Sit Here: Placing Virtual Machines Securely in Cloud Environments

Mansour Aldawood[1], Arshad Jhumka[1] and Suhaib A. Fahmy[2,3]
[1]*Department of Computer Science, University of Warwick, Coventry, U.K.*
[2]*King Abdullah University of Science and Technology (KAUST), Thuwal, Saudi Arabia*
[3]*School of Engineering, University of Warwick, Coventry, U.K.*

Keywords: Cloud Computing, Side-channel Attacks, Virtual Machine Scheduling.

Abstract: A Cloud Computing Environment (CCE) leverages the advantages offered by virtualisation to enable virtual machines (VMs) within the same physical machine (PM) to share physical resources. Cloud service providers (CSPs) accommodate the fluctuating resource demands of cloud users dynamically, through elastic resource provisioning. CSPs use VM allocation techniques such as VM placement and VM migration to optimise the use of shared physical resources in the CCE. However, these techniques are exposed to potential security threats that can lead to the problem of *malicious co-residency* between VMs. This threat happens when a malicious VM is co-located with a critical (or target) VM on the same PM. Hence, the VM allocation techniques need to be made secure. While earlier works propose specific solutions to address this malicious co-residency problem, our work here proposes to investigate the *allocation patterns* that are more likely to lead to a secure allocation. Furthermore, we introduce a security-aware VM allocation algorithm (SRS) that aims to allocate the VMs securely, to reduce the potential for co-residency between malicious and target VMs. Our study shows: (i) our SRS algorithm outperforms all state-of-the-art allocation algorithms and (ii) algorithms that adopt stacking-based behaviours are more likely to return secure allocations than those with spreading or random behaviours.

1 INTRODUCTION

Cloud computing environments (CCEs) are enabling the deployment of a broad range of services such as web applications, enterprise systems or large-scale analytics applications, among others. They enable the abstraction, pooling, and scalable sharing of computing resources (e.g., CPU, storage), accessible across a network, by a pool of users. Virtualization is a technique that enables these CCEs to dynamically provide an extensive distributed computing resource. Users access these physical resources, hosted on physical machines (PMs), through virtual machines (VMs). As the resource requirements of an executing workload fluctuate, physical resources can be dynamically allocated to or reclaimed from the application. In other words, the provisioning of resources is elastic, based on user requirements (Hu et al., 2009).

Resource allocation is also more flexible and requires less time and management than traditional methods (Zhang et al., 2010). Such CCEs will enable multiple users to share a common computing platform where resources are dynamically available.This invariably means that a PM can potentially share its resources among a set of distinct users (or VMs)[1], in what is known as *VM co-location*. As a consequence of VMs co-location, rather than VMs having dedicated resources, the security threats for these new computing environments have invariably shifted. The types of threats that arise when a malicious VM shares with a (target) VM range from data confidentiality breach to denial-of-service attacks (Jansen, 2011). Thus, VMs co-location, though enabling efficient resource sharing, is creating unwanted side channels, which can be sources of potential side-channel attacks (SCAs), such as cache-based SCAs, timing SCAs among many others. Informally, side channels are (unwanted) communication channels between processes that may leak sensitive outputs from a process (Zhou and Feng, 2005). SCAs will have impact that can extend from application level to the hardware level (Bazm et al., 2017; Spreitzer et al., 2017) and will become more prevalent due to the range of side channels that exists.

When VMs are co-resident (or co-located) on the

[1]We will henceforth use the term users and VMs interchangeably.

Aldawood, M., Jhumka, A. and Fahmy, S.
Sit Here: Placing Virtual Machines Securely in Cloud Environments.
DOI: 10.5220/0010459202480259
In *Proceedings of the 11th International Conference on Cloud Computing and Services Science (CLOSER 2021)*, pages 248-259
ISBN: 978-989-758-510-4

same PM, one (malicious) VM can analyse characteristics of another (target) VM, e.g., the timing properties, to infer various information such as cryptographic keys. Thus, it is crucial that malicious VMs, i.e., those wishing to steal information, and target VMs, i.e., those with sensitive information, are not co-resident on the same PM. There are two major steps involved to overcome this problem: (i) identifying the malicious and target VMs and (ii) keeping the malicious and target VMs apart. In this paper, we focus on the second problem. For completeness, we briefly explore work that focused on the first problem.

