Security Games for Virtual Machine Allocation in Cloud Computing

Yi Han[1], Tansu Alpcan[2], Jeffrey Chan[1], and Christopher Leckie[1]

[1] Department of Computing and Information Systems
[2] Department of Electrical and Electronic Engineering
University of Melbourne, Melbourne, Australia
andrew.hanyi@gmail.com,
{tansu.alpcan,jeffrey.chan,caleckie}@unimelb.edu.au

Abstract. While cloud computing provides many advantages in accessibility, scalability and cost efficiency, it also introduces a number of new security risks. This paper concentrates on the co-resident attack, where malicious users aim to co-locate their virtual machines (VMs) with target VMs on the same physical server, and then exploit side channels to extract private information from the victim. Most of the previous work has discussed how to eliminate or mitigate the threat of side channels. However, the presented solutions are impractical for the current commercial cloud platforms. We approach the problem from a different perspective, and study how to minimise the attacker's possibility of co-locating their VMs with the targets, while maintaining a satisfactory workload balance and low power consumption for the system. Specifically, we introduce a security game model to compare different VM allocation policies. Our analysis shows that rather than deploying one single policy, the cloud provider decreases the attacker's possibility of achieving co-location by having a policy pool, where each policy is selected with a certain probability. Our solution does not require any changes to the underlying infrastructure. Hence, it can be easily implemented in existing cloud computing platforms.

Keywords: Cloud computing, co-resident attack, game theory, virtual machine allocation policy.

1 Introduction

In cloud computing environments, when a user requests to start a new machine, in most cases the allocated machine is not an entire physical server, but only a virtual machine (VM) running on a specific host. This is enabled by hardware virtualisation technologies [1] such as Hyper-V, VMWare, and Xen, so that multiple VMs of different users can run on the same physical server and share the same underlying hardware resources. While this increases the utilisation rate of hardware platforms, it also introduces a new threat: although in theory, VMs running on the same server (i.e., co-resident VMs) should be logically isolated from each other, malicious users can still circumvent the logical isolation, and obtain sensitive information from co-resident VMs [2].

It has been shown that this new co-resident attack (also known as a co-residence attack, or co-location attack) is indeed feasible in real cloud platforms. By building different kinds of side channels, the attacker can extract a range of private statistics, from the coarse-grained [2], like the victim's workload and traffic rate, to the fine-grained [3], such as cryptographic keys.

Most of the previous work has focused on the side channels, and proposed to solve the problem either by mitigating the threat of side channels [4-7], or designing a new architecture for the cloud system to eliminate side channels [8, 9]. However, few of these are practical for current commercial cloud platforms as they require significant changes to be made. In this paper, we address this issue with a novel decision and game-theoretic approach.

The co-resident attack that we are discussing comprises two steps. Before the attacker can extract any useful information from the victim, they first need to co-locate their own VMs with the target VM. Experiments in [2] show that the attacker can achieve a surprisingly high efficiency rate of 40% (i.e., 4 out of 10 malicious VMs launched by the attacker will be co-resident with the target(s)). This observation motivates us to study practical methods for decreasing this efficiency rate of co-resident attacks. For cloud providers, one important factor they can control that will influence the efficiency rate is the VM allocation policy. Hence, we compare different VM allocation policies in cloud computing, and investigate the impact of these policies on the efficiency of achieving co-residence.

When cloud providers decide on their VM allocation policies, workload balance and power consumption are used as additional important criteria. Therefore, we carry out a comparative study of four basic VM allocation policies using a game theoretic approach, namely: choosing the server (1) with the least number of VMs, (2) with the most number of VMs, (3) randomly, and (4) on a round robin basis. These policies form the basis of most policies used in real cloud systems. Specifically, we model this as a two-player security game, where the attacker's goal is to maximize the attack efficiency, while the defender (cloud provider) aims to minimize it on the premise of balancing the workload and maintaining low power consumption.

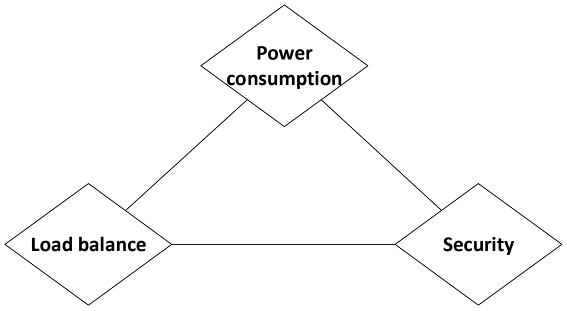


Fig. 1. Different focuses of VM allocation policies

Our **contributions** in this paper include: (1) we introduce a novel game theoretic approach to the problem of defending against co-resident attacks; (2) we model different VM allocation policies using zero- and non-zero-sum security games; (3) we perform extensive numerical simulations to develop and evaluate a practical solution for mitigating the threat of co-resident attacks; and (4) we show that in terms of minimising attack efficiency, the deterministic VM allocation policy behaves the worst, while a mixed policy outperforms any single policy.

The rest of the paper is organized as follows. In Section 2, we further introduce the focus of our study – the co-resident attack, and give our problem statement. In Section 3, we propose our game model. A detailed analysis and comparison of the different VM allocation policies is presented in Section 4, while Section 5 concludes the paper and gives directions for our future work.

2 Background and Problem Statement

In this section, we first introduce the co-resident attack in detail. We discuss how to achieve co-residence, the security risks, and potential countermeasures. We then define the problem that we aim to solve in this paper.

2.1 Methods to Achieve Co-residence

In order to achieve co-residence, i.e., locate their own VM and the victim on the same host, the attackers have several options.

1. The most straightforward approach is to use a brute-force strategy: start as many VMs as possible until co-residence is achieved.
2. Experiments in [2] show that in the popular Amazon EC2 cloud, there is strong sequential and parallel locality in VM placement, which means if one VM is terminated right before another one is started, or if two VMs are launched almost at the same time, then these two VMs are often assigned to the same server. As a result, the attacker can increase the possibility of co-locating their VM with the targets if they are able to trigger the victim to start new VMs, and then launch their own VMs after that.

