automatically generate the attack to the victims, with the help of compromised servers.

Multi-Level Caches

Multi-level caches are the finest resolution to deliver a fast access to all the data that resides in the main memory. By using the cache hierarchy the CPU processing time and the latency is reduced. Multi-Level cache is used to overcome the performance bottleneck in the CPU. Accessing the main memory for every instruction execution will slow down the clock cycle speed of the CPU so with the help of cache hierarchy the data will be searched in the cache memory, which is near to the processor. Higher level caches L1, resides closer to the processor core and also it is divided as Instruction and Data cache. L2 is larger but slower than L1. L3 cache in the Intel processor is inclusive which includes the content from both L1 and L2 caches. So, the contents in the L1 and L2 will also be present in the L3 cache. Whenever data is evicted from L3, it should also be evicted from L1 and L2. Cache size will vary in terms 2n, where n ranges from 4, 8, 16, etc., either in KB or MB. The L1 is virtually indexed whereas L2 and L3 are physically indexed. L3 caches are mainly targeted in the LLC DoS attack.

In our implementation, we assume the IaaS cloud environment and the underlying CPU based on x86 architecture's, hierarchical level of cache with its CPU cycle and size of 4 core Intel Xeon i7 7400 3.5 GHz has advanced transfer cache architecture as mentioned in the table 1. This architecture is supported in the Windows 10 Operating System.

Table. 1 Different levels in cache along with their size, associativity and mapping

Hierarchy Level	Size	Set Associativity	Mapping
L1 instruction	32 KB per core	8-way	Direct-mapped
L1 Data	32 KB per core	8-way	Direct-mapped
L2	3 MB per 2 core	4-way	Non-inclusive Direct-mapped
L3	12 MB or 16 MB	16-way	Inclusive Common between all cores

Tianwei Zhang and Yinqian Zhang offered a system[19]; CloudRadar in which Signature-based and anamoly-based detection techniques are used to monitor the VMs to identify the abnormal cache behaviours and to detect the cache based side channel attack. The three modules (Victim monitor, Attacker monitor and Signature/Anomaly detector) in cloudRadar have a limitation of utilizing a dedicated CPU core for the security purpose, which in turn will increase the server overhead. Yarom and Falkner [2] established a Flush+Reload technique, which concentrates on the shared L3 cache. Initially they targeted the implementation of RSA algorithm and later this attack was used to retrive the AES key. G. I. et al., established a cross VM attack demonstrated a technique to trace the AES algorithm and hack the security key [8]. The cached and the non-cached details during the memory access was measured by generating the cflush instruction in the Flush+Reload [3] attack, which results in the increased cache miss as during every cflush instruction the data stored on the cache lines will be evicted and therefore during the next memory access cache miss occurs. Another cross VM attack by Liu et al. in [4] recovers the key in less than seconds, aims at different forms of GnuPG against execution of ElGamal 420 decryptions. However, these require manual identification of data. Therefore, in order to automatically exploit the cache vulnerabilities, an overall attack framework was presented in [5]. In this attack, private information like passwords is retrieved, leading to a greater threat to the virtual computing environments like cloud.

New extraordinary firmness attack generated on LLC removes outflow at high accuracy from the cache and in addition, a concurrent attack is generated on the information cache to identify when does the victim executes encryption to differentiate applicable memory accesses from noise. In [1] it was explained how the attack allows extracting a secret key in little minutes, applicable for both the energetic probe phase and the post-attack analysis.

III. SIDE CHANNEL ATTACK

Side-channel attacks (SCA) can happen either through the shared architectural structure or through the OS's memory management. The modern technologies mainly rely on the shared architectural structures, in which the sharing happens like cross CPU, Cross VM, Cross core, Cross user, Shared buffer in D-RAM, L3 Caches, etc...Cache side channel attacks are permitted by micro architectural design of CPU[7]. There are various types of side-channel attack which mainly focuses on stealing the victim's data/ Compromise the victim OS by breaking the secret key and then extend the attack further into DDoS by overloading the servers of the other end user's, who share the common memory with the victim and also make other server's, as victim based and will distribute the side-channel attack further.

Last Level Cache Side Channels

The cache is one of the main concepts used in cloud computing to achieve high performance web based services. As cache has become much essential, the cache miss rate has to be reduced in order to increase the latency time of the CPU. Therefore, various levels of the cache have been included in the order of size and distance (to CPU) ranging from maximum to minimum (L3-L2-L1). LLC side channel attack on the cloud environment happens when the virtual memory is shared among several users. Ghost (General hardware-oriented system transfer) is disk cloning and a backup tool used in the cloud computing environment and the Ghost memory should not be shared with the OS compromised system [9]. If the cache lines of physical memory are mapped to the ghost memory, then the operations like read or eviction should not happen to that memory.