The analysis of VM behaviour is crucial for CSPs to be able to identify VMs with suspicious behaviour and isolate them from other VMs. In (Han et al., 2015a), the authors present a model for analysing the behaviour of VMs by monitoring specific factors that help categorise VMs into specific classes. These are: (i) a user launching a small or large number of VMs at a particular time, (ii) or at a periodical time, (iii) keeping at least one VM active at all times or (iv) all of a user's VMs consuming minimal active time to save cost. After monitoring these factors, a CSP can classify VMs as either high, medium or low-risk. Furthermore, we will assume that a CSP analyses the behaviours of VMs, to identify suspicious VMs and to allocate VMs according to this analysis. The analysis could be performed initially by merely asking VMs users to submit their list of security constraints if they are aware of the security threats on the cloud systems. Alternatively, analyses can be performed on the VMs, based on the information gathered from the VMs users about the type of applications or data processed on their VMs. This step could help to identify the level of data sensitivity initially. After the initial analysis, the CSP can categorise the VMs and subsequently start the allocation process. Another technique is to perform the analysis during VM execution, to capture their activities and possible suspicious behaviour. The result of this analysis could lead to a possible re-allocation of the VMs. It is worth mentioning that suspicious VMs are not necessarily malicious ones. However, their suspicious behaviours may lead the CSP to categorise them as a high-risk. The CSP should handle these VMs from a security perspective and perform allocations according to the result of the categorisation while meeting hosting requirements.

Focusing on the allocation problem, the architecture we assume in this paper is presented in Figure 1: Given (i) a set of VMs, partitioned into malicious VMs, target VMs, and normal VMs, (ii) a set of PMs with varying physical resources and (iii) a set of new VMs coming into the system, we develop (i) an allocation algorithm and (ii) a migration algorithm to

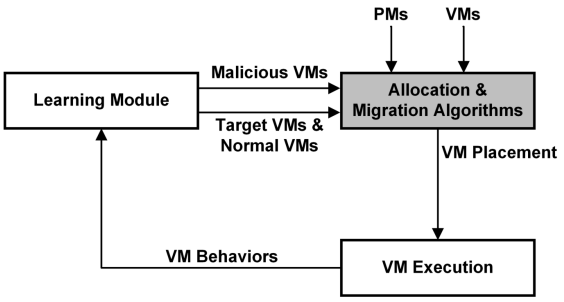


Figure 1: Cloud architecture assumed in this paper.

ensure secure VM allocation, i.e., malicious VMs are not co-resident with target VMs. We assume that the behaviours of VMs are monitored and analysed to identify malicious VMs. In this paper, we assume the learning component is predefined, and accordingly, we classify several VMs as malicious, target and normal.

We run extensive simulations of our algorithms under a variety of scenarios and parameters. We further investigate the effect of three algorithms, namely (i) Round Robin, (ii) Random and (iii) previously selected servers first (PSSF) algorithms. Each of these algorithms have unique allocation behaviours. We consider three VM allocation behaviours: (i) stacking, (ii) spreading and (iii) Random. The stacking behaviour captures the fact that an algorithm will try to use as few PMs as possible to reach a suitable allocation, e.g., a Bin-Packing algorithm. The spreading behaviour means that the allocation algorithm will allocate VMs to the whole span of PMs, e.g., round robin. The random allocation algorithm will allocate the VMs randomly as long as the candidate PM is suitable. The algorithms we develop in this paper is *stacking-based*, similar to bin-packing. However, binbaking not classified as a secure-aware algorithm, for example, the first-fit, is a heuristic of bin-packing, that require modification to be a secure-aware allocation algorithm (Natu and Duong, 2017). Thus, our proposed algorithm is an enhancement of Bin-packing, that aims to obtain a secure VM allocation. The objective of studying the proposed algorithms is to investigate the allocation behaviour that leads to a secure allocation. It is worth mentioning that Random and Round Robin allocation are not secure by design. However, for this study, we have modified them to be security-aware by integrating them with the leaning component in Figure 1.

The main contributions for this paper are as follows:

1. We propose a secure stacking-based algorithm (SRS) that securely allocates the VMs to reduce the effect of malicious co-residency.

2. We study the effect of VM allocation behaviour on obtaining a secure allocation. The behaviours are stacking, spreading and random behaviour.
3. We investigate the factors affecting the outcome towards obtaining a secure allocation. These are: (i) the PMs heterogeneity level and (ii) the diversity of available resources and (iii) the VMs arrival time for each type of VMs considered in this work.
4. We show that our algorithm SRS outperforms all state-of-the-art allocation schemes, even after being transformed for security.

The paper is structured as follows: We present a survey of the literature in Section 2 and related work to our paper in Section 3. We present our system model and a formalisation of the problem we focus on in Section 4. We develop our algorithms in Section 5, and we present their performance evaluation in Section 6. We conclude the paper in Section 7.

2 LITERATURE REVIEW

Over the past years, defending SCA focused on securing the allocation and migration process of the VMs. Other areas, concentrated on applying modifications to the PMs level to obtain secure isolation.