2.2 Potential Security Risks

After co-residence is achieved, there are a number of potential security risks:

1. *VM workload estimation* – In [2], the authors adopt the Prime+Probe technique [10, 11] to measure cache utilisation. The basic idea is that the execution time of the cache read operation is heavily influenced by the cache utilisation. Hence, by performing intensive read operations and then measuring the execution time, the attacker can infer the cache usage, which also indicates the target VM's workload.
2. *Web traffic rate estimation* – Similarly, the attacker performs cache load measurements on the co-resident VM, and at the same time, they send HTTP requests from

non-co-resident VM(s) to the victim. Experimental results show that there is a strong correlation between the execution time of the cache operation and the HTTP traffic rate. In other words, the attacker is able to obtain information about the web traffic rate on the co-resident VM. This can be useful information if the victim is a corporate competitor.

3. *Private key extraction* – In [3], the authors demonstrate that it is possible to extract cryptographic keys by using cross-VM side channels. In particular, they show how to overcome the following challenges: regaining the control of the physical CPU with sufficient frequency to monitor the instruction cache, filtering out hardware and software noise, and determining if an observation is from the target virtual CPU or not due to the core migration.

In addition, there are a number of papers discussing how to build side channels between co-resident VMs in cloud computing environments [12-17].

Note that there are other types of denial-of-service attacks where the attacker does not care who the victim is, and only aims to obtain an unfair share of resources from the physical server, so that co-resident VMs will experience a degradation of quality of service [18-21]. This type of attack is outside the scope of our research.

2.3 Possible Countermeasures

Previous studies have proposed a number of possible defence methods, which can be broadly classified into the following three categories:

1. *Preventing the attacker from verifying co-residence* – In current cloud computing platforms, it is relatively easy to check if two VMs are on the same host. For example, by performing a *TCP traceroute* the attacker can obtain a VM's Dom0 IP address (where Dom0 is a privileged VM that manages other VMs on the host). If two Dom0 IP addresses are the same, the corresponding VMs are on the same server. If we can prevent the attacker from verifying whether their own VM and the target victim's VM are on the same physical machine, then they will not be able to launch further attacks. However, there are a number of alternative methods to verify co-residence that do not rely on network measurement [2], even though they are more time-consuming. Therefore, it is difficult, if not impossible, to prevent all these methods.
2. *Securing the system to prevent sensitive information of a VM from being leaked to co-resident VMs* – Countermeasures against side channels have already been extensively studied, including (1) mitigating the threat of timing channels by eliminating high resolution clocks [5], or adding latency to potentially malicious operations [6], and (2) redesigning the architecture for cloud computing systems [8, 9]. Nevertheless, these methods are usually impractical for current commercial cloud platforms due to the substantial changes required.
3. *Periodically migrating VMs* – The authors in [22, 23] propose to solve the problem by periodically migrating VMs. The number of chosen VMs and hosts are decided based on game theory. In addition, they also discuss how to place VMs in order to minimize the security risk. However, frequently migrating VMs may increase

power usage and lead to load imbalances, which are undesirable from the cloud provider's perspective.

2.4 Problem Statement

In this paper, we aim to find a solution for defending against the co-resident attack. In order to make our proposed method practical, we assume that the cloud providers (1) do not have any prior knowledge of the attacker; (2) will not apply any additional security patches, and (3) will not have access to an effective detection mechanism. Therefore, the question is under these assumptions, how can they mitigate the threat of co-resident attacks, while maintaining a reasonably high workload balance and low power consumption for the system?

3 Proposed Game Model

We consider this problem as a static game between the attacker and the defender (cloud provider). In this section, we first define the attack and defence scenarios, and then propose our game model.

3.1 Attack Scenarios and Metrics

Before giving the formal description of the game model, we first define the attack scenario: in a system of N servers, there are k (separate) attackers $\{A_1, A_2, \ldots, A_k\}$, each controlling one single account. No limit on the number of VMs is enforced for an account, which means the attackers can start as many VMs as frequently as they want (in practice, the attackers maybe restricted by costs and other factors). The target for attacker A_i is the set of VMs started by legitimate user L_i, i.e., $Target(A_i) = \sum_t VM(L_i,t) = \{VM_{i1}, VM_{i2}, \ldots, VM_{iT_i}\}$, where $VM(L_i,t)$ is the set of VMs started by L_i at time t. During one attack started at time t, A_i will launch a number of VMs, $VM(A_i,t)$. Let $SuccVM(A_i,t)$ denote the VMs of attacker A_i that co-locate with at least one of the targets, i.e., $SuccVM(A_i,t) = \{v \mid v \in VM(A_i,t), Servers(\{v\}) \subseteq Servers(Target(A_i))\}$, where $Servers(\{a\ set\ of\ VMs\})$ is the set of servers that host the set of VMs. Similarly, let $SuccTarget(A_i,t)$ denote the VMs of the target user L_i that are co-located with at least one VM of the attacker A_i, i.e., $SuccTarget(A_i,t) = \{u \mid u \in Target(A_i), Servers(\{u\}) \subseteq Servers(VM(A_i,t))\}$. Then an attack is considered as successful if $SuccVM(A_i,t)$ and $SuccTarget(A_i,t)$ are non-empty, i.e., at least one of the attacker's VMs is co-located with at least one of the target VMs.

In order to further measure the success for one attack, two definitions are introduced:

(1) *Efficiency*, which is defined as the number of malicious VMs that are successfully co-located with at least one of the T_i targets, divided by the total number of VMs launched during this attack, i.e.,

$$Efficiency(VM(A_i,t)) = \frac{|SuccVM(A_i,t)|}{|VM(A_i,t)|} \quad (1)$$

(2) *Coverage*, which is defined as the number of target VMs co-located with malicious VMs started in this attack, divided by the number of targets T_i, i.e.,

$$Coverage(VM(A_i,t)) = \frac{|SuccTarget(A_i,t)|}{|Target(A_i)|} \quad (2)$$

Table 1. Definitions of symbols used

Name	Definition
$Target(A_i)$	The target set of VMs that A_i intends to co-locate with. $\lvert Target(A_i)\rvert = T_i$
$VM(L_i,t)$	The set of VMs started by user L_i at time t
$SuccTarget(A_i,t)$	A subset of $Target(A_i)$ that co-locates with at least one VM from $VM(A_i,t)$
$SuccVM(A_i,t)$	A subset of $VM(A_i,t)$ that co-locates with at least one of the T_i targets
Servers({a set of VMs})	Servers that host the set of VMs

3.2 Defence Policies

Recall that the attack we consider comprises two steps. First, the attacker has a clear set of targets, and they will try different methods to co-locate their own VMs with the targets. Second, after co-residence is achieved, the attacker will use various techniques to obtain sensitive information from the victim.

