Journal of Information Security and Applications 000 (2020) 102397

Contents lists available at ScienceDirect

Journal of Information Security and Applications

journal homepage: www.elsevier.com/locate/jisa

A signaling game approach to mitigate co-resident attacks in an IaaS cloud environment

M.G.M. Mehedi Hasan[a,*], Mohammad Ashiqur Rahman[b,*]

[a] Department of Computer Science, Tennessee Tech University, Cookeville, USA
[b] Department of Electrical and Computer Engineering, Florida International University, Miami, USA

ARTICLE INFO

Article history:
Available online xxx

Keywords:
Co-resident attacks
Collaborative attacks
Signaling game
Nash equilibrium
Cloud security

ABSTRACT

Cloud service providers (CSPs) offer a variety of services that are opening the door to the infinite possibilities of cloud computing. Despite numerous benefits offered by the CSPs, there are, however, some security issues that may dissuade users. In cloud computing, different virtual machines (VMs) often share the same physical resources, which 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 in which a VM attacks another co-resident VM through a covert side channel that can be used to extract another user's secret information or launch denial of service attacks. The attack campaign becomes more damaging when multiple co-resident VMs collaborate. In this paper, we analyze the co-resident attacks and corresponding defense strategies, with respect to benign and malicious VMs and the defender, i.e., the VM monitor (VMM), using a signaling game model. The solutions to the game provide optimal defense strategies for the VMM with respect to the expected number of malicious VMs in collaboration. We evaluate the game results through simulations on various synthetic attack scenarios. The results show that the defender can effectively resist co-resident attacks by distinguishing the benign and malicious VMs.

1. Introduction

Cloud computing opens the door to co-resident, cross-side channel attacks between virtual machines (VMs) in a public cloud when the VMs share the same hypervisor, CPU, memory, storage, and network devices. Some of the resources can be partitioned (e.g., CPU cycles, memory capacity, and I/O bandwidth). However, VMs also share resources that cannot be well partitioned, such as last-level cache (LLC), memory bandwidth, and IO buffers. The shared resources can be exploited by attackers when launching cross-side channel attacks to extract other users' secret information or when launching a denial of service (DoS) attack.

The number of people drawn to use cloud computing services is increasing rapidly because of the varieties of services offered by the cloud service providers (CSPs) coupled with the flexibility of use and cost effectiveness. CSPs provide cloud services in different forms. Infrastructure as a service (IaaS) is one of them, which offers each cloud service user a VM that emulates a physical machine, thereby providing the user the greater flexibility an actual machine would yield. Several VMs sharing the same physical resources, like CPU, memory, and storage devices, are called co-resident VMs, which can lead to new kinds of attacks known as co-resident attacks. In these attacks, malicious users build various types of side channels [1,2] 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), which is also known as the L3 cache. The L3 cache is shared between all the residing VMs, which opens the door for the attacker to launch cache-timing attacks. Ristenpart et al. were the first to discover this kind of attack [1] that exploited cache timing. They discussed how the concept of covert channel could be extended to launch attacks in clouds. They used the idea of Bernstein [3] show that it is possible to extricate AES key using cache-timing attacks. Hence, various types of secret information including cryptographic keys can be leaked to malicious users through these side channels. Since co-resident attacks are carried out using covert channels and they leave few traces in the system logs, it is harder to detect them.

* Corresponding authors.
E-mail addresses: mmehediha42@students.tntech.edu (M.G.M. Mehedi Hasan), marahman@fiu.edu (M.A. Rahman).

https://doi.org/10.1016/j.jisa.2019.102397
2214-2126/

There are several techniques that a malicious VM uses to launch co-resident attacks. One such technique is the PRIME+PROBE technique, which is based on cache-timing [3–5]. In this case, an attacker shares the physical resources with its target victim through a VM. The attacker VM observes the cache activity of his victim VM by first priming the cache memory of the victim VM, then remaining in *busy-wait* state for a certain amount of time. During this time, the attacker waits for the victim to use the cache. After that, the attacker VM primes the cache again and if the attacker's prime results in a cache *hit*, that particular cache line was not used by the victim process. However, if the prime results in a cache *miss*, the cache line was used by the victim process, which evicted the attacker's data, thus resulting in more time to acquire the data. This cache-timing of loading the data reveals what kind of activity is executing on the victim VM. The pattern of activities makes it possible to extricate cryptographic keys. The detail technique of this attack type is discussed in the appendix of the technical report [6]. There is another co-resident attack technique known as the FLUSH+RELOAD technique [2,7,8], which requires the attacker to reside in the same core. The attacker first flushes the cache of the victim and then remains in *busy-wait* state for a certain amount of time for the victim to run its process. Now, if the attacker reloads his data and observes the timing, he can find that some data result in longer loading times. The attacker can perceive that this cache line was used by the victim process, which can be leveraged to extract the secret keys.

Co-resident attacks are normally carried out in two steps. In the first step, the attacker VM attempts to co-reside with the target VM. After achieving the co-residence, the attacker VM launches the side channel attacks as a second step. Since the co-resident attacks make the benign VMs suffer, it is important for the VM monitor (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. If proper caution is not taken while taking defensive action, a benign VM might get punished through a wrong decision. Therefore, it is crucial to identify potential attackers with a certain level of confidence. An attacker can strive to deploy several co-resident VMs forming a collaborative attack team to carry out the attack process. This coordinated attack is highly appealing for an attacker as the use of multiple attacking VMs, instead of one single VM, increases the chance of realizing the attack goal.

Contributions. The detection of the attacker VMs and the application of appropriate countermeasures should be done with proper discretion. We address this need in this research by modeling the co-resident attack and defense problem as a game, which we named as the co-resident attack 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. We model the CAMP game as a signaling game [9]. Since the signaling game is suitable for distinguishing between different types of senders (i.e., the benign and malicious VMs) with respect to the defender (i.e., the VMM), using a belief model, it will be a perfect mechanism for defending against co-resident attacks. The proposed game assumes an expected capability for the attacker with respect to the number of VMs employed for launching the co-resident attack. The defense mechanism includes a set of security measures among which the VMM selects its strategy against the attacker VMs. In this work, we consider a cloud environment where the same VM images provide the same services i. e., we consider "standard cloud servers" instead of "clustered cloud servers."

According to this game framework, we analyze the interactions between the VMs and the VMM and obtain both the pooling and separating equilibria [10]. We evaluate the equilibrium strategies, especially with respect to the defender, by developing a simulation program and running synthetic co-resident attack scenarios. Our game model defends against any number of attacker VMs, where an attacker can deploy a single VM or multiple (collaborative) VMs. The results show that the optimal defense strategies provided by the CAMP game can foil the attacker's effort by successfully detecting the malicious VMs. A preliminary outcome of this work, especially when the attacker deploys only a single VM to carry out the attack, was published in [11].

Organization. The rest of this paper is organized as follows. In Section 2, related work on co-resident attacks is presented. Section 3 presents the strategy of our game model. We present the game model in Section 4. We analyze the game result in Section 5. The following section presents the evaluation of the game results. We further explain several intriguing points on this research in Section 7. We conclude the paper in Section 8.

2. Related work

Co-resident attacks have been studied in different research works. Han et al. explored how malicious users aim to co-locate their virtual machines (VMs) with the target VMs on the same physical server to extract private information from the victim using side channels [12]. The authors proposed how the use of game theory can help reduce these attacks through efficient VM placement, which reduces the chance of co-residency of the attacker VM with the victim VM [13]. Jensen et el. talked about malware injection attacks in [14]. In this kind of attack, a malicious attacker attempts to inject malicious services or virtual machines into the cloud. If the attacker can successfully launch malicious services, the CSP can face serious consequences. Zhang et al. showed how side channel attacks can be done by placing a malicious VM on a target cloud server [2]. If the attackers become successful, they can extract the private keys of the target VM.

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 [15]. 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) [16]. Shi et al. designed a dynamic page coloring solution to limit cache side channel attacks [17]. Vattikonda et al. [18] and Wu et al. [19] worked on how the high resolution clocks, that many side channels rely on, can be modified or removed. Jin et al. [20] and Szefer et al. [21] worked on redesigning the architecture of cloud computing systems. Zhang et al. proposed how the side channel can be made noisy to defeat the attacker's effort to extract information [22]. Luo et al. pointed out how the lack of security border and isolation can allow the possibility of information leakage [23]. They proposed a divide and conquer strategy, where cloud computing virtualization problems are classified and analyzed, respectively. Based on this classification, they provided the corresponding response measures.

Manshaei et al. did an extensive survey on how game theory can be utilized to meet the network security issues [24]. That survey covered various research papers dealing with a variety of security concerns. Han et al. applied game theory in virtual machine allocation policy to prevent co-residence in the cloud [12]. Maghrabi et al. used game theory to assess the risk involved in moving critical assets of an IT system to a public cloud [25]. Qiu et al. worked on secure virtual machine deployment strategy to reduce co-residency in the cloud [26]. Wang et al. added reputation deriving from the social cloud as part of the utility and proposed the interaction between two rational parties in the social cloud as a game where two parties receive their opponent's trust or reputation from the social cloud [27]. They showed that every party has the motivation to cooperate and increase their

reputation. Jebalia et al. focused resource allocation in storage and securing stored data from malicious cloud users [28]. They resorted to game theory and used a sequential revocation game where cloud users can cooperate to revoke malicious ones. Obfuscation is popular among malware and virus developers who use it to conceal the operation of their code while executing their activities in an uncontrolled environment. This technique is also used by game developers and industries who need to protect their intellectual property. Hataba et al. proposed that this technique can also be used for cloud security [29]. Levtin et al. proposed optimal data partitioning in the cloud with random server assignment to prevent co-residency [30]. Zhuang et al. appllied signaling game based on the three possible types of defender signals in a multiple-period attacker-defender resource-allocation to model strategies of secrecy and deception [31]. Çeker et al. applied signaling game with perfect Bayesian equilibrium (PBE) to model the way deception affects the attack–defense interactions to disguise a real system as a honeypot (or vice versa) in an attempt to mitigate Denial of Service (DoS) attacks [32].

