Protection by Detection: A Signaling Game Approach to Mitigate Co-Resident Attacks in Cloud

MGM Mehedi Hasan and Mohammad Ashiqur Rahman
Department of Computer Science, Tennessee Tech University, Cookeville, USA
Emails: mmehediha42@students.tntech.edu, marahman@tntech.edu

***Abstract*—Cloud computing is taking the technology world by storm because of the varieties of services offered by the cloud service providers (CSPs). Despite numerous benefits offered by CSPs, there are some security issues that may dissuade users from using it. In this service, different virtual machines (VMs) share the same physical resources, these VMs are known as co-resident VMs. The shared physical resources pose a significant threat to the users. As resources may belong to competing organizations as well as unknown attackers. From the perspective of a cloud user, there is no guarantee whether the co-resident VMs are trustworthy. The shared resources make privacy and perfect isolation implausible, which paves the way for co-resident attacks, where a VM attacks another co-resident VM. There is a risk that a covert side channel can be used to extract another user's secret information or launch denial of service attacks. In this paper, we analyze the co-resident attacks and corresponding defense strategies, with respect to benign and malicious VMs and the VM Monitor (VMM), using a signaling game model, named Co-resident Attacks Mitigation and Prevention. The solution to the game provides optimal defense strategies for the VMM. We evaluate the game results by conducting simulation and find that the defender can fail co-resident attacks effectively by distinguishing the benign and malicious VMs efficiently.**

***Index Terms*—Cloud Computing; Co-resident Attacks; Game Theory; Signaling Game; Nash Equilibrium.**

I. Introduction

Cloud computing is the next generation computing environment. With the varieties of services offered by the Cloud Service Providers (CSPs), people are getting attracted in using this kind of services. When several VMs share the same physical resources, these VMs are called co-resident VMs. However, several VMs sharing the same physical resources like CPU, memory, and storage devices, actually invites for new kind of attacks known as co-resident attacks. In these attacks, malicious users build various types of side channels [1], [2], [3] between their VMs and the target VM on the same server. These side channels are used for extracting sensitive information from the victim. At the heart of these attacks is the Last Level Cache (LLC) [1], which is also known as L3 cache. The L3 cache is found to be shared between all the residing VMs [1] that opens the door for the attacker to launch cache timing attacks. T. Ristenpart *et al.* were first to discover this kind of attacks [1]. They discussed how the concept of covert channel can be extended to launch attacks in clouds.

There are several techniques that a malicious VM uses to launch co-resident attacks. One such technique is PRIME+PROBE, which is based on cache-timing [4]. In this case, the attacker VM first primes the cache memory of the victim then it remains in *busy-wait* state for certain amount of time for the victim to use the cache. After that, the attacker VM primes the cache again, if the attacker's prime results in cache *hit* which means that particular cache line was not used by the victim process but if the prime results in cache *miss*, this indicates that the cache line was used by the victim process, which evicted the attacker's data thus resulting more time to bring the data. This cache-timing gives an idea of what kind of activity is going on in the victim VM [5].

There is another co-resident attacks technique known as FLUSH+RELOAD [6]. In this technique, the attacker has to reside in the same core. The attacker first flushes the cache of the victim then remains in *busy-wait* state for certain amount of time for the victim process to do its activity [3]. After that the attacker reloads his data and observes the timing, if some data result in faster loading time that means it's a *hit* and the attacker perceives that this cache line was used by the victim process. Inci *et al.* showed how this technique can be exploited to extricate RSA key [7].

Since the co-resident attacks make the benign VMs suffer, it is important for the VMM to act against them. However, the VMM should take its steps carefully so that a benign VM does not suffer and a malicious VM does not go unpunished. The main impediment that the CSP faces in this case is that the malicious VMs are also its legitimate clients. Therefore, it becomes crucial to identify potential attackers.

In order to defend co-resident attacks, we model the problem as a game, which we named as the Co-resident Attacks Mitigation and Prevention (CAMP) game. The solution to the game allows the VMM to make a smart application of defense actions by distinguishing malicious VMs from benign VMs. The CAMP problem is modeled as a signaling game [8]. According to this game framework, we analyze the interactions between the VMs and the VMM and obtain both pooling and separating equilibria [8]. We evaluate the equilibrium strategies, especially with respect to the defender, by developing a simulation program and running synthetic co-resident attack scenarios.

The rest of this paper is organized as follows: In Section II, we present the literature review on co-resident attacks. Section III models the actions of both the attacker and the VMM. We present the belief and payoff models of the game in Section IV and Section V, respectively. We present the game solution in Section VI. We evaluate the game results in Section VII, while conclude the paper in Section VIII.

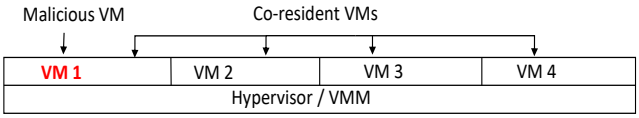


Fig. 1. Typical VM placement in a server in the cloud environment.

II. Related Work

Co-resident attacks have been studied in different research works. Y. Han et al. explored how malicious users aim to co-locate their virtual machines (VMs) with target VMs on the same physical server to extract private information from the victim using side channels [9]. The authors proposed how the use of game theory can help to reduce these attacks by efficient VM placement, which reduces the chance of co-residency of the attacker VM with the victim VM [10].