Because of the assumptions we made in Section 2.4, any solution that focuses on the second step, and any attempt to identify the attacker or their VM requests are infeasible. Therefore, one of the remaining options for the defender is to find an allocation policy that minimizes the overall possibility of achieving co-residence.

For simplicity reasons, we only consider four policies, namely: choosing the server (1) with the least number of VMs ("Least VM"), (2) with the most number of VMs ("Most VM"), (3) randomly ("Random"), and (4) on a round robin basis ("Round Robin"). The reason why we choose these policies is that the first two are two extremes in the policy spectrum, with one spreading the workload and the other one concentrating the workload, while the other two are the most straightforward policies. In addition, most real cloud VM allocation policies are based on these four policies.

We can classify these policies into two main categories: deterministic (Policy 4), and stochastic (Policies 1, 2, 3).

Deterministic VM Allocation Policies

Round Robin: suppose that all the servers form a queue. When a new VM request arrives, all the servers in the queue will be checked sequentially from the beginning, until a server *Sr* is found with enough remaining resources. Server *Sr* will be selected to host the new VM, and all the servers that have been checked will be moved to the end of the queue, keeping the original order.

We classify the Round Robin policy as deterministic because the servers will be chosen with the same order, if the cloud system and the workload are the same.

Stochastic VM Allocation Policies

1. *Least VM/Most VM*: for every new VM request, the policy will select the server that hosts the least/most number of VMs, among those with enough resources left (n.b., if multiple servers meet the criterion, the policy will choose one randomly). This kind of policy spreads/concentrates the workload within the system for better workload balance/lower energy consumption.
2. *Random*: for every new VM request, the policy will randomly select one server from those having enough resources.

We classify these three policies as stochastic because in contrast to the deterministic policy, even if the same workload is submitted to the system, the order in which the servers are selected may still be different.

3.3 Game Model

Given the attack and defence scenarios, we define the two-player security game model [24] as follows.

Players

In this strategic game, there are two players, the attacker A, and the defender D: $P = \{A, D\}$.

Action Set

According to our earlier analysis, the defender treats every customer's request in the same way, and their action set is to choose a specific VM allocation policy: AS^D = {Least VM, Most VM, Random, Round Robin}. On the other hand, from the attacker's point of view, they can decide when to start the VMs, and how many VMs to start. In order to simplify the problem, we only consider one single attack (in reality, the attacker may launch the attack periodically). Hence, the action set of the attacker is: $AS^A = \{VM(A,t)\}$.

Utility Functions

The attacker's goal is to maximise the efficiency/coverage rate, and their utility function is:

$$\begin{aligned} U^A\left(VM(A,t), Policy\right) = w_A \cdot Efficiency(VM(A,t), Policy) + \\ \left(1 - w_A\right) \cdot Coverage(VM(A,t), Policy) \end{aligned} \quad (3)$$

where w_A is a weight that specifies the relative importance of efficiency vs. coverage, and $0 \leq w_A \leq 1$.

Note that compared with (2), the efficiency/coverage rate in (3) takes another parameter into consideration – *Policy* – since these two rates are different under the four allocation policies. In addition, the attacker's cost is also implicitly included, because the efficiency rate will be low if the attacker starts a large number of VMs.

In contrast, the defender's goal is to find a policy that achieves an optimal balance between minimising the attacker's efficiency/coverage rate, decreasing the overall power consumption, and balancing the workload. Suppose that P_i and B_i, $i = 1, 2, 3, 4$, represent the system's normalised power consumption and workload balance under the four policies respectively, then the defender's utility function is:

$$U^D\left(Policy\right) = -w_{D_1} \cdot U_i^A - w_{D_2} \cdot P_i + \left(1 - w_{D_1} - w_{D_2}\right) \cdot B_i \tag{4}$$

where U_i^A is the attacker's utility under policy i, $i = 1, 2, 3, 4$, and w_{D_1} and w_{D_2} are weights such that $0 \leq w_{D_1}, w_{D_2}, w_{D_1} + w_{D_2} \leq 1$.

Therefore, the security game G is written as $G = \{P, AS^i, U^i, i \in \{A, D\}\}$. In the next section, we discuss *Efficiency*(*A*,*VM*(*A*,*t*),*Policy*), *Coverage*(*A*,*VM*(*A*,*t*), *Policy*), P_i and B_i in detail.

4 Analysis of VM Allocation Policies Using the Game Model

In this section, we present a simulation-based analysis of the different VM allocation policies. First, we introduce the simulation platform for our experiments. Then we give the detailed results of the efficiency rate, coverage rate, power consumption and workload balance under the four policies, which will help us develop the appropriate parameters in the game model. Finally, we calculate the numerical solution for the game, and discuss the implications of our findings.

4.1 Simulation Environment

We conducted our experiments on the platform CloudSim [25], which has been widely used in previous studies [26, 27]. The settings for our experiments are as follows.

Physical Servers and Virtual Machines

All the configurations of servers and VMs used in our simulations are similar to those of certain real-world models. Note that in CloudSim the CPU speed is measured in MIPS instead of MHz (a higher value of MIPS indicates faster processing speed).

Table 2. Configurations of servers and VMs

	Type	Quantity	CPU speed (MIPS)	No. of CPU cores	RAM (MB)
Servers	1	150	2600	16	24576
VMs	1	random*	2500	1	870
	2	random*	2000	1	1740
	3	random*	1000	1	1740
	4	random*	500	1	613

* Each VM request randomly decides the type of VM it requires.