There are several attacks, which occur on the LLC and a brief overview about the attack are as follows:

Ristenpart et al. reached an attainment rate of 40% by co-residing with the victim VM and it proves it by testing on major cloud service provider Amazon EC2 [6]. When the victim is co-found with the targeted VM, several side-channels are thrown to retrieve sensitive information. Therefore, it is evident that cross VM side-channel attacks are possible on public cloud environment.

The L2 cache actions are measured using Prime + Probe technique [4] and the usual cache activity is renowned from malicious activity broadcast by the attacker. El Gamal encryption keys remained improved by checking L1 Instruction cache in a simulated environment by Zhang et al. in [7]. Being smaller in size as compared to L3 cache, earlier L1 cache was attacked. Later, shared use of L3 was targeted.

In Cache-based side channel attack, the information on cache is monitored by generating the common side-channel attacks namely, PRIME+PROBE and FLUSH+RELOAD. These attacks can be imposed on private caches and the LLC's. To resolve this issue catch partitioning and protective software's were tried implementing but it has no much effect on the compromised OS.

The three types of attacks are time-driven, trace-driven and access-driven. In time-driven the attacker tries to retrieve the data by measuring the time taken for encryption operation. In trace-driven attacks [10], [11], the invader observer cache actions (i.e., memory lines which are accessed) of the target during its implementation and finds a sequence of cache hits and cache misses. In access-driven attacks [3], the attacker tries to find any kin between e.g., an encryption process and accessed cache lines, to abuse the outline of cache accesses of the victim.

IV. ATTACK GENERATION AND MONITORING

Flush+Reload attack

The major two possible attacks of cache-based side-channel attacks are Meltdown and Spectre attacks. FLUSH+RELOAD attack is the frequently selected attack, as recommended by original Meltdown and Spectre implementations [31, 38]. Because of the relevance of FLUSH+RELOAD, the main focus is on this attack. This attack is explained in detail for better understanding. The major two parties involved in this attack are spy and the victim. The spy has to monitor the operations of the victim. Spy has to start the attack whenever the victim starts cryptographic process. The spy and the victim share the shared memory pages, in which spy will monitor the cache. The cache memory is flushed completely using the cflush instruction and after the wait time, the attacker will measure the time taken to access the data from cache line. If it takes less time, it is understood that the victim has already read the data so it was copied into the cache already.

Flush+Reload Algorithm

```
Step 1: Assume 0xADDRESS as the shared virtual address of Victim and Spy.
Step 2: Victim access, load the data into 0xADDRESS.
Step 3: Spy evicts 0xADDRESS from caches and wait for few clock cycles.
Step 4: Spy measures time taken to access the data from 0xADDRESS.
if (Victim access == true)
            Data is in cache (Min time).
else
            Data is not in cache (Max time).
            Continue step 5.
Step 5: Spy reloads 0xADDRESS into cache.
      if (time<threshold)
            Victim accessed the data already.
      else
Victim has not yet accessed the data from shared memory line.
```

The Intel i7 processors have the privilege of exhibiting the "cflush" command from the user-level, so even unprivileged user can generate this attack. CFLUSH command will evict the cache line contents. The Intel CPUs has multilevel caches in which if the data is evicted from the LLC, contents from lower level caches are also evicted.

Speculative Execution

This execution is the widely used optimization technique, to increase the CPU performance by executing the process based on the priority and prediction about the next consecutive instruction. This execution is used to save the waiting time during the load of next instruction from the main memory to cache. There may be execution of tasks that are not necessary but can be reverted by the CPU without causing any issue. This optimization method uses a branch prediction, to fetch the predicted data from the main memory into the cache which reduces the hundreds of CPU clock cycle time wasted in waiting for the data fetch. After the Spectre and Meltdown attacks, they try to exploit the system with the leakage information e.g. CPU content [16].

The Meltdown attack uses tricks to read the sensitive information from the main memory. For example if this attack requests for the sensitive data from cache, then the protected CPU might not disclose the same. Therefore this attack may use a trick by executing a program X trying to read the value by requesting sum of 10+ "value in cache" to the address location #0x12, now if it tries to retrieve the value from cache, then again it is protected by the CPU and immediately this attack will start reading the values from the address location #0x10, and starts measuring the CPU cycle time, as the value from the address location #0x12 is already stored in the cache, its CPU cycle time will be less and therefore the attacker will retrieve the sensitive data. This will also abuse the conditional and unconditional branch prediction on Intel and ARM processors, which in turn is prevented by the implementation of KAISER in OS [12].