Kamhoua et. al proposed a game model that considers a set of users who share the same hypervisor in a public cloud [33]. The goal of the game model is to provide incentives to all cloud participants to invest in vulnerability discovery and share their cyber-threat information despite the potential cost involved. Kwiat et al. proposed a cloud security game [34]. They considered four players, an attacker and three users, acting across two hypervisors. The strategy to launch an attack may include steps such as: collecting information, credential compromising, executing attack payload, establishing a backdoor, and scanning. Using numerical analysis they showed that one pure strategy and multiple mixed strategies are possible. The CSPs are supposed to comply with the Service Level Agreement (SLA), but there is an incentive for the CSPs not to provide the mentioned resources to the users. As a solution to this problem, Zhou et al. proposed a competitive mechanism based on game theory, which will motivate semi-honest CSP for the credible implementation of SLA [35].

Varadarajan et al. proposed a scheduling mechanism called minimum run time (MRT) that can prevent cache-based side channel attacks [36]. Their proposed method requires fewer changes to existing cloud platforms and hence are easier to deploy. Sundareswaran and Squcciarini worked on identifying abnormalities in CPU and RAM utilization, system calls, and cache miss activity [37]. Yu et al. also worked to identify similar kinds of activities [38]. In their approach, when malicious users adopt the PRIME+PROBE technique to obtain information from the victim, certain kinds of abnormalities arise.

Although the researchers have proposed several techniques to mitigate the co-resident attacks by preventing or limiting the chance of being co-resident, limited research has been performed to analyze the interactions between the malicious VMs and the VMM so that optimal defense strategies can be identified during the second step of the co-resident attack (i.e., the side channel attack part). The game theory is widely proven to be useful in resolving strategically conflicting behaviors [10]. 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.

3. CAMP game: strategy model

Infrastructure as a Service (IaaS) is one of the popular and prominent services offered by CSPs. Several VMs are hosted on a single server for this service. A typical VM placement on a server in the cloud is shown in Fig. 1. 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 the 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 sender i.e., whether malicious or benign. We show the illustration of the CAMP game model in Fig. 2.

We assume that an attacker VM tries to extract important information like private key from a victim VM. We use the terms attacker VM and malicious VM interchangeably. Without the loss of generality, we use the third person masculine gender to represent the attacker when it is needed. Here, the VMM wants to defend the benign VMs and prevent the attacker from gaining any information. The VMM (as a defender) needs to deal with the benign senders and the 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, while the target can consider all the senders as its clients. The malicious VM often accesses the physical resources right after the access of the target VM so that it can extract useful data about the VM from the L3 Cache. If any VM shows this malicious behavior frequently or consistently with respect to a certain VM, this VM is considered malicious VM.

3.1. Actions of the players

Every player has some predefined actions. The actions of the players are also known as 'move' of the player. Based on the actions, each player develops his strategies, which are also know as action profiles of those players. In this section, we define different actions of the players participating in the CAMP game.

3.1.1. 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. The reason behind this assumption is that we consider the victim changes his secret information after a time interval of T_M. The value of T_M may vary depending on the VMs. A malicious VM can start the attacking process at any time during a particular T_M. 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. For instance, if T_M is 7 s, T_A should be such that the attacker will have enough time to exploit the information gain. We show this scenario in Fig. 3, where T_A is 5 s. Though there is variable communication latency, we assume that the sending time is the same as the receiving time.

In the CAMP game, the attacker determines the amount of information he gains by his access. 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 them. On the 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 no intention of attack, there is still a probability that its action might seem to be greedy. We define a simple threshold-based mechanism to distinguish these two strategies of the sender, either malicious or benign. In this respect, we define G as an average of the total potential information gain that is required to retrieve the secret information of the target VM.

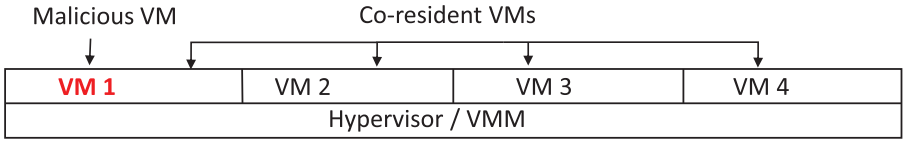

Fig. 1. Typical VM placement on a server in the cloud environment.

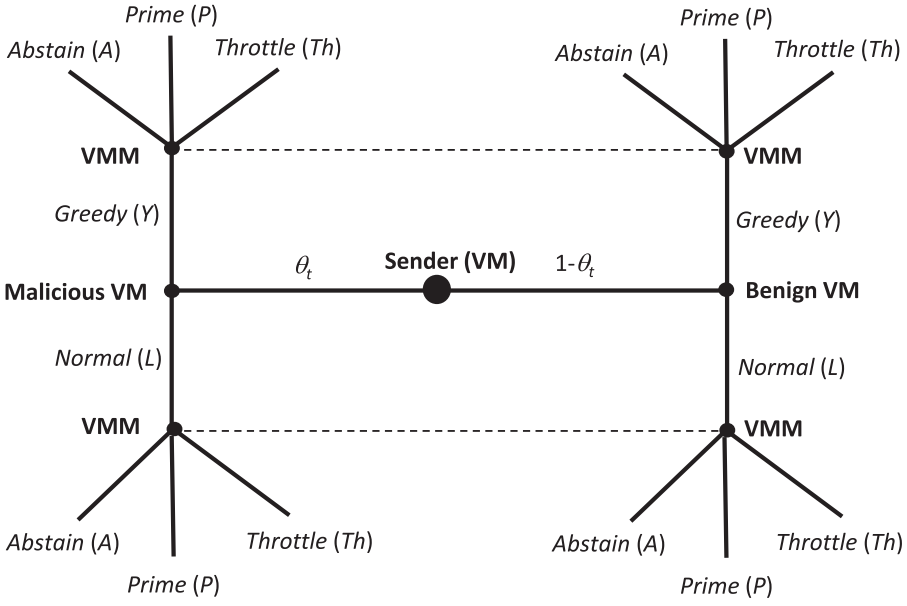

Fig. 2. Co-resident attack defense model. A VM (sender) tries to access the physical resources and the VMM (hypervisor) provides services to several VMs. The VMM can take either of the three actions (*Abstain* (*A*), *Prime* (*P*), or *Throttle* (*Th*)).

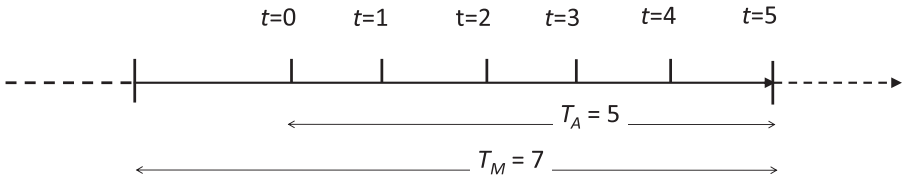

Fig. 3. An example showing the relation between T_M, T_A and t, when the maximum time for information gain is $T_M = 7$ s and the time of the game between the attacker and the hypervisor is $T_A = 5$ s.

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. This is Obvious, according to the expected behavior, $G^B \leq G \leq G^A$. Table 1 summarizes the notation used throughout this paper.

If there are total of N VMs on 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.

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

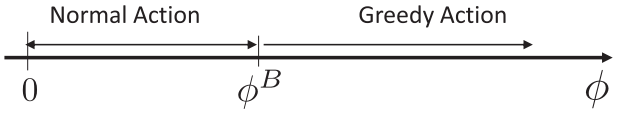

Fig. 4. Strategy of the sender. The sender is playing *Greedy* if it asks for more information than ϕ^B. Otherwise, it plays *Normal*.

3.1.2. Hypervisor's actions

The hypervisor takes three kinds of actions based on the belief (see Section 4.3). When the hypervisor does not take any kind of defensive action, we call this an *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 and incurs additional unit time cost. The actions taken by the hypervisor depend on a belief, which is based on the behavior of the VMs. The corresponding belief is updated based on the behavior of the corresponding VM. As a matter of fact, some benign VMs might show greedy behavior, which should not be enough to judge the VM as the ultimate malicious VM. In such a case, when a benign VM is indeed behaving greedily, a mild punishment will alert the VM to show benign behavior. The same idea is not be

Table 1
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
$\bar{\theta}_{j,0}$	Initial belief about VM j
$\hat{\theta}_{j,t}$	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)
$\bar{C}_j$	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

true for a malicious VM. This is because if a VM is indeed a malicious one, it will ultimately behave greedily to achieve its attack objective.

4. CAMP game model

An attacker can deploy one or more than one VMs to launch co-resident attacks. When the attacker deploys only one VM for his malicious purpose, the chance of getting caught by the defender is relatively high. A defender can easily identify the malicious VM using our CAMP game model. In this case, the attacker might deploy several VMs instead of one VM. Each of the VMs will carry out the fair share of the "assigned tasks". In this case, the "assigned task" means gaining the private key of the victim VM. We call these kinds of attacks "collaborative co-resident attacks" and the VMs responsible for launching the collaborative attacks are called "attacker VMs". All of the attacker VMs deployed by the attacker form a team, which we call the "attack team". Each of the members of the attack team will work for the attacker and contribute toward gaining the private information/secret key of the victim VM.

4.1. Needs for collaborative attacks

The hypervisor keeps tabs on every VM in the cloud server. The belief of every VM is updated based on its activities. When a VM behaves greedily, the corresponding belief of the VM reflects its greedy behavior. If a particular VM continues its greedy behavior for a certain amount of time or does it regularly, then it is easy for the hypervisor to identify it as a potential attacker VM. In our CAMP game model, we assume that the attacker can deploy one or more than one VMs. When the attacker has only one malicious VM, the defender's job is to identify that particular VM. However, the task of identifying the attacker VM will get complicated if the attacker deploys his attack team, i.e., he has more than one attacker VMs. In this way, no particular VM is burdened with all the work of attacking the victim VM, where attacking tasks will be distributed among all the members of the attack team.

The greater the number of attacker VMs in the attack team, the harder it is for the defender to identify the attacker. The attacker will launch one of the VMs from its attack team. Let $\mathbb{M}$ denote the set of attack team, where each of the members of the team is a malicious VM (attacker VM) and the set has two members, i.e.:

$$\mathbb{M} = \{A_1, A_2\}$$

Here, A_1 and A_2 are the attacker VMs.

If in the first step, the attacker VM A_1 attacks the victim VM then in the next step the other attacker VM A_2 will launch the attack. So, when the A_1 VM behaves greedily, the belief of that particular VM is updated. Similarly, when the A_2 VM behaves greedily, its corresponding belief is updated. In this case, the greedy behavior of one VM does not affect the belief of the other VM. Because the defender treats every VM as a distinct entity, the behavior of one VM does not affect other VM. This property gives the attacker a new horizon of opportunities. The attacker will exploit this property to launch several VMs and deploy them as needed. If there are several VMs involved in attacking a particular VM, the defender's task to identify potential malicious VMs becomes harder and the attacker might get away with his malicious activities because the beliefs of those attacker VMs might be no more unusual than those of other benign VMs.