H. Singh et al. focused on a denial of service (DoS)-based attack, where a co-resident VM can congest the network channel shared among other co-resident VMs in the cloud [11]. F. Liu et al. discussed how Intel's Cache Allocation Technology (CAT) can be utilized to prevent co-resident attacks by smartly managing Class of Services (CoS) in [12]. Shi et al. designed a dynamic page coloring solution to limit cache side channel attacks [13]. Vattikonda et al. [14] and Wu et al. [15] worked on how the high resolution clocks, that many side channels rely on, can be modified or removed. Jin et al. and Szefer et al. [16] worked on redesigning the architecture of cloud computing systems [17]. Zhang et al. [18] proposed on how the side channel can be made noisy to defeat the attacker's effort to extract information.

Varadarajan et al. [19] propose a scheduling mechanism called minimum run time (MRT) that can prevent cache-based side channel attacks. Sundareswaran and Squcciarini [20] worked on identifying abnormalities in CPU and RAM utilization, system calls, and cache miss activity. Yu, et al. [21] also worked identifying similar kind of activities. In their approach, when malicious users adopt the PRIME+PROBE technique to obtain information from the victim, there happens to be certain kind of abnormalities.

Although the researchers have proposed several techniques to mitigate the co-resident attacks by preventing or limiting the chance of being co-resident, a limited research has been performed to analyze the interactions between the malicious VMs and the VMM so that optimal defense strategies can be identified. The game theory is widely proven to be useful in resolving strategically conflicting behaviors [8]. In this paper, we model the behaviors of the benign (target) and malicious (attacker) VMs and the VMM (defender) using a signaling game, a solution of which provides optimal defense strategies for the VMM with respect to the VM behaviors.

III. CAMP Game: Strategy Model

Infrastructure as a Service (IaaS) is one of the popular and prominent services offered by CSPs. Several VMs are hosted in a single server for this service. A typical VM placement in a server in the cloud is shown in Fig. 1.

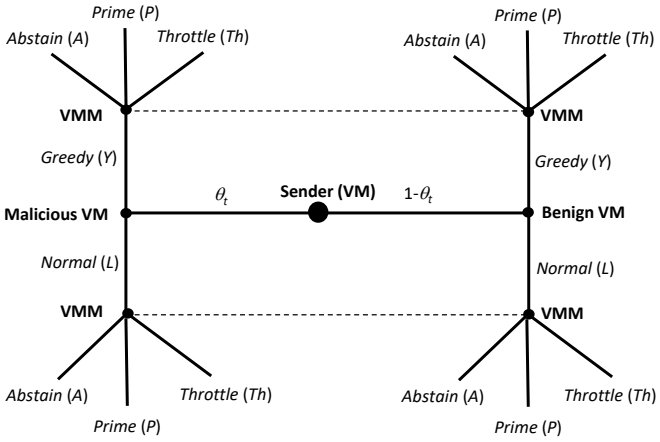


Fig. 2. Co-resident attacks defense model. A VM (sender) tries to access the physical resources and the VMM (target) provides services to several VMs. The VMM can take either of the three actions (*Abstain (A)*, *Prime (P)*, or *Throttle (Th)*.

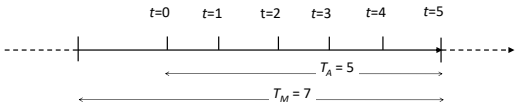


Fig. 3. An example showing the relation between T_M , T_A and t, when the maximum time for information gain is $T_M = 7\ Sec$ and the time of the game between the attacker and the hypervisor is $T_A = 5\ Sec$.

We apply the signaling game concept to model our CAMP game. The participating players are the VMs and VMM. We use the terms VMM, CSP, and hypervisor interchangeably. The VMs act as sender and the hypervisor (VMM) acts as the receiver. Here, the VMs send signals to the VMM. The VMM analyzes the signals and determines the type of the sender *i.e.*, whether malicious or benign. The CAMP game model is shown in Fig. 2. We use "he" (or "him"), when it is needed, to represent the attacker.

The VMM (as a defender) needs to deal with benign senders and malicious sender simultaneously, while it does not know explicitly about the sender type. The attacker is able to attack different target VMs and he may have different levels of interest (or benefit) in them. However, the attacker has a certain interest in a particular VM, which we call victim VM. We focus on modeling the interaction between one attacker VM and the target (VMM).

Sender's Actions: We assume that the attacker, a malicious VM, should launch and complete the required information gain in a period of time smaller than T_M, where T_M is the maximum time bound for the information gain. We define T_A as the period during which the attacking process continues. T_A cannot be more than T_M. Furthermore, T_A should be such that the attacker will have ample time to retrieve the information and use it before the victim changes the secret key. We show this scenario in Fig. 3, where T_A is 5 seconds. Hence, we define the set of actions for the attacker as $s_A = \{Greedy,\ Normal\}$. When the attacker plays $Greedy$, he is avaricious for information and he tries to access the physical resources right after the victim has accessed. On the

Table I
NOTATION TABLE

Notation	Definition
T_M	Time interval for changing the secret key
T_A	Time during which attacking process continues
t	The number of access attempts made by a VM (till time x_t)
x_t	The time when attempt t is made
$\theta_{j,t}$	Total belief (initial and dynamic belief) of the VMM about VM j and physical resource access attempt t
$\theta_{j,0}$	Initial belief about VM j
$\theta_{j,t}$	Belief about VM j at attempt t (dynamic belief)
$r_{i,j}$	Correlation coefficient of VM j with VM i
$\hat{C}_j$	Risk associated with physical resources more than usual time for VM j
$\mathbb{S}$	The set of successful access sequence
$\mathbb{A}$	The set of access sequence by an attacker
n_s	Number of times attacker needs to launch co-resident attacks
b	Number of bits in a key
$\nu(b)$	Complexity of retrieving the key when length of bits is n
G_t	Gain at attempt t
C_p	Initial cost per VM (same for every VM)
C_j	Cost of achieving co-residence
H_k	The VM's (hourly) service cost to pay at between attempt k and $k+1$
X_t	Probability of successful cache access at attempt t
ψ_t	The VMM's aggregated cost (with respect to the sender VM, if it is benign) till attempt t
$\rho_t(A_t)$	The VMM's cost (with respect to the sender VM, if it is benign) at attempt t

other hand, with the $Normal$ strategy, the attacker does not follow any such sequence.