CPU Clock cycle (Attacker tries to access the value in cache address 0x12)

```
if (readMemory(0x12) == not possible)
      add (10+ "value in 0x12)
then Read memory values from 0x10 and
```

```
calculate CPU clock cycle
if(value already in cache)
Less CPU cycle then read the data by executing 10 – "value in 0x12"
else
        Data not required.
```

Whereas the spectre attacks are applicable to vast CPU architecture, which exploits the branching predictions and also its prevention methods are not implemented effectively, therefore the main focus is given on detecting the spectre attacks [13].

Representation of spectre attack through speculative execution (Array_size is assumed to be size of Array_1)

LISTING 1: Conditional Branch Example [13]

```
1 if (Variable 'a'<array_size)
2   b = Array_2[Array_1[a] *10]
```

In the above example, due to the branching prediction even, if the condition inside the "if statement" is wrong, predicting that the contents inside the conditional statement will be executed after the true condition. The attack is established by executing the example code with malicious 'a' value, so that Array_1[a] determine a secret byte 'z' somewhere in the victim processes memory and after that array_size and array_2 will be removed from cache, but 'z' is cached. Due to CPU branch prediction, the rival runs the code multiple times with legal 'a' values, expecting the "if condition" to be true [13]. While one implementation unit is hectic waiting for the outcome of the branch condition, the CPU does speculative execution on the next instructions. Therefore this execution logic will add, 'a' to the address of Array_1 and requests the resulting address (the location of the secret byte 'z') from memory. The value of the secret key 'z' is revealed using FLUSH+RELOAD or PRIME+PROBE attack.

JIT compiler is used in modern browsers (like Chrome, Firefox, Safari and Edge) for Javascript code execution, so the valuable details like credit card info, passwords, cookies whenever the attackers sites are visited by the user.AMD processors are believed to be not affected by this attack because they check the page accessibility before the speculative read execution, so the cache is never polluted, and data is not leaked.[14] Therefore in this paper, the main concentration in about the defence mechanisms of spectre attacks on Intel processor.

Hardware performance counters (HPC):

Hardware performance counters (HPC), are the special purpose registers that are implemented in most of the modern processors, to gather the fine-grained information's to from a statistics based on the various CPU events like, number of cache hit, cache miss, CPU clock cycle speed and context-switches. After the data is read, the HPC will be reset to zero for the next branching instruction. The major concept behind this detection approach is to periodically monitor the performance of active VMs, in order to trace malicious activities, side-channel attack. During the Virtual Machines execution, plenty of fine-grained (provided by the hardware) information's are gathered.

The hardware's used for providing high resolution cache-related information on running VMs are Intel CMT and PMCs (CPU hardware Performance (Monitoring) Counters). The prototype to detect side-channel attacks is using PMCs and Intel CMT. The purpose of implementing HPCs is that they provide information to detect fine-grained attacks.

HPCs are used to detect the malicious VM in the virtual environment by the event-descriptions; few of them are mentioned in table 1. The detection module discovers all type of malicious activity that happen to VM, cache. Here few of the hardware cache events that will be useful during the detection of cache based side channel attack [17].

Table. 2 Specific hardware performance events related to cache

Name	Event Description
LLC-misses	Last-level cache misses
L1-icache-misses	L1-instruction cache misses
LLC-references	Last-level cache references
LLC-w-accesses	Last-level cache write accesses
LLC-r-accesses	Last-level cache read accesses
L1-dcache-misses	L1-data cache misses
iTLB-cache-misses	Instruction TLB read misses
iTLB-r-accesses	Instruction TLB read accesses

As we focus on the LLC, the events considered are LLC-misses, LLC-reference and LLC accesses.