4.2. Collaborative attacks: team size

The attacker needs to select the team size carefully. If the total number of VMs on a cloud server is N, then the upper bound of the team size is N and the attacker might opt not to have any VM, which gives the lower bound of the team size as 0. However, it is very difficult to have N number of VMs in the team. The challenge here for the attacker is to select an optimal team size that would maximize the gain within the stipulated game time. If the team is large, the chance of accessing the physical resources by any of the team members right after the victim has accessed the resources becomes highly likely. The constraints on team size is as follows:

$$0 \leq |\mathbb{M}| \leq N\,.$$

There are several constraints that the attacker needs to consider while deciding the team size. Most importantly, some of the constraints are not within the attacker's capability (see Section 4.2.1). The attacker has to put a lot of effort in achieving co-residence with the victim VM even when he is using a single attacker VM for this purpose. The effort increases significantly when he tries to achieve co-residence with victim VM using multiple attacker VMs.

4.2.1. Difficulty of achieving co-residence

The first phase of launching co-resident attacks is to co-locate/co-reside with the victim VM. However, achieving co-residence is not easy. Even though an attacker spends a lot of time and money for this purpose, he may still not be successful.

The attacker has to spend valuable resources to identify the likely residence of the victim VM. The time constraint that the attacker needs to overcome can most of the time outweigh the benefits gained from co-resident attacks. It might be the case in many times that by the time the attacker is able to find the likely residence of the target VM, the need for launching attacks becomes futile. For example, if the attacker plans to attack a member with conflict of interests (COIs), he needs to know what kind of product the victim is planning to market in the near future. Based on this information, the attacker might devise a new product that can beat the victim's. Now, if the attacker fails to get the information on time, i.e., he gets the information after the product is marketed by the victim, then the attacker has no gain from launching co-resident attacks even though he has already spent valuable resources to achieve co-residence. This indicates that achieving co-residence long before the due time is very important. An attacker has to launch several VMs in order to co-locate the victim/target VM and has to bear the financial cost for launching each of the VMs. Launching a lot of VMs might not guarantee success if the cloud service provider deploys schemes that make achieving co-residence harder [13,39–41].

4.2.2. Difficulty of managing multiple VMs

The cloud user needs to spend money for enjoying the cloud services. Different CSPs deploy different schemes to charge their cloud users [42]. Some CSPs charge hourly, while others serve on contract basis [43]. So, if an attacker wants to deploy VMs to launch co-resident attacks, he needs to spend money for the service, which implies that managing multiple VMs will increase the cost. There is no way out for the attacker to get away with the cost, as every VM user is bound to follow service level agreements [44–46]. The amount of financial cost that the attacker has to spend might dissuade him from launching co-resident attacks.

4.3. Belief model

Each member of the attack team keeps track of the victim VM. The attacker distributes the tasks among all the attacker VMs. The attacker VMs, in turn, carry out the assigned tasks. The assigned tasks include everything that an attacker needs to do to launch successful attacks, which range from conducting reconnaissance on victim VM to launching the final attacks. We consider every team as an entity just like a normal VM (benign or malicious). The defender observes the behavior of every team just like it does for other VMs. As a result, the hypervisor will maintain a belief for every team and it will make a decision based on this belief. The belief will consist of initial belief and dynamic belief. The hypervisor will work as the dedicated agent of the CSP, whose responsibility is to protect the benign users.

The hypervisor maintains a belief θ_t, which is belief at attempt t, about each of the VMs. The value of θ_t is comprised 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 [10]:

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

Here, the number e is a well-known mathematical constant (Euler's number), approximately equal to 2.71828, $\bar{\theta}_{\Gamma_u,0}$ is the belief about the attack team Γ_u at time $t = 0$, $\bar{\theta}_{\Gamma_u,t}$ is the belief about the attack team Γ_u at time $t = t$, and u indicates the serial number of the team. In fact, $\bar{\theta}_{\Gamma_u,0}$ and $\bar{\theta}_{\Gamma_u,t}$ specify the initial belief and the dynamic belief, respectively. We observe from Eq. (1) that the larger $\theta_{\Gamma_u,t}$ is, the higher is the belief that the VM is malicious.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 $\theta_{\Gamma_u,t}$ toward greedy behavior will have a larger impact on the belief. The $\bar{\theta}_{\Gamma_u,t}$ part is dynamic and changes over time depending on the behavior of the VMs.

4.3.1. Initial belief function

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

4.3.2. 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 is a requirement of n VMs from the same user (the requester of VM j), then:

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

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 fact. When the number of VMs requested by a user is small, the impact is low, but as the number of requested VMs by a single user increases, the impact rises significantly.

4.3.3. Similar VM requirements

If a certain number of VMs has the same type of requirements, then this indicates a suspicious activity and those VMs might be considered malicious. We measure the suspicious activities of VM j using S_j, which is computed as follows:

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

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

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

The hypervisor will maintain a normalized value for the initial belief. We denote the initial belief for team Γ_u as $\bar{\theta}_{\Gamma_u,0}$, where u indicates the serial number of the team. The value of $\bar{\theta}_{\Gamma_u,0}$ is calculated as follows:

$$\bar{\theta}_{\Gamma_u,0} = \min\left\{1, \forall VM_j \in \Gamma_u \sum (R_j + S_j)/(2 \times |\Gamma_u|)\right\}. \quad (2)$$

4.3.4. 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 an SLA between the CSP and the VM user. Both the parties are bound to comply by 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 the L3 Cache. The value of the correlation coefficient between the two VMs' access times gives us an idea whether a malicious VM's access to the physical resources has anything to do with the access time of the target VM. Let $Access(VM_i, t)$ return 1, if VM_i accesses the physical resources at attempt t and return 0 if it does not access the resources. 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, the behavior of the VM j is suspicious. But to have a strong correlation, the access sequence has to be as immediate as possible. The more distant the access sequences are, the less the correlation in terms of access sequence between the two VMs.

We define the correlation coefficient, $r_{(i,j),t}$ at attempt t in the following way:

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

Here, n_t indicates the number of tries at attempt t. The higher value of $r_{(i,j),t}$ indicates that the behavior of VM j is suspicious toward VM i.

If the attacker can deploy multiple VMs co-resident with the target VM, the attack can be done collectively. If the attack team includes two members, i.e., VM_j and VM_p, and the target VM is VM_i, the correlation coefficient, which is denoted by $r_{(i,j,p),t}$ for VM_j and VM_p against the victim VM_i at attempt t, can be represented in the following way:

$$r_{(i,j,p),t} = \frac{1}{t}\sum_{k=1}^{k=t} \Big(Access(VM_i, k) \wedge (Access(VM_j, k+1) \vee Access(VM_p, k+1))\Big)\,. \quad (3)$$

If the attacker can deploy up to three VMs for his attacking purpose, the attack team can contain up to three VMs. Let VM_j, VM_q, and VM_p are the members of the attack team. The correlation coefficient, which is denoted by $r_{(i,j,p,q),t}$ for VM_j, VM_p, and VM_q with respect to the victim VM_i at attempt t, can be represented in the following way:

$$r_{(i,j,p,q),t} = \sum_{k=1}^{k=t} \Big(Access(VM_i, k) \wedge (Access(VM_j, k+1) \vee Access(VM_p, k+1) \vee Access(VM_q, k+1))\Big)/t\,. \quad (4)$$

We can generalize Eqs. (3) and (4) for a team Γ_u in the following way:

$$r_{(i,\Gamma_u),t} = \frac{1}{t}\sum_{k=1}^{k=t} \Big(Access(VM_i, k) \wedge \exists VM_j \in \Gamma_u \wedge \bigvee_{j=1}^{j=|\Gamma_u|} (Access(VM_j, k+1))\Big)\,. \quad (5)$$

The dynamic belief of team Γ_u at attempt t is denoted as $\bar{\theta}_{\Gamma_u,t}$. We calculate the dynamic belief as follows:

$$\bar{\theta}_{\Gamma_u,t} = \min\left\{1, r_{(i,\Gamma_u),t}\right\}. \quad (6)$$

We can find the total belief ($\theta_{\Gamma_u,t}$) from Eq. (1) using Eq. (2) and (6). 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 the attack team Γ_u at time t.

4.4. Payoff model

The purpose of a collaborative attack is to make the identification of the attacker difficult. When there are several malicious VMs, the defender needs to keep tab of every VM. The defender does not know beforehand how many attacker VMs are on the server. If the hypervisor tries to defend its benign VM considering only one malicious VM, but the attacker actually launches attacks using more than one VM, then it gives the attacker the leeway to continue attacks even though the hypervisor's defense mechanism is in place.

4.4.1. Gain function

The purpose of both the attacker VM and the VMM remains the same i.e., to maximize their payoffs. When the attacker launches collaborative attacks, the chances of gain become higher. However, higher gain also invites higher cost. This time, the attacker needs to maintain several VMs, which incurs relatively higher maintenance costs. Besides, the attacker needs to make the moves with more caution because, in collaborative attacks, the whole team (i.e., Γ_u) will have to bear the consequences. The attacker needs to tread carefully to balance the cost and gain. The hypervisor tries to utilize this fact to foil collaborative attacks.

As we have already mentioned, the large portion of the attacker's success is dependent on how frequently he can access the cache right after his target VM. Now, the attacker can rely on several VMs instead of just one, the attacker can deploy any of the VMs from the attack team Γ_u for this purpose. The attacker can randomly pick any of the attacker VMs and follow the victim VM beguiling the defender into thinking that those attacker VMs are actually benign VMs. The set $\mathbb{S}$ constitutes the successful accesses of the cache right after his target by any of the attacker VMs. The constraint for successful attack (i.e., $|\mathbb{S}| \geq n_s$) will now be fulfilled by the members of the attack team.

We assume that one attack VM is randomly selected by the attacker from the attack team. As long as the attacker is able to access the physical resources successfully by any of the attacker VMs, this will count toward his required successful access (i.e., n_s). Now, the modified gain function (G_t) for collaborative attacks can be represented as follows:

$$G_t = G_{t-1} + \frac{X_t \times (1 + \frac{|\Gamma_u|-1}{N-1})}{n_s}\,. \quad (7)$$

We may recall, X_t represents the attacker's probability of success at attempt t.