Background Workload
Our earlier study [28] shows that the VM request arrival and departure processes in cloud computing follow a power law distribution, and exhibit self-similarity. In order to make the background workload more realistic in our simulation, we implement this finding using the program developed in [29]. More specifically, we use this program to generate two self-similar time series, indicating the number of VM requests that arrive/departure in each minute. In addition, we assume that every new request needs only one VM, whose type and CPU utilization for each minute are both randomly selected.

Experimental Settings
In each experiment, a legal user L starts 20 VMs at the 18000th second (note that the system reaches the steady state in terms of the number of started VMs around the 4800th second, so our results are very unlikely to be affected by simulation boot-up behaviours), and a certain time later (we call this time difference the *lag*) at the t^{th} (t = 18000 + lag) second, an attacker A starts $VM(A,t)$ VMs. The simulation stops a few minutes after that. Both the lag and $VM(A,t)$ range from 1 to 100 (note that we use "lag" and "t" interchangeably in the rest of this paper).

For every VM allocation policy/lag combination, we carry out the above experiment 50 times, and the final results presented below are the average values.

4.2 Attack Efficiency under Different VM Allocation Policies

In this subsection, we summarise the attack efficiency under the four policies.

Least VM Allocation Policy
Fig. 2 shows the impact on the efficiency rate of varying the lag and the number of VMs started by the attacker ($VM(A,t)$) under the Least VM policy. The following observations can be made from the experiment:

(1) The number of started VMs, $VM(A,t)$, has little impact on the attack efficiency.

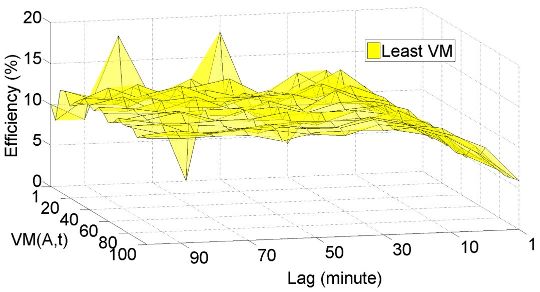
(2) When the lag is small, it is difficult to achieve co-residence. This is consistent with the aim of balancing the workload, which means it is unlikely that a server will be chosen twice within a short period of time.

(3) After the lag reaches 10 minutes, the efficiency rate remains stable. The only exception is when $VM(A,t)$ equals one: the efficiency rate is volatile, but the average value in this case is still similar to the overall average value.

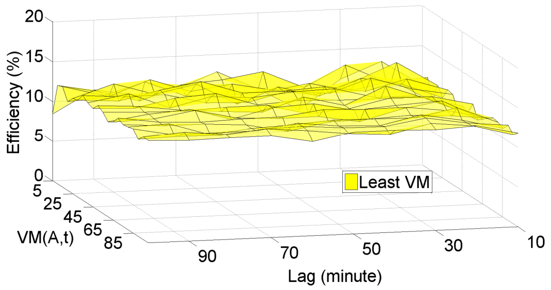
(4) It can be seen from Fig. 2(b) that when the lag is longer than 10 minutes and $VM(A,t)$ is larger than 5, the attack efficiency stays at approximately the same value.

Most VM Allocation Policy
Under the Most VM policy, our simulation shows that:

(1) In most cases, the efficiency rate first grows with $VM(A,t)$, but then the trend reverses. A closer inspection of the trace files shows that only the first $m(t)$ of the $VM(A,t)$ VMs are assigned to different servers, while the rest are all allocated together. $m(t)$ is the number of servers that are already turned on, and have sufficient remaining resources at time t.



(a) $1 \leq VM(A,t)$, lag ≤ 100



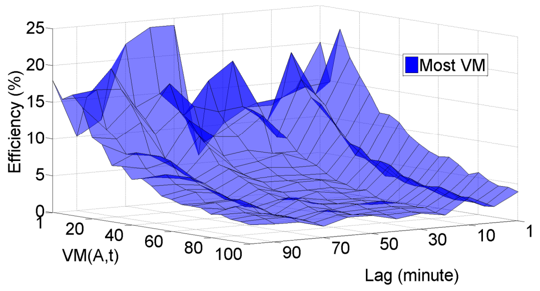
(b) $5 \leq VM(A,t) \leq 100$, $10 \leq$ lag ≤ 100

Fig. 2. The impact of the lag and the number of VMs started by the attacker ($VM(A,t)$) on attack efficiency under the Least VM policy. Fig. 2(a): the overall case, where $1 \leq VM(A,t)$, lag ≤ 100. Fig. 2(b): the stable region, where $5 \leq VM(A,t) \leq 100$, $10 \leq$ lag ≤ 100.

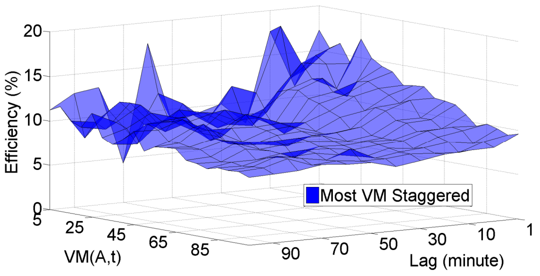
(2) The Most VM policy allocates new VMs to the same server until its remaining resources are less than required. Hence, the efficiency rate is relatively high with small lags, and decreases as the lag increases. However, similar to the situation with the Least VM policy, once the lag is larger than a certain value, the efficiency remains approximately the same.

A clever attacker would learn from the first observation that in order to achieve a higher efficiency with a large $VM(A,t)$, instead of starting all the VMs at the same time, they should start S VMs ($0 < S < VM(A,t)$) at a time, and repeat that $VM(A,t)/S$ times at certain intervals.

We re-ran the experiment with S set to five, and the interval set to one minute. As can be seen from Fig. 3(b), when $VM(A,t)$ is large, the efficiency rate is much higher if the attacker starts their VMs using the staggered approach described above. We use this set of results as the input to our game model.