V. ANOMALY DETECTION

Anomalies are the patterns that do not fall into the normal distribution boundary. Anomalies include malicious activity like credit card fraud, cyber-intrusion, terrorist activity or breakdown of a system. The seriousness of the precise anomaly notion varies for different application domains. For example, a minute variation in the medical domain (e.g., oscillations in body temperature) is an anomaly, while related deviation in the stock market domain (e.g., fluctuations in the value of a stock) may be treated as normal. Therefore, method developed for one domain might not be applicable to another domain[20]. Along with PMC and Intel CMT, Gaussian distribution is used to remove the anomalies from the dataset, in which the data points that does not fit into the data set are considered as anomalies and it is being used in monitoring and diagnostic applications (e.g.: Healthcare applications). The first step is to remove the anomalies by forming a normal distribution. Consider, 'D' as data set; 'P' as data points. Each data point, di in the data set 'D' has 'n' features. Anomalies have to be detected from the 'P's of 'D'. The mean μi and the variance σ2i for every feature ranging from i=1,…..,n for d(1)i,……, d(p)i is used to calculate the Gaussian distribution. The ideal threshold is calculated using the false positive (fp), true positive (tp) and false negative(fn) over the precision (prec) and recall (reca) attributes.

$$p(x;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \qquad \text{-Eq 1}$$

$$F_1 = 2.\frac{prec.rec}{prec+rec} \quad prec = \frac{tp}{tp+fp} \quad rec = \frac{tp}{tp+fn} \qquad \text{-Eq 2}$$

The major four anomaly learning algorithms are DLD–SVM, CLASS-S, GML and MGML. All the four algorithms are measured with values of 'ρ', ranging from (.01 to 100), (.005 to 50) providing with an assumption of acquiring better accuracy when implementing DLD–SVM and MGML [23]. Thus combining it with Gaussian algorithms, the better results arrived during anomaly detection. Our detection prototype is composed of two threads, thread one will monitor the VMs and generate the statistics of LLC- cache hit, LLC-cache Miss, etc., the thread two will detect the anomalies using the Gaussian algorithm.

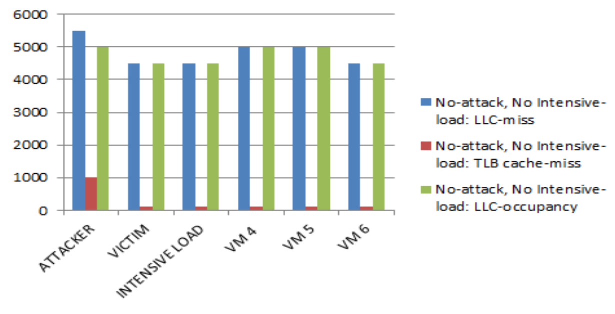
VI. DESIGN AND IMPLEMENTATION

Flush+Reload attack is generated by considering one attacker VM and one victim VM. In our experiment we consider that among the 6 VMs only one VM behave abnormally therefore the other normal VM will concentrate on the Compute Intensive Workload (CIW) calculation. CIW is required to determine whether our detection module can distinguish between a malicious and non-malicious VM.

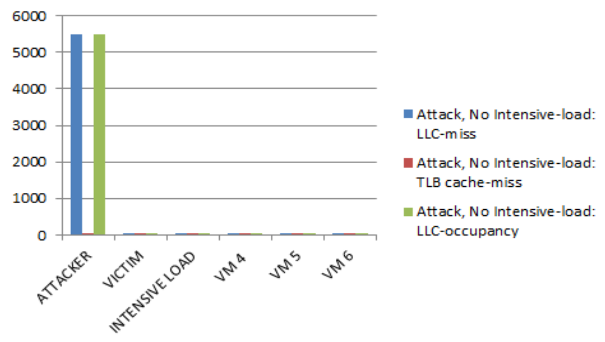
Environmental Set-up for various scenarios

In our experiment Intel Core i7 7400 3.5 GHz is used which works with the chip Multiprocessor in which LLC cache is shared among the core, whereas L1 and L2 caches are not shared. Specifications are mentioned in table 1. Ugo and Adrian [25] has proved that trying to optimize the cache sharing reduces the performance and scalability of the system. They have analysed various countermeasures for the same and one such is when the compiler tries to remove the branching conditions from executing the target code, implementing it for the real-world applications becomes challenging. So, it was proved that if LLC cache sharing is optimized to provide security then the performance will be affected, which is not advisable therefore In our experiment we have considered the scenario in which the system load varies like No Load (NL), Intensive-Load (IL) and Average Load (AL) conditions by also combining each of them with attack and no-attack scenarios.