If the VMM takes an *Abstain* action at attempt t, there will be no change in X_t (i.e., $X_t = X_{t-1}$). When the VMM takes the *Prime* action, all the VMs of the attack team Γ_u experience slower data processing times and the definition of X_t is as follows:

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

We may recall that X_0 is the cache access probability before the game begins, and α is defined as follows:

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

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

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 for a longer time (t) on 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 the *Prime* action until the condition for *Throttle* action is met. We showed this scenario in Fig. 5.

4.4.2. Cost function

The cost of the attacker becomes manifold in the case of collaborative attacks. The costs of launching and maintaining VMs increase with the increase of members in the attack team. There are two ways an attacker can subscribe to a CSP. In one way, he can subscribe for a short period of time, which incurs a relatively low cost. However, this approach has its downside as well. In this case, he will have to complete his attacking process within that short period of time, which requires an aggressive action. This

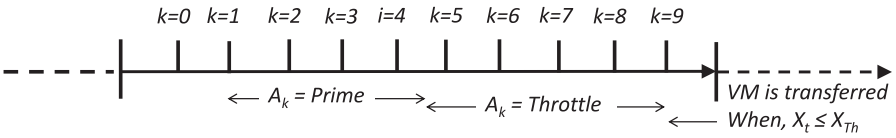

Fig. 5. An example showing how the VMM changes its course of actions. Once an action is selected, it will be taken again as long as the conditions for taking higher action are not met.

aggressive action, in turn, makes it suspicious to the hypervisor. The hypervisor will be quick to run a background check (belief) and take action. The attacker can also take a slow and steady approach that means he will subscribe for a longer period of time. This will give the attacker enough time to study his victim and launch attacks. But a longer subscription period also means a relatively large amount of money and longer waiting time. While this approach might do well in terms of hiding his malicious intents, the attacker's whole efforts might go in vain if the hypervisor, for some reason, decides to migrate either of them. The attacker will then have to start the whole process of achieving co-residence to monitoring the victim for launching the attacks. Here, the attacker wants $|\Gamma_u|$ VMs to co-reside with his target and the attacker needs to launch $f(\Gamma_u)$ VMs. If C_p is the initial cost per VM and C_{fmax} is the maximum cost the attacker is willing to bear, then the cost function for the financial part (ν_0) for VM j can be defined in the following way:

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

The attacker incurs cost of his own when he attacks. We may recall that τ_t be the cost incurred by the attacker at attempt t. The cost of an attack is very trivial when compared to the reconnaissance cost. Since all the information about the victim is known, i.e., the attacker is already co-resident with the victim, the attacker only cares about the number of times he attacks for the information gain. We define τ_t as a simple linear concave function:

$$\tau_t = \upsilon_0 + \sum_{1\le k<t} H_k(x_{k+1} - x_k) + s(\Gamma_u)\,. \tag{8}$$

Here, $s(\Gamma_u)$ is the selection overhead that indicates the attacker's effort in selecting a random attack VM from the attack team, and H_k is the per-unit time cost at the try k. When the *Prime* action is chosen, all the VMs of the attack team get punished and the attacker has to spend more time to get his job done. This action reduces the attack window (T_A) and increases H_k.

The *Throttle* action will have a more severe impact than the *Prime*, especially since, this time, the attacker has to bear the cost for all the VMs in his attack team. The attacker VMs suffer physical resource constraints, which severely hamper their usual activity. This might prompt the attacker to suspend its activity for some time, and at the same time, its hourly cost also increases. Now, if H_k and H_{k-1} are the per-unit time costs at the tries k and $k-1$, respectively, then the increase in unit time cost (ΔH_k) is defined as follows:

$$H_k = \exists VM_j \in \Gamma_u \max\{H_{k-1}\} + \Delta H_k\,.$$

The value of ΔH_i depends on the discretion of the CSP. In this model, we consider the following values of ΔH_i for different actions:

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

The attacker has to consider 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 must start from the beginning. The transfer step is taken as part of the *Throttle* action and occurs 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 restart its whole activity to gain the required information.

The hypervisor needs to take actions cautiously; otherwise, a benign VM might suffer, or an attacker VM might get away with its malicious activity. When any action is taken against a benign VM, the hypervisor incurs a cost. However, if that action is taken against a malicious VM, the cost is disregarded. We define the cost of the hypervisor at attempt t, denoted by ψ_t, in the following way:

$$\psi_t = \sum_{1\le k\le t} \rho_k(A_k) = \psi_{t-1} + \rho_t(A_t)\,. \tag{9}$$

Here, $\rho_k(A_k)$ 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:

$$\rho_k(A_k = Prime) = \alpha X_0\,. \tag{10}$$

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:

$$\rho_t(A_t = Throttle) = \beta H_0\,. \tag{11}$$

The costs defined in Eqs. (10) and (11) 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.

4.4.3. Payoff function

We model the attacker VM's payoff by U_A as follows:

$$U_A = G_t - \tau_t\,. \tag{12}$$

Here, G_t and τ_t have been defined in Eqs. (7) and (8), respectively and both τ_t and G_t are normalized values between 0 and 1. All the terms in Eq. (12) are normalized values between 0 and 1. We model the hypervisor's payoff by U_D as shown in the below:

$$U_D = \lambda(-G_t + \tau_t) - (1-\lambda)\psi_t\,. \tag{13}$$

Here, ψ_t takes any of the forms defined in Eqs. (10) and (11) based on *Prime* action and *Throttle* action, respectively. Here, λ indicates the VMM's preference for taking defensive action.

5. 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 the 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 [10]. The solution of the CAMP provides the following two theorems. The detailed solution will be found in [6].

Theorem 1. *[(Greedy, Normal), (Prime, Abstain)] 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. $\beta \geq 0$.

When an attacker spends aggressively to achieve co-residence and launch attacks, the chance to be caught by the defender also increases. The value of n_s is a fixed value for a particular attack. If, in certain circumstances, the value of n_s is 1, then according to the above constraints, the hypervisor is better off transferring the VM. The VMM takes a top down approach, i.e., the condition is first checked for *Throttle* action. if the necessary conditions are met, the defender takes *Throttle* action and so on. Theorem 1 shows that at the separating equilibrium, the VMM defends (i.e., plays either *Prime* or *Throttle*); if the sender VM plays *Greedy*, the VM is expected to be malicious. It plays *Abstain* if the sender plays *Normal*, and the VM is expected to be benign. The conditions at Theorem 1 indicate 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) believes that the *Greedy* action is taken only by the attacker (a malicious VM), it has no advantage in taking the *Prime* option, a less damaging action against the attacker. However, since a benign sender can seldom show greedy behavior, 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 the above-mentioned 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 the necessary conditions. Here, the VMM (defender) believes that the sender plays *Greedy* for each given type and the expected behavior of the VMM is defending (i.e., plays either *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.

Table 2
Parameter values used in simulations.

Parameter	ϕ^B	N	δ	T_M	T_A	λ	α
Value	0.33	10	0.03	7	5	0.5	0.16

A VM's increase in access duration and the defender's increase in defense preference both impact the action of the attacker. If a VM accesses more frequently, the belief of that VM will move closer to the threshold value. When a defender wants to be more conservative, it will increase the defense preference level, which is denoted by λ.

The hypervisor analyzes the behavior of the VMs and updates the belief. 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 is reduced beyond a threshold, the VMM will transfer the VM to a new physical host.

6. Evaluation of the game results

We evaluate the CAMP game results, particularly the optimal defense strategies, through conducting simulations on CloudSim, a framework for simulating cloud computing infrastructures and services [47]. It is worth mentioning that we do not simulate how the attacker achieves co-residence with its target VM.

6.1. Methodology

We extend or customize the existing CloudSim classes for this simulation. The CloudSim framework is structured in three main layers. The bottom layer is the simulation engine, that supports the core functionalities of the simulation, such as the queuing and processing of events, the creation of software entities and the clock simulation management. The CloudSim layer, that stands in the middle, provides support for the modeling and the management of cloud datacenters, allowing the simulation of VM provisioning, application execution management and providers monitoring. The User Code layer is the top-most one and provides support for high-level modeling of application workloads and cloud resources. Moreover, it provides tenants for implementing scheduling policies through the broker entities. For the purpose of modeling the CAMP mechanism, we modified the VM class to accommodate the necessary VM properties required to launch co-resident attacks.

The experimental values for the different parameters are listed in Table 2. We chose the experimental values for the different parameters arbitrarily but not without some caveat. The parameter ϕ^B represents the expected behavior from a benign VM, which is a normalized value, and this value cannot be greater than 1. The parameter N represents number of VMs on a single server. We did not consider very large value of N to keep our experiment tractable. The parameter δ is the system defined tolerance level, which is a normalized value and must be smaller than ϕ^B. The parameter T_M is the time interval for changing the secret key and T_A is the time during which attacking process continues. In a practical scenario, a successful attack requires T_A to be less than T_M as we considered in our experiment. The parameter λ indicates the VMM's preference for taking defensive action, which is also a

normalized value. The lower value of λ indicates a lenient VMM, whereas, a higher value indicates a more conservative (defensive-minded) VMM. In our experiment, we chose values between these two grounds. The parameter α represents average probability per access, which depends on other parameter settings, especially on N. We create 10 VMs and 1 VM is randomly selected as the victim VM. We then look for potential malicious VMs. The selection of the malicious VM is done randomly. Once the malicious VM is selected, it will show the characteristics of a malicious VM. Instead of selecting a malicious VM randomly, we can also assign a VM to act maliciously. However, in both the cases, our CAMP defense model will show similar outcome.

Algorithm 1 Strategy selection.