(a) Starting *VM*(*A*,*t*) VMs at once



(b) Starting *VM*(*A*,*t*) VMs in a staggered way

Fig. 3. The impact of the lag and the number of VMs started by the attacker (*VM*(*A*,*t*)) on attack efficiency under the Most VM policy. Fig. 3(a): starting all *VM*(*A*,*t*) at once. Fig. 3(b): starting *VM*(*A*,*t*) in a staggered way (in batches of *S*).

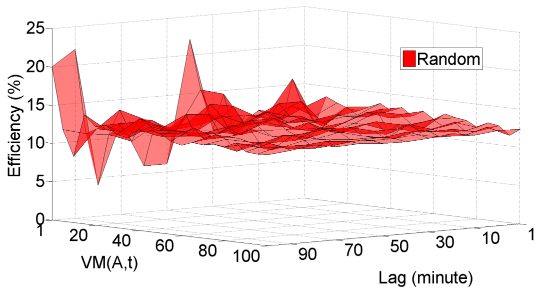
Random Allocation Policy

The attack efficiency under the Random policy is similar to that of Least VM. It stays at almost the same value regardless of the lag and *VM*(*A*,*t*).

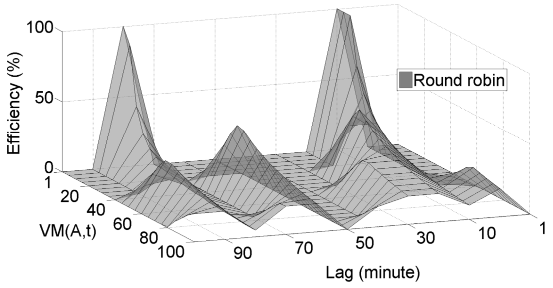
Round Robin Allocation Policy

Under the Round Robin policy, the servers are selected sequentially. As a result, the attacker can only achieve a high efficiency rate if the time when they launch their VMs happens to be close to the time when the target server is chosen. As we can see from Fig. 4(b), there are only a few spikes along the Lag-axis.

However, this is not difficult to implement: because the servers are selected in a fixed order, the attacker can keep starting one VM every a few minutes, and tracking the chosen servers. When they find the target server will be selected again, they can then start their own VMs.



(a) Random



(b) Round Robin

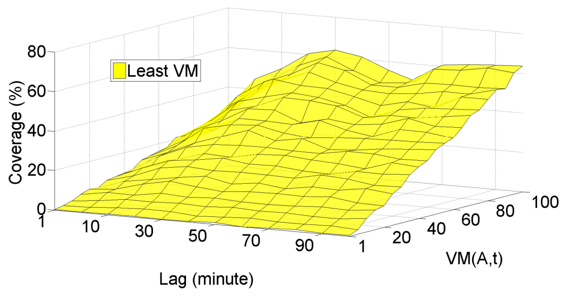
Fig. 4. The impact of the lag and the number of VMs started by the attacker (*VM*(*A*,*t*)) on attack efficiency under the Random and Round Robin VM allocation policies

In other words, due to its deterministic behaviour, the Round Robin policy is the least secure. Therefore, in our game model, we set the attack efficiency under the Round Robin policy to 100%.

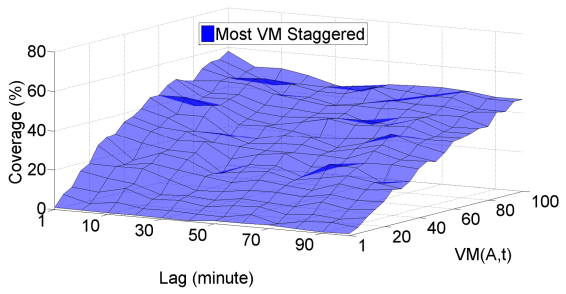
4.3 Coverage Rate under Different VM Allocation Policies

Under the three stochastic policies, the general trend of the coverage rate is similar: it increases almost linearly with *VM*(*A*,*t*), and the lag has little impact after it reaches 10-20 minutes. The only difference is that when the lag is small, the coverage rate under the Most VM policy is much higher than under the Least VM policy.

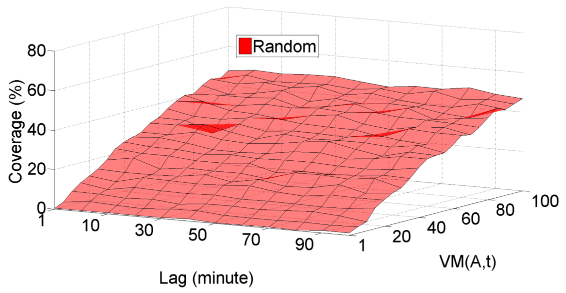
As for the Round Robin policy, the situation is similar to that of the attack efficiency, where the attacker can achieve a high rate periodically. Hence, we also set the coverage to 100% for the Round Robin policy in our game model.



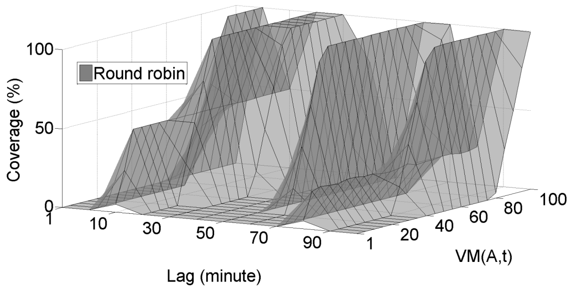
(a) Least VM



(b) Most VM



(c) Random



(d) Round Robin

Fig. 5. The impact of lag and the number of VMs started by the attacker (*VM*(*A*,*t*)) on the coverage rate under the four VM allocation policies

4.4 Power Consumption under Different VM Allocation Policies

When comparing the power consumption under the four policies, we ignore the influence of *VM*(*A*,*t*), because it only contributes a tiny portion of all the VMs in the system. Fig. 6(a) shows the normalised results, with the consumption under the Least VM policy set to 1. We can see that except for the Most VM policy where the value is around 0.5, the power consumption of other policies are essentially the same.