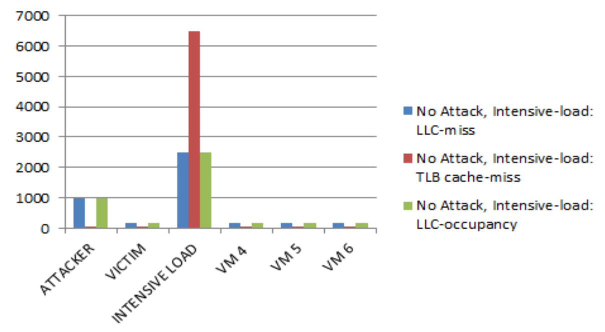
Scenario 1: No Intensive-load and No-Attack - In this experimental setup, two virtual machines, no-load and no-attack are in idle state. The attacker will not perform any side-channel attack and also there was no CIW. In such situation almost all the VM will behave with normal values for all the hardware performance events listed in the table 1.



Scenario 2: Attack and No-Intensive Load - In this experimental set up, the victim will remain idle, which means that the victim will not perform any operation so the cache lines will not have the data which the victim is interested in, therefore the attacker tries to generate the side-channel attack multiple number of times by executing the Flush+Reload attack but as the victim has not stored any of the data in cache for the fast access, the attacker will end up with the multiple LLC-miss and as a result of which it occupies more space in the LLC of the VM. In Figure, explains the LLC-misses and LLC space occupancy of the attacker compared to other VMs. As the attacker reads the same line, multiple numbers of times the TLB cache results in cache hit.



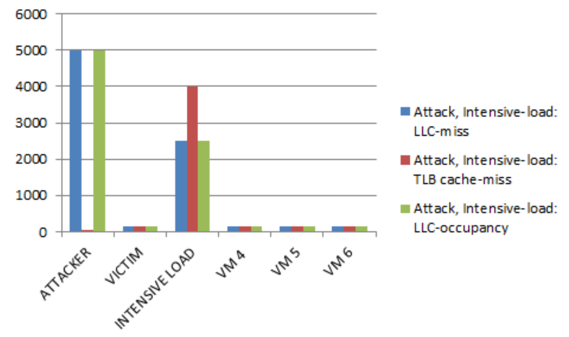
Scenario 3: No-attack, Intensive-Load–



In this environmental setup the attacker VM remains idle, without establishing any attack. One VM from other VMs establish CIW. The VM with load, have highest range of LLC-access and also LLC-misses than other VM in the environment. In the figure, it is shown that the VM with load occupies more cache line as well as faces more cache misses. But this cannot be considered as abnormal/attack because the VM with intensive load leaves greater instruction footprints, that results in increasing LLC-miss and TLB-cache miss, which could be used as feature for differentiating between the attack VM and intensive load VM.

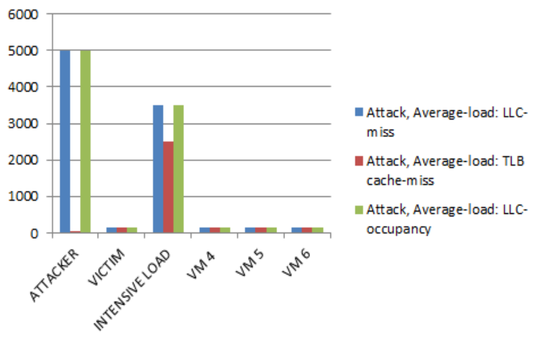
Scenario 4: Attack, Intensive-Load - In this environmental setup both the attacker and side-channel attack remains active. As a result of which both the attacker VM and intensive workload VM results has maximum range of LLC-misses and LLC-occupancy compared to the other VMs. Regarding the LLC-occupancy, the VM with the intensive load has greater value than the attacker VM and also it is observed that the attacker VM has less TLB-cache-misses than the VM with load.

This scenario has a greater chance of generating false positive alerts, though we could distinguish the attacker and Load VM with LLC-occupancy value.



For the scenario 3 and Scenario 4 there is a greatest variation in TLB-cache misses ratio (TLB-cache miss/TLB-cache access), which is used to differentiate the attacker VM and Load VM. Intensive load VM has greater TLB-cache misses compared to the VM with attack and with no attack. In scenario 4, the TLB cache miss difference is very less as shown in the figure, therefore another environmental setup is implemented with average load VM. All the above four scenarios has intensive load VM, therefore scenario 5 is implemented.

Scenario 5: Attack, Average Load-



In this environmental set up one VM is used to generate the attack and the other VM generates average load. The feature used in scenario 4 (TLB cache miss) is not applicable in this scenario because, the established load by VM is average and hence there is no huge difference in the TLB cache miss generated by the attacker and average-load VM. To improve the attack detection and to reduce the false positive rates, the anomaly based and signature based algorithms are used [27]. Hence, Naïve-Bayes classifier algorithm (supervised algorithm) is used to implement the Bayes theorem and to classifies the attack by finding attacker using signature-based label for the scenario 5[29].