Require: defense preferences (λ), preference of the CSP (β), no. of times attacker needs to launch co-resident attacks (n_s), per time unit cost before the game (H_0), cache access probability before the game begins (X_0), average probability per access (α), cumulative gain in previous time (G_{t-1}), cache access probability in previous time (X_{t-1}), and the set of VMs ($\mathbb{N}$)

1: Compute the initial belief ($\bar{\theta}_{j,0}$) of every VM considering the following items:
   1. R_j for every VM
   2. S_j for every VM
2: Compute the initial belief ($\bar{\theta}_{j,0}$) of every possible team combination with the maximum team size being 3 considering the following items:
   1. Average R_j of the team members
   2. Average S_j of the team members
3: Compute the dynamic belief ($\bar{\theta}_{j,t}$) of every VM considering the following item:
   1. $r_{(i,j),t}$ for every VM at attempt t
4: Compute the dynamic belief ($\bar{\theta}_{j,t}$) of every possible team combination with the maximum team size being 3 considering the following items:
   1. $r_{(\Gamma_u,j),t}$ for every VM at attempt t
5: Compute total belief ($\theta_t(x)$) from ($\bar{\theta}_{j,0}$) and ($\bar{\theta}_{j,t}$) for respective VMs and teams
6: SingleAttackerDecision($\theta_t(x), \lambda, n_s$)
   # Take decision considering single malicious VM
7: CollaborativeAttacksDecision($\theta_t(x), \lambda, n_s$)
   # Take decision considering collaborative attacks (attack team)

The defense mechanism follows Algorithm 1. The algorithm takes values of different parameters as input and calculates initial belief. The algorithm then continues to observe the behavior of different VMs and updates dynamic beliefs. Based on the total beliefs, it tries to identify potential attacker VMs. In an attack team, there can be a maximum of three VMs, who are collaborating to launch an attack against the victim VM. While the defender is not certain about an attacker, the attacker identification process continues in three steps, which are demonstrated in Algorithm 1. As a first step, it considers a single attacker and tries to identify that attacker. This process is demonstrated in Algorithm 2. The defender maintains an action set for every VM. In this step, an appropriate action is added to the respective VM's action set. After this, a team attack or collaborative attack is considered, which is demonstrated in Algorithm 3. In this step, a team size of 2 and 3, respectively, is considered. The defender finds appropriate actions for every team and those actions are added to every VM's action set. After this, the defender chooses the more severe action from every VM's action set and applies it to the corresponding VM.

Algorithm 2 Single attacker action decision.

procedure SingleAttackerDecision($\theta_t(x), \lambda, n_s$)
  for $VM_i \in \mathbb{N}$ select appropriate action **do**
  # check the constraints for *Throttle* action
    if ($\frac{\theta_t}{1-\theta_t} \geq \frac{1-\lambda}{\lambda}\frac{n_s\beta H_0}{\alpha X_0+\beta X_0}$ and $\frac{\mu}{(1-\mu)} \leq \frac{1-\lambda}{\lambda}\frac{\beta H_0 n_s}{G_{t-1}+X_{t-1}}$) **then**
    # Add *Throttle* action in the action set $\mathbb{A}_i$ of VM_i and update belief
      $\mathbb{A}_i \leftarrow \mathbb{A}_i \cup$ *Throttle*
    else # check the constraints for *Prime* action
      if ($\frac{\theta_t}{1-\theta_t} \geq \frac{1-\lambda}{\lambda} n_s$ and $\frac{\mu}{1-\mu} \leq \frac{(1-\lambda)\alpha X_0 n_s}{G_{t-1}+X_{t-1}}$) **then**
      # Add *Prime* action in $\mathbb{A}_i$ and update belief
        $\mathbb{A}_i \leftarrow \mathbb{A}_i \cup$ *Prime*
      else # Do nothing
        $\mathbb{A}_i \leftarrow \mathbb{A}_i \cup$ *Abstain*
      end if
    end if
  end for
end procedure

Algorithm 3 Collaborative attacks action decision.

procedure CollaborativeAttacksDecision($\theta_t(x), \lambda, n_s$)
  for $\Gamma_u \in \Gamma$ select action for teams **do**
    if ($\frac{\theta_t}{1-\theta_t} \geq \frac{1-\lambda}{\lambda}\frac{n_s\beta H_0}{\alpha X_0+\beta X_0}$ and $\frac{\mu}{(1-\mu)} \leq \frac{1-\lambda}{\lambda}\frac{\beta H_0 n_s}{G_{t-1}+X_{t-1}}$) **then**
      for $VM_i \in \Gamma_u$ select action for every team member **do**
        $\mathbb{A}_i \leftarrow \mathbb{A}_i \cup$ *Throttle*
      # Add *Throttle* action in the action set $\mathbb{A}_i$ of VM_i
      end for
    else
      if ($\frac{\theta_t}{1-\theta_t} \geq \frac{1-\lambda}{\lambda} n_s$ and $\frac{\mu}{1-\mu} \leq \frac{(1-\lambda)\alpha X_0 n_s}{G_{t-1}+X_{t-1}}$) **then**
        for $VM_i \in \Gamma_u$ select action for every team member **do**
          $\mathbb{A}_i \leftarrow \mathbb{A}_i \cup$ *Prime*
        # Add *Prime* action in the action set $\mathbb{A}_i$ of VM_i
        end for
      else
        for $VM_i \in \Gamma_u$ **do**
          $\mathbb{A}_i \leftarrow \mathbb{A}_i \cup$ *Abstain*
        # Add *Abstain* in the action set $\mathbb{A}_i$ of VM_i
        end for
      end if
    end if
  end for
  # Apply defensive action on every VM.
  for $VM_i \in \mathbb{N}$ **do**
    Apply the more conservative action from the action set $\mathbb{A}_i$ for VM_i
  end for
end procedure

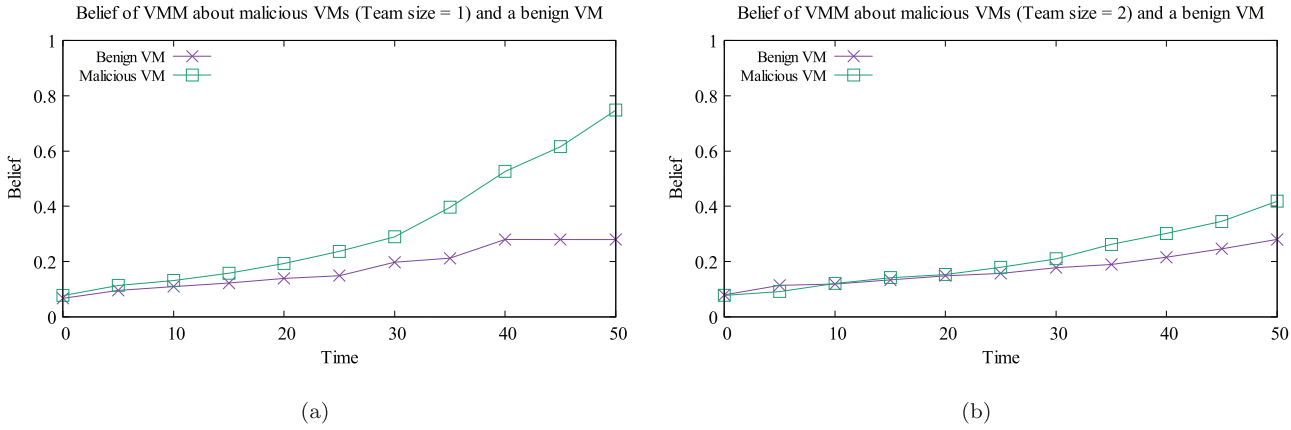

Fig. 6. (a) Belief about VMs when 1-VM attack team is considered. (b) Belief about VMs when 2-VM attack team is considered.

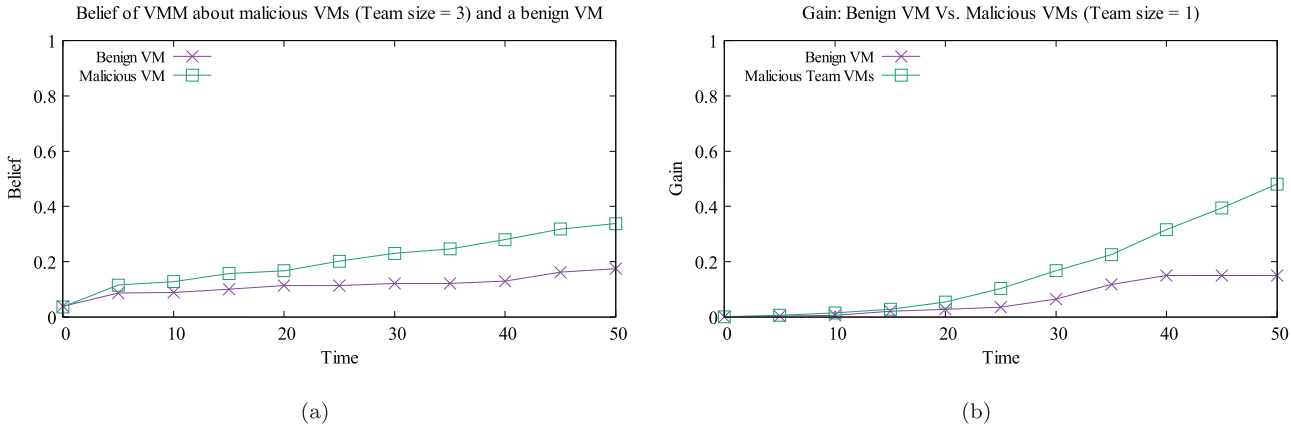

Fig. 7. (a) Belief about VMs when 3-VM attack team is considered. (b) Gain of a 1-VM attack team against a benign VM.

6.2. Characteristics analysis

We observe how the beliefs about different VMs change when an attacker deploys an attack team. When there are multiple attacker VMs, the identification process becomes difficult for the defender. Fig. 6(a) shows how the defender changes its belief while considering a single VM attack team. Fig. 6(b) shows how the defender's belief changes against a benign VM and an attacker VM, who is the member of a 2-VM attack team. Fig. 7(a) shows how the defender beliefs change against a benign VM and an attacker VM, who is the member of a 3-VM attack team. If we compare Figs. 6(b) and 7(a), we observe that the belief about an attacker VM of a 3-VM team is very close to a benign VM whereas the belief about an attacker VM of a 2-VM team is a bit far off from a benign VM. This phenomenon indicates that as the attacker increases the number of VMs in his attack team i.e., with the increase of team size, it becomes difficult for the defender to identify the attacker VM(s). When the team size increases, the attacker can choose his attack VM from a wider range of options. This way, he does not have to deploy the same VM several times and the hypervisor does not see an attacker VM's behavior as *greedy*. With the increase of attack team size, the attacker VMs' behavior becomes like that of a benign VM.

6.3. Performance analysis

In this section, we discuss how the size of the attack team impacts the attacker's gain and how our CAMP defense mechanism can help to thwart the attacker.