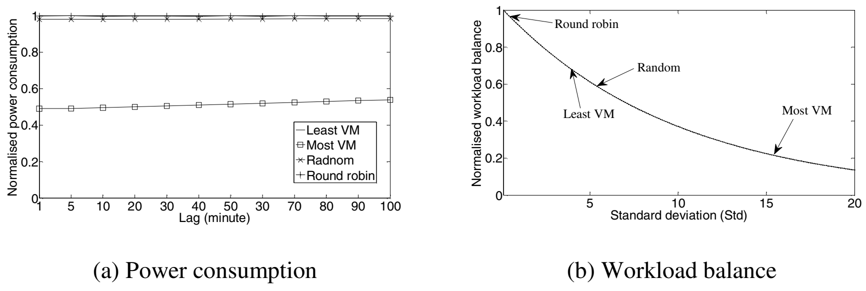


Fig. 6. Normalised power consumption and workload balance under the four VM allocation policies

4.5 Workload Balance under Different VM Allocation Policies

We count the number of times that each server is selected during one experiment, and then calculate the standard deviation (*Std*) to quantify the workload balance under the four policies. Finally, the function $f(Std) = e^{-Std/10}$ is applied to normalise the standard deviation (we acknowledge that there are many other ways for normalisation, and we choose this function as a starting point because it generally reflects the degree of balance under the four policies). As can be seen from Fig. 6(b), the Round Robin policy achieves the best workload balance, while the Most VM policy performs the worst.

4.6 Other Criteria

When comparing the four VM allocation policies, we also considered SLA (service level agreement) related criteria. Here, we use the definition of a SLA violation in [30]: "SLA violation occurs when a VM cannot get amount of MIPS that are requested". The three SLA related criteria below are measured in our experiment: SLA violation time per host, overall SLA violation, and average SLA violation. Our results show that there is no major difference in terms of these criteria between the four policies. Therefore, they are not included in our game model.

Table 3. Definitions of SLA related criteria

Name	Definition
SLA violation time per host (%)	∑ SLA violation time of each host / ∑ Active time of each host
Overall SLA violation (%)	(∑ Requested MIPS - ∑ Allocated MIPS) / ∑ Requested MIPS
Average SLA violation (%)	Only consider the SLA violation incidents, (∑ Requested MIPS - ∑ Allocated MIPS) / ∑ Requested MIPS

4.7 Numerical Solutions and Discussion

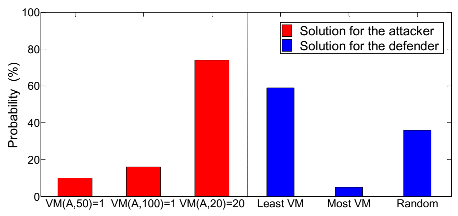
In the previous subsections, we have presented the attack efficiency, coverage, power consumption and workload balance under the four VM allocation policies. These are used to build the game matrices for the attacker and the defender. In this subsection, we compute the numerical solution of the game using Gambit [31], a tool for constructing and analysing finite, non-cooperative games, and interpret the results.

Zero-Sum Game

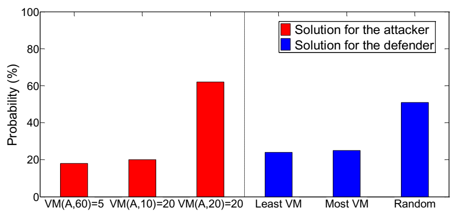
We begin with the simplest scenario where $w_{D_1} = w_{D_2} = 0$, which becomes a zero-sum game. We consider the following two situations: (1) $w_A = 1$, $U^A = Efficiency(VM(A,t),Policy)$, $U^D = -U^A$; (2) $w_A = 0$, $U^A = Coverage(VM(A,t),Policy)$, $U^D = -U^A$.

As can be seen from the following figures, both the solutions are mixed strategies. For the attacker, the solution is straightforward: they should start a small number of VMs each time if they aim to maximise the efficiency, but if the goal is to co-locate with as many target VMs as possible, they should start a large number of VMs at a time. For the defender, the result indicates that instead of deploying a single VM allocation policy, it is better to use a set of policies, and when a VM request arrives, each policy will be selected with a pre-set probability.

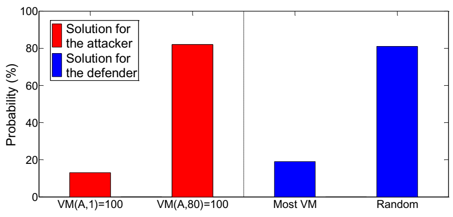
The following points should be noted. (1) As stated in our previous analysis, the Round Robin policy is the least secure, and is selected in neither case. (2) Even though, generally speaking, the attack efficiency under the Most VM policy is the lowest (especially when the lag is larger than five minutes), the peak value in this case is higher than that under the other two policies. This is the reason why the percentage of choosing Most VM is the smallest, if the defender intends to minimise the attack efficiency. However, if we only consider the situation where $VM(A,t) > 1$ and lag > 1 minute (which is closer to the real case), then the Most VM policy contributes a much larger percentage of the solution. (3) Under the Least VM policy, if the attacker starts multiple VMs at the same time, it is very likely that all of these VMs will be allocated to different servers. In contrast, under the other two policies there is a greater chance that some of these VMs will be located on the same server, which has a negative influence on the coverage rate. As a result, if the defender aims to minimise the coverage rate, they should only use the Most VM and Random policies.



(a) $w_A = 1$, $1 \leq VM(A,t)$, lag ≤ 100



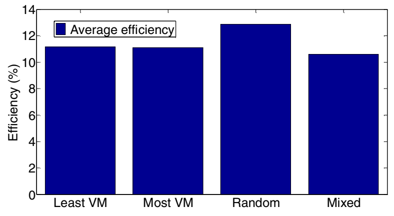
(b) $w_A = 1$, $5 \leq VM(A,t)$, lag ≤ 100



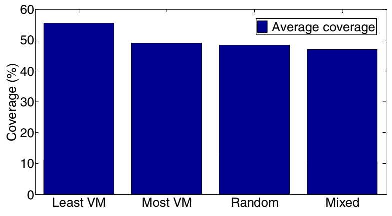
(c) $w_A = 0$, $1 \leq VM(A,t)$, lag ≤ 100

Fig. 7. Nash equilibrium in zero-sum game. Fig. 7(a): in the case where $w_A = 1$, $1 \leq VM(A,t)$, lag ≤ 100, the best strategy for the attacker is to start 1 VM when the lag is 50 minutes, i.e., $VM(A,50)=1$, with a probability of 10%, start 1 VM when the lag is 100 minutes, i.e., $VM(A,100)=1$, with a probability of 16%, and start 20 VMs when the lag is 20 minutes, i.e., $VM(A,20)=20$, with a probability of 74%. The best strategy for the attacker is to choose the Least VM, Most VM, and Random policies with a probability of 59%, 5%, 36%, respectively. The definitions of the symbols in the other two figures are the same.