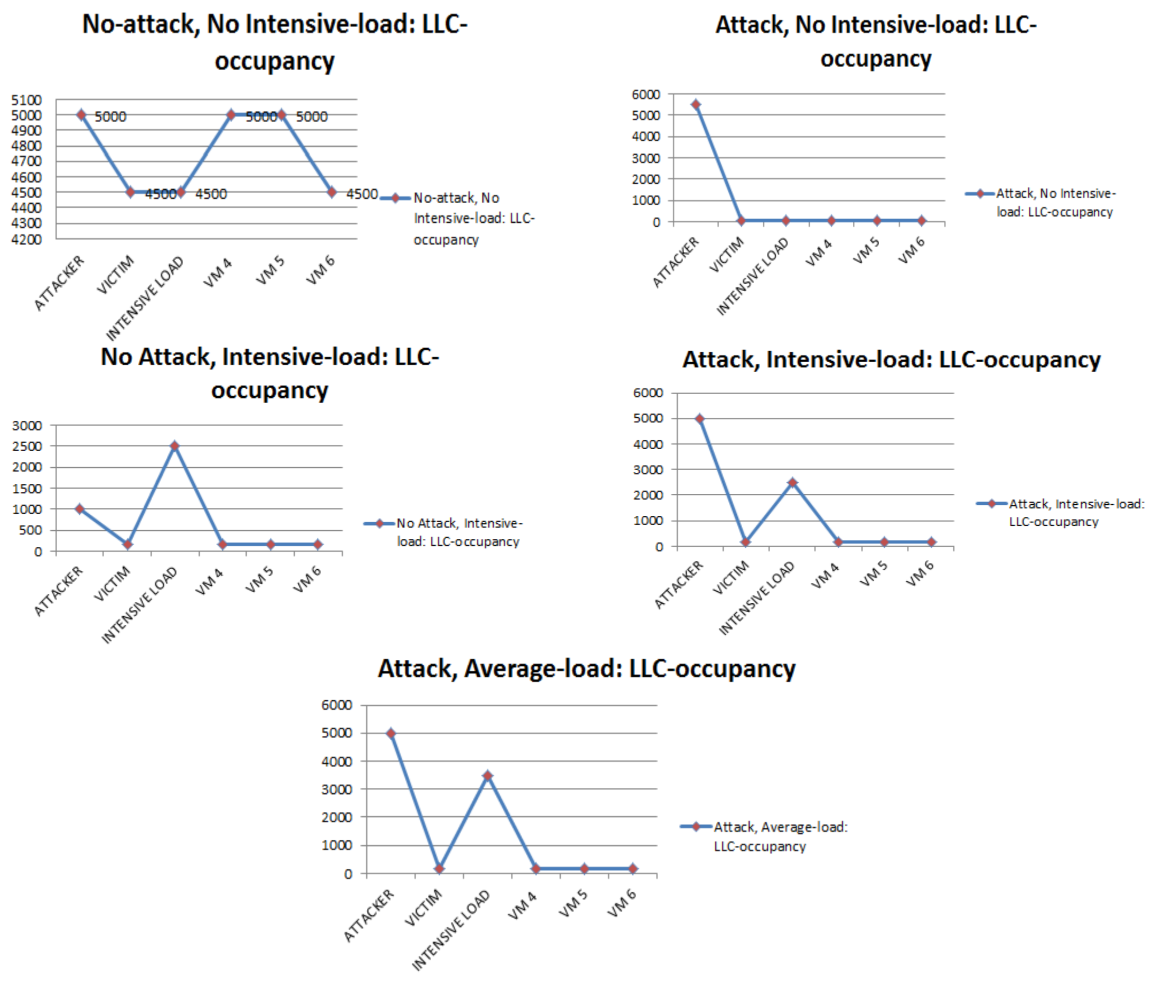


Fig. 3 LLC Occupancy for various scenarios

Our results have detection rate of 99.41%, 99.32% and 98.40% when Flush+Reload attack is generated with No-Load, Attack; Intensive-Load, Attack; Average-Load, Attack; respectively, which uses Intel CMT, High Performance Computer (HPC) and Gaussian distribution algorithm (anomaly detection from dataset) for detecting the cache-based side channel attack and also the performance overhead was >2% because of so many false positive alerts, which in turn allows 2 times, execution of the detection technique in the same scenario, in order to find the originality of attack. Therefore, by combining the classifier algorithm, to detect the attack in the early stage and to reduce the same over the VM, the detection rate accuracy was improved as 99.72%, 99.41% and 99.40% when the same Flush+Reload attack was generated on the scenario 2,4 and 5 respectively with a performance overhead of <2%.