6.3.1. Gain and the team size

Figs. 7(b), 8(a), and (b) show the gain acquired by a single VM attack team, 2-VM attack team, and 3-VM attack team, respectively, against a benign VM. We observe that the gain by the 3-VM team is greater than that of the 2-VM attack team. This indicates that the attacker is better off increasing the team size. Fig. 9(a) and (b) show the 2-VM team gain and 3-VM team gain, respectively, against a lone attacker VM. We observe that the team gain outperforms the individual gain in both scenarios. This motivates the attacker to form an attack team because the attacker is more likely to be successful deploying an attack team rather than a single attacker VM.

Fig. 10(a) strengthens the claim of deploying an attack team with increasing team size, as we can see that the gain of the 3-VM team is greater than that of both the 2-VM team and a single attacker VM. However, the attacker cannot increase the team size arbitrarily because of the cost involved in maintaining VMs in the cloud (see Section 4.4.2). The attacker needs to find an optimal team size. We describe the rationale behind this in Section 4.2. When the CAMP game defense mechanism is in place, the attacker's gain is reduced significantly. We observe in Fig. 10(b), the gain of a lone attacker VM in a 2-VM team gets throttled down drastically when the CAMP game defense is employed compared to its gain without the defense mechanism.

As we can observe in Fig. 11(a), the defense mechanism is successful against a collaborative attack. We observe that the gain of a 2-VM team reduces significantly. In both of the cases, the attacker is unable to access the full information. We observe a similar scenario for a 3-VM team in the case of a gain for a single attacker VM and a team in Figs. 11(b) and 12(a), respectively.

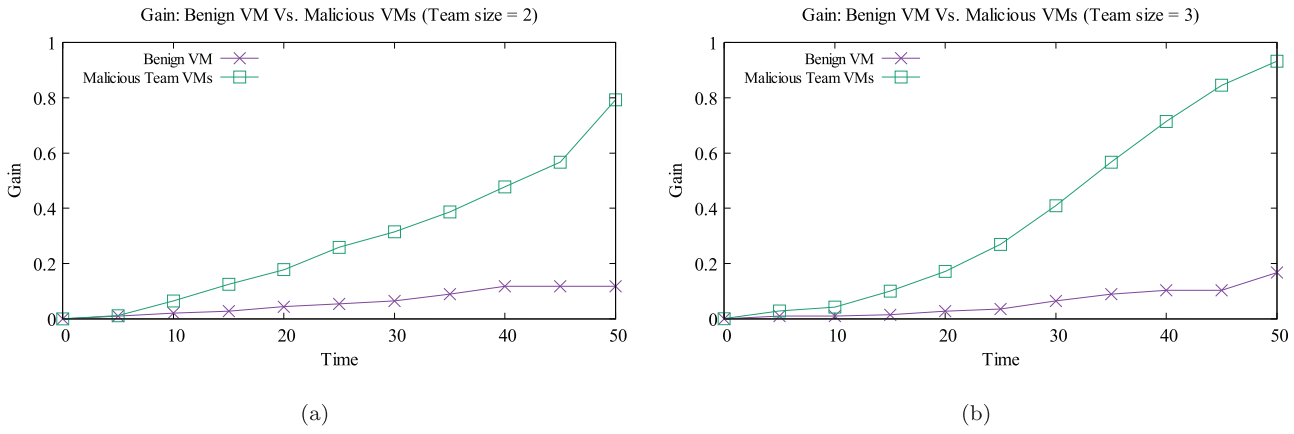

Fig. 8. (a) Gain of a 2-VM attack team against a benign VM. (b) Gain of a 3-VM attack team against a benign VM.

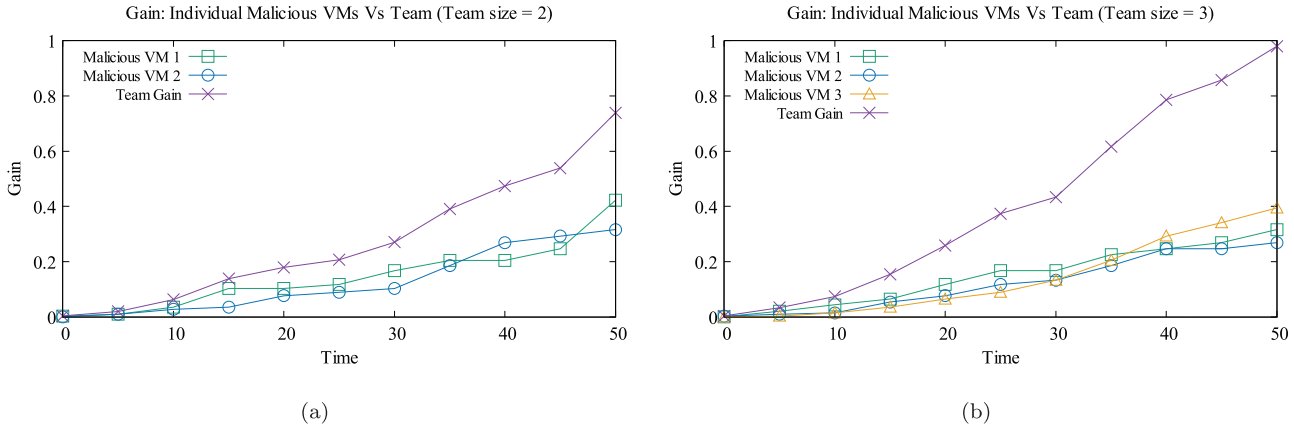

Fig. 9. (a) Gain of a 2-VM attack team along with the gains of lone attacker VM. (b) Gain of a 3-VM attack team along with the gains of a lone attacker VM.

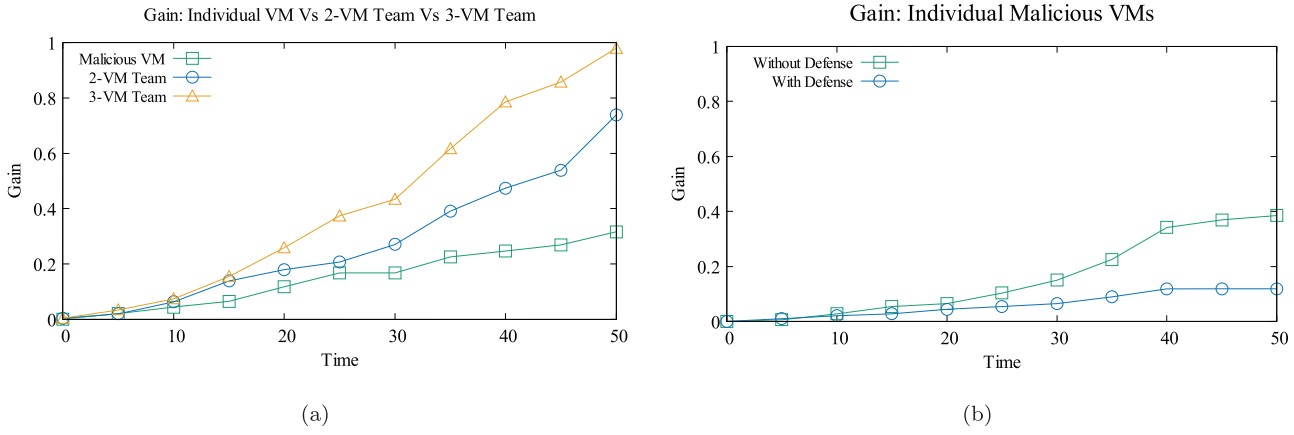

Fig. 10. (a) Gain of a lone attacker VM along with the gains of 2-VM attack team and 3-VM attack team. (b) Average gain of an attacker VM in a 2-VM team without CAMP defense and with CAMP defense.

However, if we compare Figs. 10(b) and 11(b), we observe that the gain of a malicious VM in a 2-VM team is greater than that of a 3-VM team because, in the 2-VM team, an attacker VM is deployed more often than in the case of the 3-VM team. We observe contrasting scenarios when we compare Figs. 11(a) and 12(a). We observe the team gain of a 3-VM team is greater than that of a 2-VM team because 3 VMs contribute more as a whole than 2 VMs, which again corroborates the fact that the team size and gain have a positive relationship. However, the attacker cannot increase the team size arbitrarily because of the underlying cost involved in launching VMs and maintaining them.

We observe from Fig. 12(b) that with the increase in team size the total gain increases but at the same time the cost also increases. This increase in cost will demotivate the attacker from increasing the team size, which might ultimately result in the discontinuation of the attacking process.

6.3.2. CAMP game performance

The hypervisor takes the *Prime* action as the 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 that it is a malicious VM and the

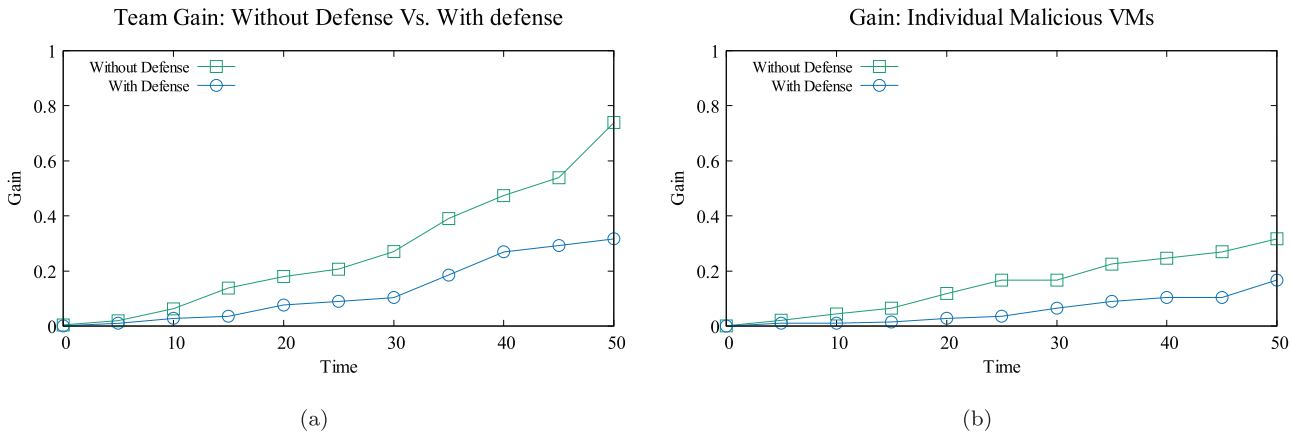

Fig. 11. (a) Gain of a 2-VM attack team without CAMP defense and with CAMP defense. (b) Gain of a single attacker VM in a 3-VM team without CAMP defense and with CAMP defense.