We re-ran the experiment with the following two sets of configurations: (1) for the attacker, $VM(A,t) = 20$, and the lag ranges from 5 to 100 minutes, while the defender uses the second mixed policy (Least VM, 24%, Most VM, 25%, and Random, 51%); (2) for the attacker, $VM(A,t) = 100$, $1 \leq t \leq 100$, and the defender uses the third mixed policy (Most VM, 19%, and Random, 81%). The result shows that in overall terms, the average efficiency/coverage rate is the lowest under the mixed policies.



(a) Mixed policy 2: minimising the efficiency



(b) Mixed policy 3: minimising the coverage

Fig. 8. Comparison between the mixed policies and the stochastic policies in terms of the efficiency/coverage rate

Non-zero-sum Game

Different policies have their own advantages/disadvantages. For instance, the power consumption under the Most VM policy is the lowest, while the other policies achieve better workload balance. In practice, the defender can adjust the weights of security, power consumption, and workload balance, according to their different requirements.

Here we consider the situation where the three aspects are considered as equally important, i.e., $w_{D_1} = w_{D_2} = 1/3$. From Fig. 9, we can see that the results are similar to those shown in Fig. 7, which further demonstrates that a mixed policy may outperform any single policy.

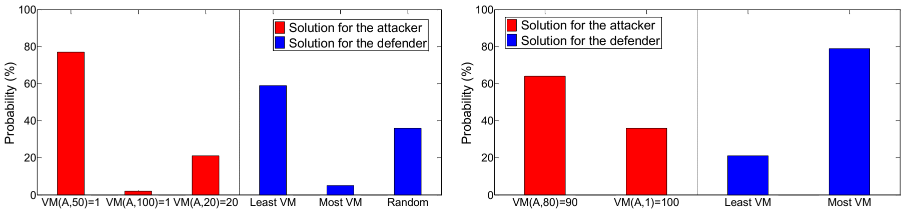
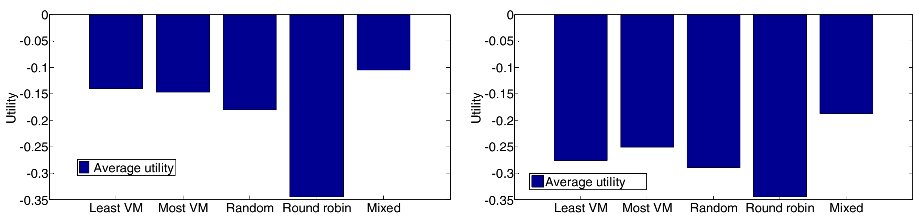


Fig. 9. Nash equilibrium in non-zero-sum game (all the definitions of symbols are the same as those in Fig. 7)

Similarly, we re-ran the experiment with the following configurations to compare the mixed policy with the four pure policies: (1) for the attacker, $VM(A,t) = 1$, and $1 \leq t \leq 100$, while the defender uses the first mixed policy (Least VM, 59%, Most VM, 5%, and Random, 36%); (2) for the attacker, $VM(A,t) = 90$, $1 \leq t \leq 100$, and the defender uses the second mixed policy (Least VM, 21%, and Most VM, 79%).

However, in this case, the defender cannot simply mix the policies with the specified percentages. Otherwise, on the one hand, an excessive number of servers will be turned on because the mixed policy contains Least VM and Random policies. On the other hand, the workload will not be balanced due to the Most VM policy. In other words, the mixed policy integrates the disadvantages instead of the advantages of each policy.

Therefore, we make the following changes and the allocation process comprises two rounds. In the first round, only the servers that are already being used and have sufficient remaining resources will be considered. If such a kind of server does not exist, then in the second round all servers are taken into consideration. In both rounds, each policy is still selected with the specified probability. As we can see from Fig. 10, the defender's utility is highest under mixed policies in both cases.



(a) First mixed policy (b) Second mixed policy

Fig. 10. Comparison between the mixed policies and the pure policies in terms of the defender's utility. Fig. 10(a): first mixed policy (Least VM, 59%, Most VM, 5%, and Random, 36%). Fig 10(b): second mixed policy (Least VM, 21%, and Most VM, 79%).

5 Conclusion and Future Work

In this paper, we introduce a game theoretic approach to compare four basic VM allocation policies for cloud computing systems, and propose a practical method for mitigating the threat of the co-resident attack. Our results show that in order to minimise the efficiency and coverage rates for the attacker, the cloud provider should use a policy pool, such that for each VM request, a policy is chosen at random from the pool according to their predefined probabilities.

In the future, we intend to test our findings in larger scale systems. In addition, we will also study what the differences are between the behaviours of the attacker and normal users under the mixed policy, and how to identify them.