Our results show discovery exactness of 99.51%, 99.50% and 99.44% for F+R attack in case of No-Load, Average Load and Full Load conditions, respectively, with recital overhead of < 2% at the highest detection speed, i.e., within 1% accomplishment of a single RSA encryption round. In case of Flush+Flush (stealthier) attack, our outcomes show 99.97%, 98.74% and 95.20% detection precision for NL, AL and FL conditions, respectively, with performance overhead of < 2% at the highest detection speed, i.e., within 12.5% achievement of 400 AES encryption rounds needed to complete the attack[18].

VII. CONCLUSION

In this paper, the cache based side channel attack on the VMs of the cloud environment are detected using HPC, Intel CMT and the dataset are improvised by removing the anomalies using Gaussian distribution algorithm and also the signature-based detection is done using the Naïve-Bayes classifier. The improvised dataset is implemented on the various scenarios which includes, attacker VM, Victim VM, Intensive and Average Load VM and normal VM. The accuracy rate of the attack and the performance overhead has been improvised by reducing the false positive attacks.

REFERENCES

1. Mehmet Kayaalp, Nael Abu-Ghazaleh, Dmitry Ponomarev, AamerJaleel, "A High-Resolution Side-Channel Attack on Last-Level Cache", DAC '16 Austin, Texas USA c 2016 ACM. ISBN 123-4567-24-567/08/06.
2. Y. Yarom and K. Falkner, "FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack," in Proc. of the 23rd USENIX Security Symp., San Diego, CA, USA, August 20-22, 2014., 2014, pp. 719–732..
3. D. G. et al., "Flush+ flush: a fast and stealthy cache attack," in Int. Conf. on Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, 2016, pp. 279–299.
4. F. L. et al., "Last-level cache side-channel attacks are practical," in Security and Privacy (SP), 2015 IEEE Symp. on. IEEE, 2015, pp. 605–622.
5. D. G. et al., "Cache template attacks: Automating attacks on inclusive last-level caches." in USENIX Security Symp., 2015, pp. 897–912.
6. T. R. et al., "Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds," in Proc. 16th ACM Conf. on Computer and Communications Security, ser. CCS '09. New York, NY, USA: ACM, 2009, pp. 199–212.
7. Xiaowan Dong, ZhuojiaShen, John Criswell, Alan L. Cox, Sandhya Dwarkadas,"Shielding Software From Privileged Side-Channel Attacks", Proceedings of the 27th USENIX Security Symposium, August 15–17, 2018, https://www.usenix.org/conference/usenixsecurity18/presentation/dong
8. G. I. et al., "S$a: a shared cache attack that works across cores and defies vm sandboxing–and its application to aes," in Security and Privacy (SP), 2015 IEEE Symp. IEEE, 2015, pp. 591–604.
9. Xiaowei Jiang and Yan Solihin, "Architectural Framework for Supporting Operating System Survivability", at the IEEE International Symposium on High-Performance Computer Architecture in San Antonio, Texas, Feb. 16, 2011.
10. ArtiOchani, Ramrao Adik, "Security Issues In Cloud Computing", International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC 2017), pp 783-787. VMM
11. Yiwen GAO1,2, Wei CHENG1, Hailong ZHANG1, Yongbin ZHOU, "Cache-Collision Attacks on GPU-based AES Implementation with Electro-Magnetic Leakages", 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications,2018, pp 300-306.
12. Clémentine Maurice and Stefan Mangard. 2017. KASLR is Dead: Long Live KASLR. In Engineering Secure Software and Systems: 9th International Symposium, ESSoS 2017, Bonn, Germany, July 3-5, 2017, Proceedings, Vol. 10379. Springer, 161
13. Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2018. Spectre Attacks: Exploiting Speculative Execution. ArXiv e-prints (Jan. 2018). arXiv:1801.01203
14. Younghyun Kim, Woo Suk Lee, Vijay Raghunathan , Niraj K. Jha and Anand Raghunathan, "Vibration-based Secure Side Channel for Medical Devices", June 7–11, 2015, San Francisco, CA, USA, ACM.
15. Kanthakumar Pongaliur, Zubin Abraham1, Alex X. Liu, Li Xiao, Leo Kempel, "Securing Sensor Nodes Against Side Channel Attacks", 2008 11th IEEE High Assurance Systems Engineering Symposium,pp 353-361.
16. Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom,"Spectre Attacks: Exploiting Speculative Execution", ArXiv e-prints (Jan. 2018). arXiv:1801.01203.
17. Mohammad-Mahdi Bazm, Thibaut Sautereau, Marc Lacoste, Mario Südholt, Jean-Marc Menaud, "Cache-Based Side-Channel Attacks Detection through Intel Cache Monitoring Technology and Hardware Performance Counters" ,HAL Id: hal-01762803 https://hal.inria.fr/hal-01762803v2 Submitted on 14 Sep 2018
18. Maria Mushtaq, Ayaz Akram, Muhammad Khurram Bhatti, Vianney Lapotre, Guy Gogniat, "Cache-Based Side-Channel Intrusion Detection using Hardware Performance Counters", Computer Science [cs] / Cryptography and Security [cs.CR], Feb 2019.
19. Tianwei Zhang, Yinqian Zhang, and Ruby B. Lee, "CloudRadar: A Real-Time Side-Channel Attack Detection System in Clouds", in International Symposium on Research in Attacks, Intrusions, and Defenses, pp. 118–140, Springer, 2016
20. Varun Chandola, Arindam Banerjee, Vipin Kumar, "Anomaly Detection : A Survey", A modified version of this technical report will appear in ACM Computing Surveys, September 2009.
21. Longfei Wei, Amir Hasan Moghadasi, Aditya Sundararajan and Arif I. Sarwat, "Defending Mechanisms for Protecting Power Systems against Intelligent Attacks", 2015 10th System of Systems Engineering Conference (SoSE), 2015 IEEE, pp 12-17.
22. PengLi, DebinGao, Michael K. Reiter, "Mitigating Access-Driven Timing Channels in Clouds using StopWatch", 2013 IEEE.
23. Ingo Steinwart ,Don Hush, Clint Scovel, "A Classification Framework for Anomaly Detection", Journal of Machine Learning Research 6 (2005) 211–232 Submitted 11/04; Published 3/05,Modeling, Algorithms and Informatics Group, CCS-3 Los Alamos National Laboratory Los Alamos, NM 87545, USA
24. Shuo Chen, Rui Wang, XiaoFeng Wang, Kehuan Zhang, "Side-Channel Leaks in Web Applications: a Reality Today, a Challenge Tomorrow", 2010 IEEE Symposium on Security and Privacy, 08 July 2010.
25. Ugo Fiore, Adrian Florea2, Arpad Gellert, Lucian Vintan and Paolo Zanetti, "Optimal partitioning of LLC in CAT-enabled CPUs to prevent Side-channel attacks., Springer nature Switzerland 2018, A. Castiglione et al. (Eds.): CSS 2018, LNCS 11161, pp. 115–123, 2018.