A VM is a benign sender when it has no co-resident attack mission. The expected behavior of the benign sender is $Normal$ as it accesses the physical resources only when it needs to play its designated role. However, despite of no intention, there is still a probability that its action can seem to be greedy. We define a simple threshold-based mechanism to distinguish these two strategies of the sender, either attacker or benign. In this respect, we define G as a portion of the total potential information gain as the average gain that is required to retrieve the secret information of the target VM. We use G^B and G^A to denote the total expected gain within the game period with respect to a benign VM and a malicious VM, respectively. Obviously, according to the expected behavior, $G^B \leq G \leq G^A$. Table I summarizes the notation used throughout this paper.

If there are total of N VMs in a single server then the probability that any two of the VMs will access the physical resources consecutively is $1/N$. However, for a particular VM if the association with other VMs is greater than $1/N$ then this signals suspicious activity. We assume that consecutive access renders in information gain. We measure this activity by ϕ at an access attempt t. We define x_t as the time at attempt t and, obviously, $x_{t-1} < x_t < x_{t+1}$. Considering G^B, we define ϕ_t as the observed behavior from the VM at attempt t. We also define ϕ_t^B as the expected behavior from a benign VM. The expected behavior ϕ_t^B is computed as follows:

$$\phi_t^B = \sum_{1 \leq i \leq t} \frac{1}{N} + \delta$$

where δ is the system defined tolerance level.

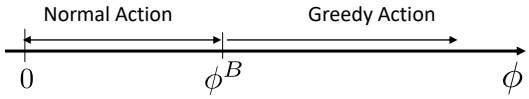


Fig. 4. Strategy of the sender. The sender is playing $Greedy$ if it asks information more than ϕ^B. Otherwise it plays $Normal$

As shown in Fig.4, we assume that if $\phi_t > \phi_t^B$, the sender plays $Greedy$, otherwise it plays $Normal$.

Hypervisor's Actions: The hypervisor takes three kinds of actions based on the belief function (θ_t). When the hypervisor does not take any kind of defensive action, we call this *Abstain* action. The hypervisor takes two kinds of defensive actions: *Prime* action and *Throttle* action. The *Prime* action slows down the cache access process for VMs, whereas, the *Throttle* action also slows down the cache access process as well as incurs additional unit time cost.

IV. CAMP GAME: BELIEF MODEL

The hypervisor maintains a belief function θ_t, which is belief at attempt t, about each of the VMs . The value of θ_t comprises of two components: the initial belief about the VMs and the dynamic belief about the VMs. The dynamic belief changes based on the behavior of the VMs. We define the belief function as follows [8]:

$$\theta_{j,t} = \min(1, \frac{e^{(\bar{\theta}_{j,0}+\bar{\theta}_{j,t})/2} - 1}{e-1}) \quad (1)$$

Here, $\bar{\theta}_{j,0}$ is the belief about VM j at time $t = 0$ and $\bar{\theta}_{j,t}$ is the belief about VM j at time $t = t$. In fact, $\bar{\theta}_{j,0}$ and $\bar{\theta}_{j,t}$ specify the initial belief and the dynamic belief, respectively. We observe from Eq. (1) that the larger is $\theta_{j,t}$, the higher is the belief about the VM towards being a malicious one.The rationale behind choosing an exponential algorithm is that, in the beginning, the greedy behavior of VM will not have much impact on the belief about the VM. However, if this kind of behavior continues then a small change in $\theta_{j,t}$ toward greedy behavior will have a larger impact on the belief function.

A. Initial belief function

The hypervisor maintains a normalized value for the initial belief ($\bar{\theta}_{j,0}$). This initial belief value depends on several factors, which are discussed below.

1) Multiple VM requirements: If there are several VM requirements from the same user, there is a probability that the user is trying to achieve co-residency. We capture this behavior by the term R_j. In this case, if there are requirement of n VMs from the same user (the requester of VM j), we define R_j as follows:

$$R_j = min\left\{1, \frac{\sum_{1 \leq k \leq n} k}{\sum_{1 \leq k \leq T_h} k}\right\}$$

Here, we consider T_h as the threshold value that denotes the maximum number of VMs that a user can request. The expression for R_j increases quickly for larger n in order to capture that: when the number of requested VMs by an user is small, the impact is low high but as the number of requested VMs by a single user increases, the impact rises significantly.

2) Similar VM requirements: If certain number of VMs have the same type of requirements then this indicates a suspicious activity and those VMs might be considered as suspicious VMs. We measure this kind of suspicious activity by S_j for VM j and define this as follows:

$$S_j = \frac{\forall j \in n \sum_{i \neq j} S_{i,j}}{\sum_{1 \leq i \leq n} \forall j \in n \sum_{i \neq j} S_{i,j}}$$

where $S_{i,j}$ is defined as follows:

$$S_{i,j} = \begin{cases} 1, & \text{if } i \text{ and } j \text{ have the same requirement} \\ 0, & \text{otherwise} \end{cases}$$

Now, the initial belief $\bar{\theta}_{j,0}$ constitutes of R_j and S_j and we define $\bar{\theta}_{j,0}$ as follows:

$$\bar{\theta}_{j,0} = min\{1, (R_j + S_j)/2\} \tag{2}$$

B. Dynamic Belief Function

Once a VM has been assigned to a user, he or she is entitled to exercise the given privilege. In this case, there is a Service Level Agreement (SLA) between the CSP and the VM user. Both the parties are bound to comply the SLA. A malicious user finds a way around the SLA to circumvent the hypervisor by launching attacks. A hypervisor can also identify a potential malicious VM based on certain behaviors which are considered as malicious VM characteristics. We need to define these malicious characteristics to compute the dynamic belief We describe these malicious VM characteristics below.