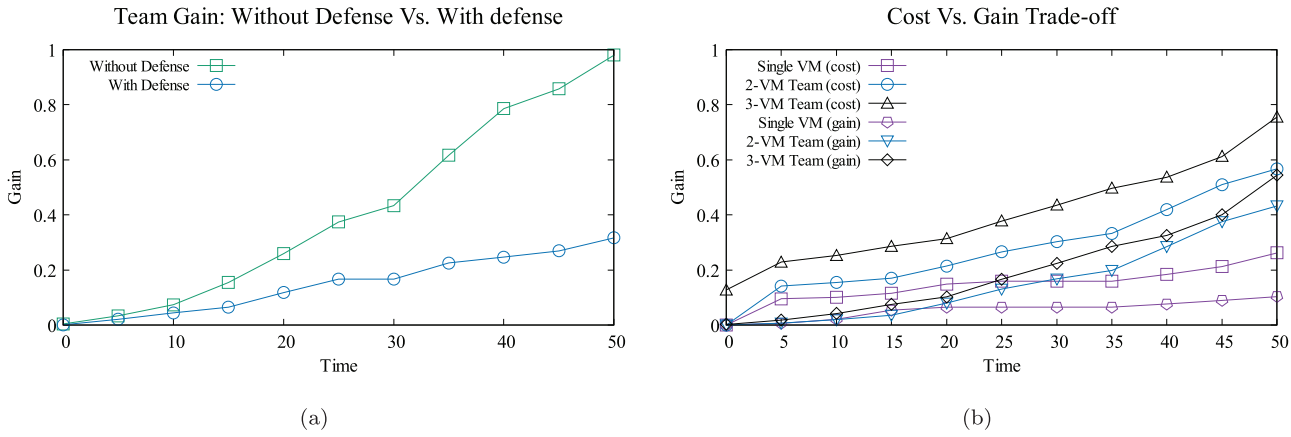

Fig. 12. (a) Gain of a 3-VM attack team without CAMP defense and with CAMP defense. (b) Cost Vs. Gain Trade-off.

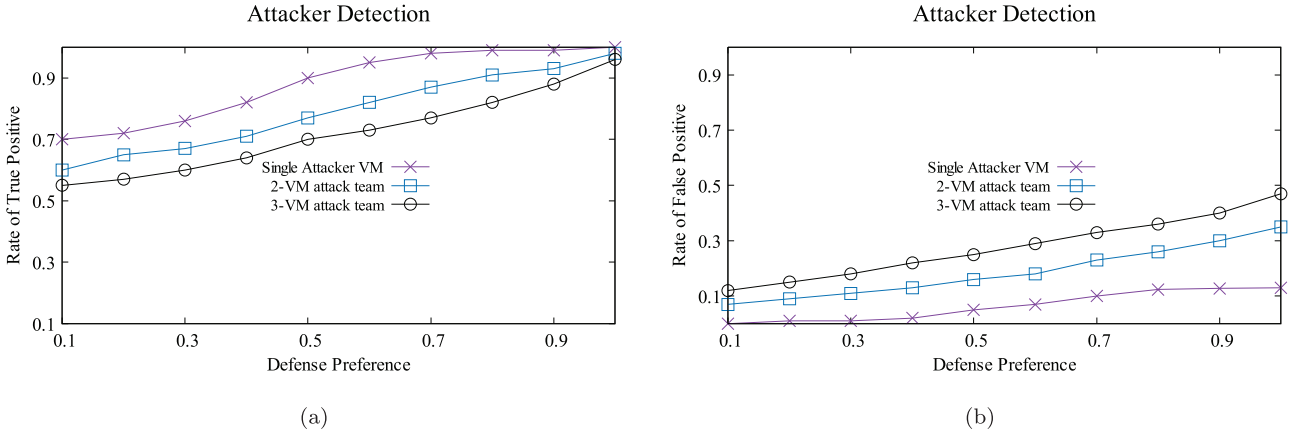

Fig. 13. Success rate of the attacker identification process with respect to defense preferences: (a) True positive rate and (b) false positive rate.

hypervisor restricts 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}), and the VM will be transferred to another server.

The hypervisor can use a different preference level (λ) for defense. The actions taken by the VMM can be heavily affected by this preference level. We observe in Fig. 13(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 in defending), the number of times the VMM takes the *Abstain* action increases. In particular, when λ is 0.1, the recall is 70% and when λ is 0.9, the recall is 98.6%. The F-measures for the defense preference of 0.1 and 0.9 are 0.7875 and 0.8187, respectively. We also observe that precisions are 0.9 and 0.7 for the preference level of 0.1 and 0.9, respectively. All of these indicate that the hypervisor needs to choose its preference level wisely based on its requirement *i.e.,* if the hypervisor takes more conservative approach, some benign VMs might be identified as malicious and vice versa. The figure also shows that the attacker identification process gets tougher with the increase in the team size. We also

observe the similar influence of the defense preference level in Fig. 13(b), which shows that false identification of the attacker increases with the increase of the preference level. Here, as well, we notice that false positive rate increases with the increase in team size.

The attacker can get several advantages in deploying collaborative attacker VMs rather than a single attacker VM. The defender, on the other hand, finds it tough to defend against collaborative attacks. Nonetheless, the defender can provide better security by opting for higher preference levels, as is indicated by better true-positive rates. The higher preference level also invites an increased true-positive rate. However, the underlying cost imposed by our CAMP model dissuades the attacker from continuing collaborative attacks. So, irrespective of the number of attacker VMs, our CAMP model can effectively prevent the attacker from launching both the collaborative attacks and single VM attack, which is supported by our experimental results.

7. Discussion

In this section, we discuss several points that further explicate the scope of this research.

7.1. Cloud environment

This research assumes a cloud environment where the same VM images provide the same services. In this case, we consider "standard cloud servers" instead of "clustered cloud servers." Standard servers accelerate performance by significantly reducing the I/O latency. Moreover, due to the storage differences and single server configurations, the standard server model offers economical pricing. Though clustered servers increase redundancy and availability, due to their higher I/O latency and prices, the standard servers are well adopted [48]. Most importantly, in the case of clustered servers, the virtual machines are transferred on top of one physical node to another when the first fails, which does not align with our game model. We also do not consider new technology like Intel SGX (Software Guard Extensions), which allows user-level code to allocate private regions of memory, called enclaves, that are protected from processes running at higher privilege levels [49].

7.2. Defense mechanism deployment

The proposed defense mechanism will be deployed by a CSP, where the hypervisor will work as a dedicated agent to shield the system by protecting the VMs from attacks. Since computing is rapidly moving to clouds while cyberattacks are increasingly targeting clouds, the security of cloud computing has become crucial and the CSPs must strive for providing secure environments to safeguard the subscribed VMs and maintain the reputation. In this case, the hypervisor will play a crucial role of identifying potential co-resident attacks. A CSP will assign the hypervisor to run the CAMP defense mechanism to detect the malicious VMs and thwart potential attacks.

7.3. Parameter units

We do not resort to any particular unit for time, cost, etc. with respect to a co-resident attack or corresponding defense. The time required to perform a successful co-resident attack or detect one will vary based on system configuration, which has little to do with our CAMP defense mechanism. The most important complexity parameter associated to a co-resident attack is the number of times the attacker (malicious VMs) needs to access the physical resources (L3 cache) right after the victim has accessed it. The target of the attacker is to achieve the successive access for the necessary number of times, whereas the defender's objective is to identify the attacker/VMs before the attack is successfully completed. Hence, the actual time duration for the attack process can be ignored to abstract the periods needed for a successful attack or identifying malicious VMs. The similar explanation also goes for the other units.

7.4. Choice of the game

The signaling game starts with a "decision" by nature, which determines whether player 1 (sender) is of type I (malicious) or II (benign) with a certain probability. Player 1, as his type is specified, must decide whether to play a certain action. Since player 1 takes actions according to the type, the actions are conditioned on it. After player 1 moves, player 2 is able to see the action taken by player 1 but she is unaware of the type. Hence, she decides about the action to play based on knowledge she earned from the observed action. Player 1 should have some beliefs about how player 2 will eventually play. The belief can evolve over the time. Since the problem of efficiently defending co-resident attacks needs to deal with different types of senders (*i.e.*, the benign and the malicious VMs) with respect to the defender (*i.e.*, the VMM), the signaling game is a perfect mechanism for modeling this problem.

7.5. Future direction

In the future, we want to experiment our CAMP defense mechanism in a real cloud environment. In this case, we will create a private cloud environment and will deploy the CAMP defense mechanism on the hypervisor. We will evaluate the proposed solution for both individual and collaborative attack scenarios. We will also try to see if we can deploy and evaluate the solution on the pubic clouds. It will be interesting to see how different cloud environments respond to co-resident attacks. We also want to explore how well our defense mechanism scales up. It will also be interesting to see how our defense mechanism performs, along with other existing proactive defense mechanisms. We want to explore some of the shortcomings of our work. In particular, as we previously mentioned, several parameters/factors (e.g., λ, T_M, etc.) impact the game and so the equilibrium result. We will further analyze these parameters to understand the system settings for the defender so that the defense success increases. We also want to explore the possibility of applying CAMP game in other distributed computing environments where similar attack and defense scenarios exist. It will be interesting to see whether there can be other forms of co-resident attacks which might take place because of some resource sharing and which are not cache-based attacks.

8. 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 because, as our evaluation has proved, the solution to the CAMP game provides optimal strategies to detect and defend the malicious VMs, while limiting the impact on the benign VMs. The proposed mechanism is shown to be effective

in both of the attack cases, using a single VM or a number of collaborative VMs. Furthermore, the CAMP game can deal with both the erratic greedy behavior from benign VMs and the sudden normal behavior from malicious VMs, which aligns with our selection.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Ristenpart T, Tromer E, Shacham H, Savage S. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In: Proceedings of the 16th ACM conference on computer and communications security. ACM; 2009. p. 199–212.

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

[3] Bernstein DJ. Cache-timing attacks on aes. Technical Report; 2005.

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

[5] Bernstein DJ, Lange T, Schwabe P. The security impact of a new cryptographic library. In: International conference on cryptology and IS in Latin America. Springer; 2012. p. 159–76.