References

1. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the Art of Virtualization. Operating Systems Review 37, 164–177 (2003)
2. Ristenpart, T., Tromer, E., Shacham, H., Savage, S.: Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds. In: 16th ACM Conference on Computer and Communications Security, CCS 2009, pp. 199–212 (2009)
3. Zhang, Y., Juels, A., Reiter, M., Ristenpart, T.: Cross-VM Side Channels and Their Use to Extract Private Keys. In: 2012 ACM Conference on Computer and Communications Security, CCS 2012, pp. 305–316 (2012)
4. Aviram, A., Hu, S., Ford, B., Gummadi, R.: Determinating Timing Channels in Compute Clouds. In: 2010 ACM Workshop on Cloud Computing Security Workshop, CCSW 2010, pp. 103–108 (2010)
5. Vattikonda, B., Das, S., Shacham, H.: Eliminating Fine Grained Timers in Xen. In: 3rd ACM Workshop on Cloud Computing Security Workshop, CCSW 2011, pp. 41–46 (2011)
6. Wu, J., Ding, L., Lin, Y., Min Allah, N., Wang, Y.: XenPump: A New Method to Mitigate Timing Channel in Cloud Computing. In: 2012 IEEE Fifth International Conference on Cloud Computing, pp. 678–685 (2012)

7. Shi, J., Shi, J., Song, X., Chen, H., Zang, B.: Limiting Cache-based Side-channel in Multi-tenant Cloud using Dynamic Page Coloring. In: 2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W), pp. 194–199 (2011)
8. Jin, S., Ahn, J., Cha, S., Huh, J.: Architectural Support for Secure Virtualization under a Vulnerable Hypervisor. In: 44th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-44 2011, pp. 272–283 (2011)
9. Szefer, J., Keller, E., Lee, R., Rexford, J.: Eliminating the Hypervisor Attack Surface for a More Secure Cloud. In: 18th ACM Conference on Computer and Communications Security, CCS 2011, pp. 401–412 (2011)
10. Osvik, D.A., Shamir, A., Tromer, E.: Cache Attacks and Countermeasures: The case of AES. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 1–20. Springer, Heidelberg (2006)
11. Tromer, E., Osvik, D.A., Shamir, A.: Efficient Cache Attacks on AES, and Countermeasures. Journal of Cryptology 23, 37–71 (2010)
12. Hlavacs, H., Treutner, T., Gelas, J.-P., Lefevre, L., Orgerie, A.-C.: Energy Consumption Side-Channel Attack at Virtual Machines in a Cloud. In: 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing, pp. 605–612 (2011)
13. Xu, Y., Bailey, M., Jahanian, F., Joshi, K., Hiltunen, M., Schlichting, R.: An Exploration of L2 Cache Covert Channels in Virtualized Environments. In: 3rd ACM Workshop on Cloud Computing Security, CCSW 2011, pp. 29–39 (2011)
14. Okamura, K., Okamura, K., Oyama, Y.: Load-based Covert Channels between Xen Virtual Machines. In: 2010 ACM Symposium on Applied Computing, SAC 2010, pp. 173–180 (2010)
15. Wu, J., Ding, L., Wang, Y., Han, W.: Identification and Evaluation of Sharing Memory Covert Timing Channel in Xen Virtual Machines. In: 2011 IEEE 4th International Conference on Cloud Computing, pp. 283–291 (2011)
16. Kadloor, S., Kadloor, S., Kiyavash, N., Venkitasubramaniam, P.: Scheduling with Privacy Constraints. In: 2012 IEEE Information Theory Workshop, pp. 40–44 (2012)
17. Xia, Y., Yetian, X., Xiaochao, Z., Lihong, Y., Li, P., Jianhua, L.: Constructing the On/Off Covert Channel on Xen. In: 2012 Eighth International Conference on Computational Intelligence and Security, pp. 568–572 (2012)
18. Bedi, H., Shiva, S.: Securing Cloud Infrastructure Against Co-Resident DoS Attacks Using Game Theoretic Defense Mechanisms. In: International Conference on Advances in Computing, Communications and Informatics, ICACCI 2012, pp. 463–469 (2012)
19. Varadarajan, V., Kooburat, T., Farley, B., Ristenpart, T., Swift, M.: Resource-Freeing Attacks: Improve Your Cloud Performance (at Your Neighbor's Expense). In: 2012 ACM Conference on Computer and Communications Security, CCS 2012, pp. 281–292 (2012)
20. Yang, Z., Yang, Z., Fang, H., Wu, Y., Li, C., Zhao, B., Huang, H.H.: Understanding the Effects of Hypervisor I/O Scheduling for Virtual Machine Performance Interference. In: 4th IEEE International Conference on Cloud Computing Technology and Science, pp. 34–41 (2012)
21. Zhou, F.F., Goel, M., Desnoyers, P., Sundaram, R.: Scheduler Vulnerabilities and Coordinated Attacks in Cloud Computing. In: 10th IEEE International Symposium on Network Computing and Applications, NCA (2011)
22. Zhang, Y., Li, M., Bai, K., Yu, M., Zang, W.: Incentive Compatible Moving Target Defense against VM-Colocation Attacks in Clouds. In: Gritzalis, D., Furnell, S., Theoharidou, M. (eds.) Information Security and Privacy Research, vol. 376, pp. 388–399. Springer, Heidelberg (2012)

23. Li, M.: Improving cloud survivability through dependency based virtual machine placement. In: The International Conference on Security and Cryptography, SECRYPT 2012, pp. 321–326 (2012)
24. Alpcan, T., Baar, T.: Network Security: A Decision and Game-Theoretic Approach. Cambridge University Press (2010)
25. CloudSim, `http://www.cloudbus.org/cloudsim/`
26. Calheiros, R., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R.: CloudSim: a Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. Software, Practice and Experience 41, 23–50 (2011)
27. Beloglazov, A., Abawajy, J., Buyya, R.: Energy-aware Resource Allocation Heuristics for Efficient Management of Data Centers for Cloud Computing. Future Generation Computer Systems 28, 755–768 (2012)
28. Han, Y., Chan, J., Leckie, C.: Analysing Virtual Machine Usage in Cloud Computing. In: IEEE 2013 3rd International Workshop on Performance Aspects of Cloud and Service Virtualization, CloudPerf 2013 (to appear, 2013)
29. Synthetic self-similar traffic generation, `http://glenkramer.com/ucdavis/trf_research.html`
30. Buyya, R., Beloglazov, A., Abawajy, J.: Energy-Efficient Management of Data Center Resources for Cloud Computing: A Vision, Architectural Elements, and Open Challenges. In: 2010 International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA 2010 (2010)
31. Gambit: Software Tools for Game Theory, `http://www.gambit-project.org/gambit13/index.html`