The malicious VM often accesses the physical resources right after a certain VM (the victim/target VM) so that it can extract useful data about the target VM from L3 Cache. The value of correlation coefficient between the access time of two VMs gives us an idea whether a malicious VM's access to the physical resources has anything to do with the access time of a target VM. Let $Access(VM_i, t)$ returns 1, if VM_i accesses the physical resources at attempt t and returns 0 if it does not access. The same explanation goes for $Access(VM_j, t+1)$ The purpose of the attacker VM j is to access the physical resources immediately after its target VM i has accessed it, so if a VM j shows strong correlation with a particular VM i in terms of accessing the physical resources, it indicates that the behavior of the VM j is suspicious. We define the correlation coefficient at attempt t, which is denoted by $r_{(i,j),t}$, in the following way:

$$r_{(i,j),t} = \sum_{k=1}^{k=t} (Access(VM_i, k) \wedge Access(VM_j, k+1))/t$$

Here, n_t indicates the number of try at attempt t. The higher value of $r_{(i,j),t}$ indicates that the behavior of VM j is suspicious towards VM i. Now, we can define the dynamic belief $\theta_{j,t}$ about a VM in the following way:

$$\bar{\theta}_{j,t} = \min\{1, \max_{1 \leq i \leq n} r_{(i,j),t}\} \tag{3}$$

We can find the total belief ($\theta_{j,t}$) from Eq. 1 using Eq. 2 and Eq. 3. For the rest of the paper, as we model the CAMP game for a particular malicious VM, we use θ_t to indicate the total belief about VM j.

V. CAMP Game : Payoff Model

The purpose of both the attacker VM and the VMM is to maximize their payoffs. However, maximizing payoff means keeping a balance between the cost and gain. While gathering as much information as possible is tempting for the attacker, at the same time it inherently incurs cost. The VMM uses this fact to deter the attacker from launching any attack.

A. Gain Function

Some attackers are interested to know what kind of activities his victim does. In this case, they might set the limit. The large portion of the success of the attacker is dependent on how frequently he can access the cache right after his target VM. Let $\mathbb{S}$ represents the set of access sequence the attacker successfully accesses the cache right after his target, V_v denotes the set of access sequence of the victim, n_s denotes the number of times the attacker needs to successfully launch co-resident attacks, and $\mathbb{A}$ denotes the set of the sequence the attacker visits the machine. The relationship between $\mathbb{S}$ and $\mathbb{A}$ is as follows:

$$\mathbb{S} \subseteq \mathbb{A}$$

The attacker will try to keep his $\mathbb{A}$ set to minimum. However, keeping the $\mathbb{A}$ set to minimum means to have a smaller size of $\mathbb{S}$. On the other hand, the attacker is deemed successful when the following constraint is satisfied:

$$|\mathbb{S}| \geq n_s$$

We consider that the attacker VM's target is to extract the private key. Let the length of the key is b bits. The attacker requires a number of successful attempts (n_s) to extract the full key. Every successful attempt gives the attacker certain bit information but not the whole about the key. In this respect, n_s will be a function of b.

Now, let the attacker is willing to give a maximum try of T_{max}, *i.e.*, he visits maximum of T_{max} times, n_t represents the number of try at attempt t, and n_s denotes the number of times the attacker needs to successfully launch co-resident attacks, then the gain, which is denoted as (G_t), can be represented as follows:

$$G_t = G_{t-1} + \frac{X_t}{n_s} \tag{4}$$

Eq. (4) indicates that the maximum gain is 1 and this is possible when $t = 1$ *i.e.*, the attacker obtains the private key in his first attempt. However, this reduces as the number of tries increases. Again, X_t represents the attacker's probability of success at attempt t.

If the VMM takes *Abstain* action at attempt t, then X_t is defined as follows:

$$X_t = X_{t-1}$$

When the VMM takes the *Prime* action, the attacker experiences slower data processing time. In this case, X_t is defined as follows:

$$X_t = X_{t-1} - \alpha X_0$$

Here, X_0 is the cache access probability before the game begins. We define α as follows:

$$\alpha = \frac{X_0 - X_{Th}}{n_s - 1}$$

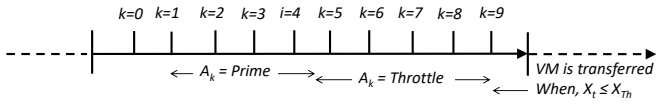


Fig. 5. An example showing how the VMM changes its course of actions. Once, an action is selected, it will be taken again as long as the conditions for taking higher action is not met.

Here, X_{Th} is the threshold value for cache access probability below which the CSP does not operate.

However, the *Throttle* action incurs added punishment to the attacker in addition to the impact caused by the previous action (*Prime*). In this case, the attacker experiences both the cache processing delay and reduced resources. The reduction in resources forces the attacker to stay longer time (t) in the server, which ultimately decreases X_t. This action makes time constraint more significant for the attacker. As t nears T_A, the attacker runs the risk of starting the attacking process from the beginning.

Once the *Prime* action is taken, the VMM continues to take *Prime* action until the condition for *Throttle* action is met. This scenario is depicted in Fig. 5.