26. Sanjeev Das, Jan Werner, Manos Antonakakis, Michalis Polychronakis, Fabian Monrose, "SoK: The Challenges, Pitfalls, and Perils of Using Hardware Performance Counters for Security ", IEEE, 2019.
27. TejvirKaur, Sanmeet Kaur, Comparative Analysis of Anomaly Based and Signature Based Intrusion Detection Systems Using PHAD and Snort", precog.iiitd.edu.in
28. Mohammad Bagher Bahador, Mahdi Abadi, Asghar Tajoddin, "HLMD: a signature-based approach to hardware-level behavioral malware detection and classification", An International Journal of High-Performance Computer Design, Analysis, and Use, ISSN: 0920-8542, Springer 2019.
29. Maiwan Bahjat Abdulrazaq, Azar Abid Salih, "Combination of Multi Classification Algorithms for Intrusion Detection System", International Journal of Scientific & Engineering Research, Volume 6, Issue 1, ISSN 2229-5518, January-2015, pp 1364-1371.
30. B.Sumitra, C.R. Pethuru, M.Misbahuddin, "A Survey of Cloud Authentication Attacks and Solution Approaches", International Journal of Innovative Research in Computer and Communication Engineering, oct 2014, pp: 6245- 6253.
31. Yongle Wang, Jun ZHang Chen, "Hijacking spoofing attack and defense strategy based on Internet TCP sessions", 2013 2nd International Symposium on Instrumentation and Measurement, Sensor Network and Automation (IMSNA), pp: 507 – 509.