[6] Hasan M, Rahman M. A signaling game approach to mitigate co-resident attacks in an iaas cloud environment. http://www.csc.tntech.edu/~marahman/papers/Technical_Reports/Coresident-Attack_Technical-Report_Mehedi-Rahman.pdf; Technical Report.

[7] Tromer E, Osvik D, Shamir A. Efficient cache attacks on aes, and countermeasures. J Cryptol 2010;23(1):37–71.

[8] Inci M, Gülmezoglu B, Apecechea G, Eisenbarth T, Sunar B. Seriously, get off my cloud! cross-vm rsa key recovery in a public cloud. IACR Cryptol. 2015;2015:898. ePrint Archive

[9] Camerer CF. Behavioral game theory: experiments in strategic interaction. Princeton University Press; 2011.

[10] Gibbons R. Game theory for applied economics. Princeton University Press; 1992.

[11] Hasan MM, Rahman MA. Protection by detection: a signaling game approach to mitigate co-resident attacks in cloud. In: IEEE 10th international conference on cloud computing (CLOUD); 2017. p. 552–9.

[12] Han Y, Alpcan T, Chan J, Leckie C, Rubinstein B. A game theoretical approach to defend against co-resident attacks in cloud computing: preventing co-residence using semi-supervised learning. IEEE Trans Inf ForensicsSecurity 2016;11(3):556–70.

[13] Han Y, Alpcan T, Chan J, Leckie C. Security games for virtual machine allocation in cloud computing. In: International conference on decision and game theory for security. Springer; 2013. p. 99–118.

[14] Jensen M, Schwenk J, Gruschka N, Iacono LL. On technical security issues in cloud computing. In: Cloud computing, 2009. CLOUD'09. IEEE international conference on. IEEE; 2009. p. 109–16.

[15] Bedi HS, Shiva S. Securing cloud infrastructure against co-resident dos attacks using game theoretic defense mechanisms. In: Proceedings of the ICACCI'12. ACM; 2012. p. 463–9.

[16] Liu F. Catalyst: Defeating last-level cache side channel attacks in cloud computing. In: IEEE HPCA; 2016. p. 406–18.

[17] Shi J, Song X, Chen H, Zang B. Limiting cache-based side-channel in multi-tenant cloud using dynamic page coloring. In: 41st IEEE DSN-W; 2011. p. 194–9.

[18] Vattikonda B, Das S, Shacham H. Eliminating fine grained timers in xen. In: Proceedings of the 3rd ACM workshop on cloud computing security workshop; 2011. p. 41–6.

[19] Wu J, Ding L, Lin Y, Min-Allah N, Wang Y. Xenpump: a new method to mitigate timing channel in cloud computing. In: Cloud computing (CLOUD), 2012 IEEE 5th international conference on. IEEE; 2012. p. 678–85.

[20] Jin S, Ahn J, Cha S, Huh J. Architectural support for secure virtualization under a vulnerable hypervisor. In: Microarchitecture (MICRO), 2011 44th annual IEEE/ACM international symposium on. IEEE; 2011. p. 272–83.

[21] Szefer J, Keller E, Lee R, Rexford J. Eliminating the hypervisor attack surface for a more secure cloud. In: Proceedings of the 18th ACM conference on CCS; 2011. p. 401–12.

[22] Zhang Y, Reiter M. Düppel: retrofitting commodity operating systems to mitigate cache side channels in the cloud. In: ACM CCS; 2013. p. 827–38.

[23] Luo X, Yang L, Ma L, Chu S, Dai H. Virtualization security risks and solutions of cloud computing via divide-conquer strategy. In: Multimedia information networking and security (MINES), 2011 third international conference on. IEEE; 2011. p. 637–41.

[24] Manshaei MH, Zhu Q, Alpcan T, Bacşar T, Hubaux J-P. Game theory meets network security and privacy. ACM Comput Surv (CSUR) 2013;45(3):25.

[25] Roy S, Ellis C, Shiva S, Dasgupta D, Shandilya V, Wu Q. A survey of game theory as applied to network security. In: System sciences (HICSS), 2010 43rd Hawaii international conference on. IEEE; 2010. p. 1–10.

[26] Qiu Y, Shen Q, Luo Y, Li C, Wu Z. A secure virtual machine deployment strategy to reduce co-residency in cloud. In: Trustcom/BigDataSE/ICESS, 2017 IEEE. IEEE; 2017. p. 347–54.

[27] Wang Y, Guo C, Li T, Xu Q. Secure two-party computation in social cloud based on reputation. In: Advanced information networking and applications workshops (WAINA), 2015 IEEE 29th international conference on. IEEE; 2015. p. 242–5.

[28] Jebalia M, Letaifa AB, Hamdi M, Tabbane S. A revocation game model for secure cloud storage. In: High performance computing & simulation (HPCS), 2014 international conference on. IEEE; 2014. p. 1016–17.

[29] Hataba M, El-Mahdy A. Cloud protection by obfuscation: Techniques and metrics. In: P2P, parallel, grid, cloud and internet computing (3PGCIC), 2012 seventh international conference on. IEEE; 2012. p. 369–72.

[30] Levitin G, Xing L, Dai Y. Optimal data partitioning in cloud computing system with random server assignment. Future Gen Comput Syst 2017;70:17–25.

[31] Zhuang J, Bier VM, Alagoz O. Modeling secrecy and deception in a multiple-period attacker–defender signaling game. Eur J Oper Res 2010;203(2):409–18.

[32] Çeker H, Zhuang J, Upadhyaya S, La QD, Soong B-H. Deception-based game theoretical approach to mitigate dos attacks. In: International conference on decision and game theory for security. Springer; 2016. p. 18–38.

[33] Kamhoua C, Martin A, Tosh DK, Kwiat KA, Heitzenrater C, Sengupta S. Cyber-threats information sharing in cloud computing: A game theoretic approach. In: Cyber security and cloud computing (CSCloud), 2015 IEEE 2nd international conference on. IEEE; 2015. p. 382–9.

[34] Kwiat L, Kamhoua CA, Kwiat KA, Tang J, Martin A. Security-aware virtual machine allocation in the cloud: A game theoretic approach. In: Cloud computing (CLOUD), 2015 IEEE 8th international conference on. IEEE; 2015. p. 556–63.

[35] Zhou Z, Zhang H, Yu X, Guo J. Audit meets game theory: Verifying reliable execution of sla for compute-intensive program in cloud. In: Communications (ICC), 2015 IEEE international conference on. IEEE; 2015. p. 7456–61.

[36] Varadarajan V, Ristenpart T, Swift M. Scheduler-based defenses against cross-vm side-channels. In: 23rd USENIX security symposium (USENIX Security 14); 2014. p. 687–702.

[37] Sundareswaran S, Squcciarini A. Detecting malicious co-resident virtual machines indulging in load-based attacks. In: International conference on ICS. Springer; 2013. p. 113–24.

[38] Yu S, Gui X, Lin J. An approach with two-stage mode to detect cache-based side channel attacks. In: The international conference on information networking(ICOIN). IEEE; 2013. p. 186–91.

[39] Han Y, Chan J, Alpcan T, Leckie C. Virtual machine allocation policies against co-resident attacks in cloud computing. In: Communications (ICC), 2014 IEEE international conference on. IEEE; 2014. p. 786–92.

[40] Sevak B. Security against side channel attack in cloud computing. Int J Eng Adv Technol (IJEAT) 2013;2(2):183.

[41] Zhang W, Jia X, Tai J, Wang M. Cacherascal: defending the flush-reload side-channel attack in paas clouds. In: International conference on wireless algorithms, systems, and applications. Springer; 2017. p. 665–77.

[42] Armbrust M, Fox A, Griffith R, Joseph AD, Katz R, Konwinski A, et al. A view of cloud computing. Commun ACM 2010;53(4):50–8.

[43] Voorsluys W, Broberg J, Venugopal S, Buyya R. Cost of virtual machine live migration in clouds: a performance evaluation. CloudCom 2009;9:254–65.

[44] Patel P, Ranabahu AH, Sheth AP. Service level agreement in cloud computing. https://corescholar.libraries.wright.edu/cgi/viewcontent.cgi?referer=&httpsredir=1&article=1077&context=knoesis; 2009. The Ohio Center of Excellence in Knowledge-Enabled Computing Publications.

[45] Li X, Du J. Adaptive and attribute-based trust model for service-level agreement guarantee in cloud computing. IET Inf Security 2013;7(1):39–50.

[46] Rajavel R, Mala T. Achieving service level agreement in cloud environment using job prioritization in hierarchical scheduling. In: Proceedings of the international conference on information systems design and intelligent applications 2012 (INDIA 2012) held in Visakhapatnam, India, January 2012. Springer; 2012. p. 547–54.

[47] Calheiros RN, Ranjan R, Beloglazov A, De Rose CA, Buyya R. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software 2011;41(1):23–50.

[48] Zhang Q, Cheng L, Boutaba R. Cloud computing: state-of-the-art and research challenges. J Internet Serv Appl 2010;1(1):7–18.

[49] Costan V, Devadas S. Intel sgx explained. IACR Cryptol ePrint Archive 2016;2016(086):1–118.

Md Golam Moula Mehedi Hasan is a Ph.D. student in the Department of Computer Science at Tennessee Tech University, USA. He earned his MS degree in Computer Science at Tennessee Tech in 2017. Earlier, he received his BS in Computer Science and Engineering from Bangladesh University of Engineering and Technology, Dhaka in 2009. Then, he worked as a software engineer in several multi-national financial companies till 2015. Hasan's primary research focuses on game-theoretic and machine learning-based analyses of security and dependability problems as well as developing security solutions using the crypto/Blockchain technology for clouds, smart grids, and autonomous vehicles.

Mohammad Ashiqur Rahman is an Assistant Professor in the Department of Electrical and Computer Engineering at Florida International University (FIU), USA. Before joining FIU, he was an Assistant Professor at Tennessee Tech University. He received the BS and MS degrees in computer science and engineering from Bangladesh University of Engineering and Technology, Dhaka, in 2004 and 2007, respectively, and obtained the Ph.D. degree in computing and information systems from the University of North Carolina at Charlotte in 2015. Rahman's primary research interest covers a wide area of computer networks and communications, within both cyber and cyber-physical systems. His research focus primarily includes computer and information security, risk analysis and security hardening, secure and dependable resource allocation and optimal management, and distributed and parallel computing. He has already published over 50 peer-reviewed journals and conference papers. He has also served as a member in the technical programs and organization committees for various IEEE and ACM conferences.