B. Cost Function

There several costs involved in part of the attacker from launching VM to maintaining it. Let the attacker wants n VMs to co-reside with his target and the attacker needs to launch $f(n)$ VMs. Let C_p be the initial cost per VM, C_{fmax} be the maximum cost the attacker is willing to bear then the cost function for financial part (ν_0) for VM j can be defined in the following way:

$$\nu_0 = \min\left\{\frac{f(n)\times C_p}{C_{fmax}}, 1\right\}$$

The attacker often needs to make a cost risk trade-off. For example, if the attacker goes for less costly solution by spending a relatively small time in physical resources in order to reduce the financial cost, he will need to act aggressively because of the time constraint. This aggressive behavior (*i.e.,* greedy behavior) signals a malicious intent of the VM. If the attacker fails to conceal itself, his effort will be in vain, which is not desirable for him. We call this as the cost of systemic risk ($\hat{C}_j$), which is defined as follows:

$$\hat{C}_j = \left(\frac{h_j}{a}\right)^2 / h_{max}$$

Here, h_j is the average activity time observed in VM j, a is the activity hours for VM j according to the SLA, and h_{max} is the activity hours that is deemed as suspicious by the CSP (*i.e.*, when a VM crosses activity hours of h_{max}, it signals suspicious behavior).

The attacker incurs cost of his own when he attacks. Let τ_t be the cost incurred by the attacker at attempt t. We define τ_t as a simple linear concave function as follows:

$$\tau_t = \upsilon_0 + \sum_{1\le k<t} H_k(x_{k+1} - x_k)$$

Here, υ_0 is the co-residency cost (see Eq. 5), n_t is the number of try at attempt t, and H_k is the cost per time unit cost at attempt k. These coefficients take the same unit as of the budget. In order to keep the game solution simple, we abstract H_k as $H_k(x_{k+1} - x_k)$ assuming that the times of any two access attempts often have a similar interval. It is to be noted that the cost equation can be quadratic or even exponential. However, these types of cost equations will make an attacker to gain required information within a short time that in turn is susceptible to be caught by the hypervisor.

When the *Prime* action is chosen, the attacker has to spend more time to get its job done, *i.e.*, it has to work for longer hours (t). This action reduces the attack window (T_A), which ultimately reduces n_t.

When the *Throttle* action is chosen, the impact becomes multifaceted. The malicious VM is imposed physical resource constraints, which severely hampers its gain. This prompts the VM to either stay longer or suspend its activity for some time and at the same time its hourly cost also increases. Now, if H_k and H_{k11} are the per unit time costs at the try k and $k-1$, respectively, then the increase in unit time cost (ΔH_k) is defined as follows:

$$H_k = H_{k-1} + \Delta H_k$$

The value of ΔH_i depends on the discretion of of the cloud service provider. However, in our model we consider the following values of ΔH_i for different action:

$$\Delta H_k = \begin{cases} 0, & \text{if } \textit{Prime} \text{ action is taken} \\ \beta H_0, & \text{if } \textit{Throttle} \text{ action is taken} \end{cases}$$

Where β is a normalized positive value that depends on the preference of the CSP. The impact of *Throttle* action on the attacker's gain is discussed in Section V-A.

The attacker has to take into consideration the risk of being transferred to the other servers. This transfer can be for the attacker or his victim or both. But either way, transfer means the attacker's efforts so far are wasted and he needs to do things from the beginning. The transfer step is taken as part of the *Throttle* action. This step is taken when the malicious VM is no longer able to operate because of the punishment imposed by the CSP and is better off transferred to other servers. In this case, when the cache access probability (*i.e.*, X_t) goes below the threshold value (X_{Th}), the VM is transferred to another server. This causes the malicious VM to start over its whole activity to gain the required information.

The hypervisor needs to take actions cautiously otherwise a benign VM might be sufferer or an attacker VM might get away with its malicious activity. We define the cost of the hypervisor at attempt t, denoted by ψ_t, in the following way:

$$\psi_t = \sum_{1\le k\le t} \rho_k(A_k) = \psi_{t-1} + \rho_t(A_t) \tag{5}$$

Where $\rho_k(A_k)$ denotes the cost of the hypervisor for taking action A_k at attempt k.

The first line of defensive action the attacker takes is *Prime*. The purpose of the *Prime* action is to increase the *cache miss* for the attacker. When the VMM goes for *Prime* action, it primes the LLC cache to a certain level after each time the

VM accesses the cache so that the attacker experiences delay in accessing the cache. When *Prime* action is taken, $\phi_t(A_t)$ is represented in the following way:

$$\rho_k(A_k = Prime) = \alpha X_0 \tag{6}$$

If the VMM goes for *Throttle* action, it will reduce the VM's bandwidth and increase the hourly rate and $\rho_t(A_t)$ takes the following form:

$$\rho_t(A_t = Throttle) = \beta H_0 \tag{7}$$

The costs defined in Eq. (6) and (7) are deemed as the cost of the hypervisor as long as the VM being punished is benign. However, if the punished VM is malicious, the hypervisor disregards this cost.

C. Payoff Function

We model the attacker VMs payoff by U_A as follows:

$$U_A = G_t - \tau_t \tag{8}$$

Where τ_t and G_t have been defined in Eq. (5) and Eq. (4), respectively and both τ_t and G_t are normalized values between 0 and 1. All the terms in Eq. (8) are normalized values between 0 and 1. We model the hypervisor's payoff by U_D as shown in the below:

$$U_D = \lambda(-G_t + \tau_t) - (1 - \lambda)\psi_t \tag{9}$$

Where ψ_t takes any of the forms defined in Eq. (6) and Eq. (7) based on *Prime* action and *Throttle* action, respectively. Here, λ indicates the VMM's preference for taking defensive action.

VI. Analysis of the CAMP Game

In this section, we present the equilibria of the CAMP game and their interpretations. To predict the outcome of the CAMP game, we use the well-known concept of Nash Equilibrium (NE): A strategy profile constitutes a Nash equilibrium if none of the players can increase his/her payoff by unilaterally changing his/her strategy [8]. The solution of the CAMP provides the following two theorems. The detailed solution will be found in [22].

Theorem 1. *[(Greedy, Normal), (Prime, Abstain)][1] is the separating equilibrium of the CAMP game, if the following conditions hold:*

1. $X_0 \geq 0$
2. $\alpha \geq 0$

[(Greedy, Normal), (Throttle, Abstain)] is the separating equilibrium of the CAMP game, if the following conditions hold:

3. $n_s \geq \frac{-\alpha X_0}{\beta H_0}$
4. $\beta \geq 0$

The VMM takes top down approach *i.e.*, the condition is first checked for *Throttle* action, if the necessary conditions are met, the target takes *Throttle* action and and so on. Theorem 1 shows that at the separating equilibrium the VMM (target) defends (*i.e.*, plays either of *Prime* or *Throttle*) if the sender VM plays *Greedy* - the VM is expected to be malicious. It plays *Abstain* if the sender plays *Normal* - the VM is expected to be benign. If we look into the conditions at Theorem 1, we can easily see that *Throttle* action will always be taken when the sender is greedy. This is because the conditions are always true for the *Throttle* action and thus the turn of the *Prime* action will not come. This strategy is obvious because when the defender (VMM) beliefs that the *Greedy* action is taken only by the attacker (a malicious VM), it has no advantage of taking the *Prime* option, a less damaging action against the attacker. However, since a benign sender can seldom behave greedy, a defense mechanism using Theorem 1 (a separating equilibrium) will be rigid, without considering the past attitude, which is reflected in belief (θ_t). The following pooling equilibrium (Theorem 2) solves this problem.

Theorem 2. *[(Greedy, Greedy), (Prime, Prime), θ_t] and [(Normal, Normal), (Abstain, Abstain), μ] are the pooling equilibrium of the CAMP game, if the conditions below hold:*

1. $\frac{\theta_t}{1-\theta_t} \geq \frac{1-\lambda}{\lambda} n_s$
2. $\frac{\mu}{1-\mu} \leq \frac{(1-\lambda)\alpha X_0 n_s}{G_{t-1}+X_{t-1}}$

[(Greedy, Greedy), (Throttle, Throttle), θ_T] and [(Normal, Normal), (Abstain, Abstain), μ] are the pooling equilibrium of the CAMP game, if the following conditions hold:

3. $\frac{\theta_t}{1-\theta_t} \geq \frac{1-\lambda}{\lambda} \frac{n_s \beta H_0}{\alpha X_0 + \beta X_0)}$
4. $\frac{\mu}{(1-\mu)} \leq \frac{1-\lambda}{\lambda} \frac{\beta H_0 n_s}{G_{t-1}+X_{t-1}}$

Theorem 2 presents the pooling equilibrium along with necessary conditions. Here, the VMM (target) believes that the sender plays *Greedy* for each given type and the expected behavior of the VMM (target) is defending (*i.e.*, plays either of *Prime* or *Throttle*) at this equilibrium. In this case, the posterior probability of a VM (sender) being an attacker VM is θ_t. If the VMs (senders) of both types would play *Normal* and the posterior probability of a VM (sender) being an attacker VM would be assumed as μ and the optimal behavior of the defender would be *Abstain*, given the conditions hold at Theorem 2. A VM's increase in access duration and the Target's increase in defense preference, both will impact the action of the attacker.

The hypervisor analyzes the behavior of the VMs and updates the belief function. As Theorem 2 specifies, when the belief is low, the hypervisor abstains from taking any punitive measure. When the belief increases for a VM over a certain value (*i.e.*, satisfies the necessary conditions), the hypervisor will take the *Prime* action that restricts the use of physical resources of that VM. This action is the first line of defense that the hypervisor employs. If the belief about the sender VM keeps growing toward being an attacker one, the VMM takes the *Throttle* action, which restricts the use of physical resources of that VM as well as increases the service charge. However, if the probability of accessing the physical resource reduces beyond a threshold, the VMM will transfer the VM to a new physical host.

[1]VM strategy profile (a, b) means that it plays a for the type θ_T and b for the type $1 - \theta_T$. In case of the hypervisor, (a, b) means that it plays a following the $Greedy$ action and b following the $Normal$ action of the VM.

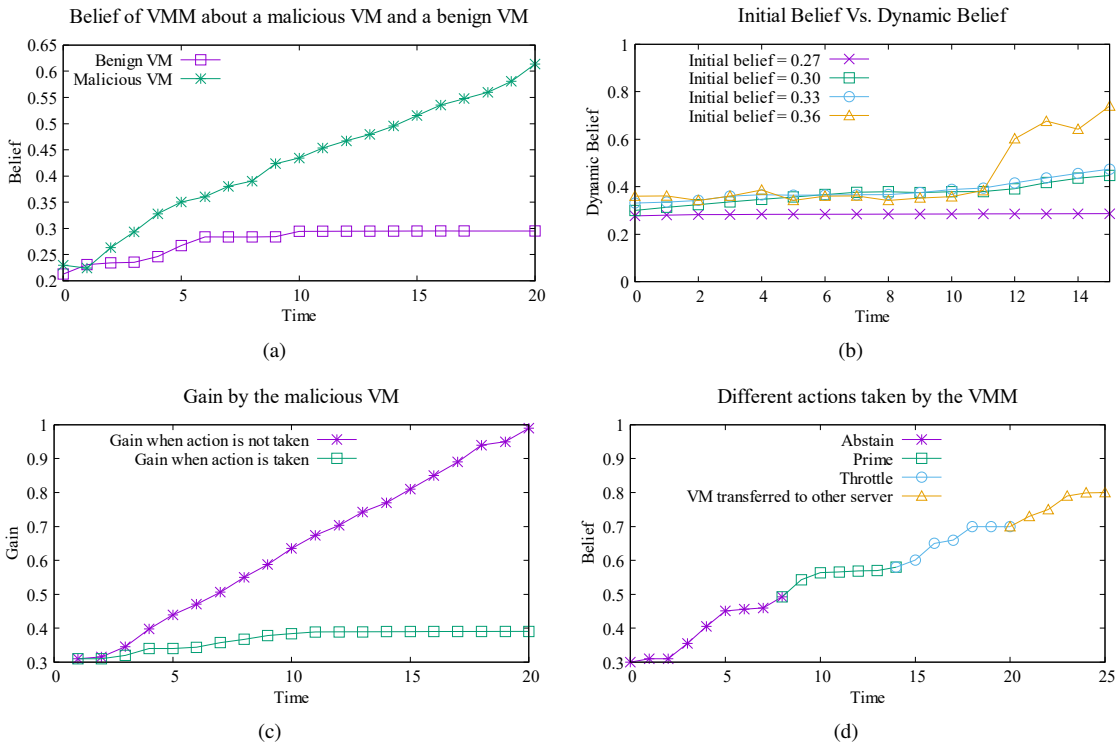


Fig. 6. (a) Change of belief of a VMM towards a benign VM and a malicious VM, (b) impact of initial belief on dynamic belief, (c) information gain achieved by a malicious VM, and (d) actions taken by the VMM at different stages,

Table II
PARAMETER VALUES USED IN SIMULATIONS

Parameter	ϕ^B	N	δ	T_M	T_A	λ	α
$Value$	0.33	8	0.03	7	5	0.5	16

VII. EVALUATION OF THE GAME RESULTS

We conduct the evaluation of the CAMP game and analyze the result. We do not simulate how the attacker achieves co-residence with its target VM.

A. Methodology:

We implement the solution of the game in C/C++. We arbitrarily choose the values for the different parameters and observe the behavior. It is worth mentioning that, these values are purely experimental and do not represent the actual values. We create $8-10$ VMs and 1 VM is randomly selected as the malicious VM. This malicious VM selects its target VM arbitrarily. We then observe the malicious VM's interactions (discussed in Section IV-B) with its target VM. As we do not simulate the achieving co-residence part, we assign initial belief (see Section IV-A) to every VM arbitrarily. The initial beliefs for different VMs are selected between 0.2 and 0.4 and assigned randomly. The values of the different parameters are shown in Table II.

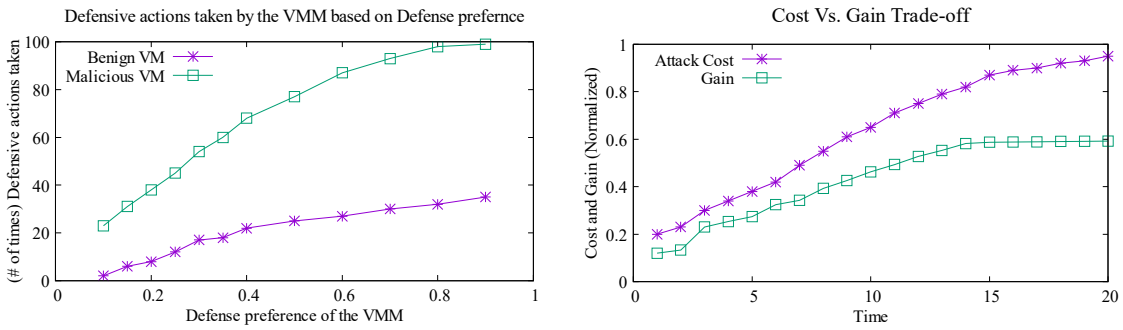
B. Characteristics Analysis:

We observe how the behavior of different type of VMs impact the respective beliefs of the VMs maintained by the VMM in Fig. 6. Intuitively, there is not that much of a change in the belief in the case of a benign VM but this might not be the same in the case for a malicious VM. A benign VM normally does not do any kind of suspicious activity. However, there might be the scenario when a benign VM's behavior might be perceived as greedy, the impact on belief remains almost the same notwithstanding this occasional deviant behavior. A malicious VM might proceed tactfully to impose as a benign VM to the hypervisor and later, act greedily. However, the hypervisor is capable to identify this occasional benign behavior. We observe these scenario in Fig. 6(a). We further observe that though the initial belief about the malicious VM is relatively lower than that of the benign VM, the hypervisor is able to identify the malicious behavior and update its belief. We observe in Fig. 6(b) how the initial belief and the dynamic belief interact.

C. Performance Analysis:

We observe the gain acquired by the malicious VM in both the instances: when the VMM abstains from taking any action and when it takes defensive actions (*i.e.*, *Prime* or *Throttle*). The experimental result is shown in Fig. 6(c). The graph shows that when the CAMP defense mechanism is not deployed, the attacker gains information rapidly. However, the attacker cannot gain enough information when the defense mechanism is in place.

The hypervisor takes *Prime* action as a first line of defense and primes the L3 cache of the suspicious VM, which typically increases the access time for the VM. Further malicious action by the attacker increases the belief towards being a malicious VM and the hypervisor restrict some access to physical resources, which we call *Throttle* action. However, if the *Throttle* action continues, the cache access probability (X_t) will go below the threshold value (X_{Th}) value and the



(a) (b)

Fig. 7. (a) The number of times defensive actions are taken, and (b) the trade-off between cost and gain by the attacker.

VM will be transferred to another server. These observations are presented in Fig. 6(d).

The hypervisor can use different preference level (λ) for defense. The actions taken by the VMM can be heavily affected by this preference level. We observe in Fig. 7(a) how a VMM's actions are influenced by the preference level. We see that as the preference level increases (*i.e*, the hypervisor is more interested to defend), the number of time the VMM takes *abstain* action increases. We also note that, the increased preference level has more impact on the malicious VM than that on the benign VM.

The attacker experiences cost for the attack and as the game progresses, the attacker's information gain increases. However, with the increase in gain, the hypervisor also updates the belief function of that attacker based on the suspicious activity that ultimately increases the belief towards being a malicious VM. The increased belief prompts the hypervisor to take more severe action that inevitably increases the attacking cost but with reduced or no gain. Ultimately, the attacker loses motivation to attacks. The interaction of this three factors are shown in Fig. 7(b).

VIII. Conclusion

It is important to defend against co-resident attacks, because these attacks can reveal crucial information or create DoS. Due to the nature of the co-resident attack, its mitigation is highly challenging. The detection of a malicious VM needs to be beyond a reasonable doubt. Otherwise, a benign user will suffer with an undue consequence or a malicious user may escape the due punishment. Since a VM can be either malicious or benign, taking action according to a conservative approach will not result in a desired outcome. The proposed CAMP game is an appropriate choice as an attacker detection approach. This is because, as our evaluation has proved, the solution to the CAMP game provides optimal strategies to detect and defend the malicious VMs, while keeps the impact on the benign VMs limited. In the future work, we would like to perform real experiments to further evaluate the efficacy of the proposed defense mechanism.

References

[1] T. Ristenpart *et al.* Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on CCS*, pages 199–212, 2009.

[2] D. J. Bernstein. Cache-timing attacks on aes, 2005. Technical report.

[3] Y. Zhang, A. Juels, M. Reiter, and T. Ristenpart. Cross-vm side channels and their use to extract private keys. In *Proceedings of the 2012 ACM conference on CCS*, pages 305–316.

[4] D. Osvik, A. Shamir, and E. Tromer. Cache attacks and countermeasures: the case of aes. In *Cryptographers Track at the RSA Conference*, pages 1–20. Springer, 2006.

[5] D. J. Bernstein, T. Lange, and P. Schwabe. The security impact of a new cryptographic library. In *International Conference on Cryptology and IS in Latin America*, pages 159–176. Springer, 2012.

[6] E. Tromer, D. Osvik, and A. Shamir. Efficient cache attacks on aes, and countermeasures. *Journal of Cryptology*, 23(1):37–71, 2010.

[7] M. S. Inci *et al.* Seriously, get off my cloud! cross-vm rsa key recovery in a public cloud. Technical report, Cryptology ePrint Archive, Report 2015/898, 2015. http://eprint. iacr. org, 2015.

[8] R. Gibbons. Game theory for applied economics. Princeton University Press, 1992.

[9] Y. Han *et al.* A game theoretical approach to defend against co-resident attacks in cloud computing: Preventing co-residence using semi-supervised learning. *IEEE Trans. Inf. Foren. Sec.*, 11(3):556–570, 2016.

[10] Y. Han *et al.* Security games for virtual machine allocation in cloud computing. In *International Conference on Decision and Game Theory for Security*, pages 99–118. Springer, 2013.

[11] H. S. Bedi and S. Shiva. Securing cloud infrastructure against co-resident dos attacks using game theoretic defense mechanisms. In *Proceedings of the ICACCI'12*, pages 463–469. ACM, 2012.

[12] F. Liu *et al.* Catalyst: Defeating last-level cache side channel attacks in cloud computing. In *IEEE HPCA*, pages 406–418, 2016.

[13] J. Shi, X. Song, H. Chen, and B. Zang. Limiting cache-based side-channel in multi-tenant cloud using dynamic page coloring. In *41st IEEE DSN-W*, pages 194–199, 2011.

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

[15] J. Wu *et al.* Xenpump: a new method to mitigate timing channel in cloud computing. In *5th International Conference on CLOUD*, pages 678–685. IEEE, 2012.

[16] J. Szefer, E. Keller, R. Lee, and J. Rexford. Eliminating the hypervisor attack surface for a more secure cloud. In *Proceedings of the 18th ACM conference on CCS*, pages 401–412, 2011.

[17] S. Jin *et al.* Architectural support for secure virtualization under a vulnerable hypervisor. In *Proceedings of the 44th Annual International Symposium on Microarchitecture*, pages 272–283. ACM, 2011.

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

[19] V. Varadarajan, T. Ristenpart, and M Swift. Scheduler-based defenses against cross-vm side-channels. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 687–702, 2014.

[20] S. Sundareswaran and A. Squcciarini. Detecting malicious co-resident virtual machines indulging in load-based attacks. In *International Conference on ICS*, pages 113–124. Springer, 2013.

[21] S. Yu, X. Gui, and J.. Lin. An approach with two-stage mode to detect cache-based side channel attacks. In *The International Conference on Information Networking(ICOIN)*, pages 186–191. IEEE, 2013.

[22] M. Hasan and M. Rahman. A game-theoretic approach to defend co-resident attacks in cloud computing. https://dl.dropboxusercontent.com/u/35018736/Coresident_Attack_Mitigation_TR.pdf. Technical Report.