VM Clustering. The first domain focuses on grouping the VMs based on defined requirements through the allocation. (Han et al., 2017) proposed isolated zones to separate the VMs based on their risk level by considering specific security dimensions such as VMs, PMs or hypervisor risk. Then, allocates the VMs according to the obtained risk evaluation. Moreover, (Liang et al., 2017) proposed a 2-step algorithm: group and host selection. Their goal is to create randomness in the way of VM allocation to a PM. Also, the work in (Natu and Duong, 2017) proposed an algorithm that allocates the VMs based on a user security profile defined by the user. In addition, (Yuchi and Shetty, 2015) proposed an allocation algorithm that depends on the risk score obtained from US national vulnerability database (NVD). Furthermore, (Bijon et al., 2015) proposed a framework for allocating the VMs under conflict-free groups by ensuring that each group shares the same attribute value of security conflict. Another work introduced a framework for VM allocation that aims to group VMs based on constraint requirements (Al-Haj et al., 2013). Additionally, the work of (Li et al., 2012) introduced a mechanism to identify the dependency between VMs using the network connection information and to allocate the VMs accordingly. Moreover, (Afoulki et al., 2011) proposed an algorithm that allocated the VMs based on an adversary list submitted by the user.

Security Compliance. The second domain is considering another form of grouping, which is to allocate the VMs based security compliance requirements to ensure each VM hosted on the same PM share the same security compliance level. (Bahrami et al., 2017) introduced a VM allocation algorithm that adopted the Health Insurance Portability and Accountability Act (HIPA). Furthermore, (Ahamed et al., 2016) proposed a VMs migration algorithm that ensures the VMs hosted on the same PM share the same security level.

Time-triggered Allocation. The third domain is focusing on allocation the VMs for a defined time to reduce the amount of sensitive data leakage through the SCA. (Sun et al., 2016) proposed an information leakage model that measures the duration of co-located VMs without triggering a suspicious behavior. If VMs co-located for specific time without triggering intrusion detection, then these VMs considered friendly. Moreover, (Moon et al., 2015) introduced a placement algorithm to reduce the information leakage by placing VMs based on the amount of time these VMs hosted previously together without triggering a malicious behavior.

Co-residency Mitigation. The fourth domain focuses on reducing the chance of VMs co-residency with malicious VMs. (Berrima et al., 2016) introduced an algorithm that deliberately delay the start-up time for VMs to reduce the chance of co-residency with malicious VMs. Furthermore, (Zhang et al., 2012) proposed a model that stimulates the users to migrate their VMs to avoid malicious co-residency threats. The idea is to migrate the VMs frequently to reduce the chance of co-residency. (Qiu et al., 2017) developed an algorithm that considers the amount of VMs and PMs used during the allocation to avoid malicious co-residency. Moreover, (Agarwal and Duong, 2019) introduced an algorithm that aims to reduce the co-residency by allocating VMs with ones which shared the same PMs previously. Additionally, (Ding et al., 2018) proposed an optimization-driven solution, based on the firefly algorithm, that aims to find the optimal allocation which reduces the chance of malicious co-residency. Also, (Han et al., 2018) presented a secure VM allocation algorithm based on multi-objective optimization. It is a heuristic of the First-Fit algorithm, which provides an allocation that satisfies the resource constraint. Furthermore, (Wong and Shen, 2018) proposed an enhancement of the Best-Fit algorithm by allocating the VMs based on the level of familiarity between the VMs and hosted PMs.

Secure Hardware. The fifth domain is focusing on the secure allocation of VMs by securing the hardware level of the PMs. (Sprabery et al., 2017) proposed a solution that requires a change in the hardware to eliminate the side channel threats. Their framework aims to partition the shared cache into two separate regions called shared cache and isolated cache. The shared cache, will execute all shared resources. However, the isolated cache, will only execute the resources from VMs that require a secure and isolated processing unit. Additionally, (Will and Ko, 2017) Introduced a secure data processing component using FPGA (field-programmable gate array). FPGAs is reprogrammable to specific functionality, and they been used as an accelerator for the hardware processing, such as on Amazon cloud.

3 BACKGROUND

The work in (Han et al., 2015b) introduced a VM allocation policy called previously selected servers first (PSSF). This policy is allocating the VMs belonging to a user on the same PM, as long as the number of VMs for the same user not exceeding a certain threshold, they assume its three VMs per one PM for the same user. If there is no previous VMs, it will use the existing allocation policy to place the VM based on available resources. They discovered that the PSSF works better when combined with the least VM allocation policy. However, their work did not consider VMs migration and its effect on the allocation process. Also, they consider the VMs arrival only in one situation, where the target VMs are allocated first, and then the malicious ones allocated afterwards. Other scenarios could be investigated and studied on different VMs arrivals time. Also, they did not consider the possibility of the presence of normal VM users. The normal VM users could potentially affect the allocation as the consumption of the resources will be higher, and the allocation of the VMs to the available resources will be challenging. Also, their work did not consider the PMs heterogeneity, meaning the effect of the allocation on PMs with diverse available resources. Finally, their work tries to produce the allocation based on the malicious VM perspective, not the CSPs perspective. In our work, we will try to fill this gap by considering the VM migration on the allocation process. We introduce three types of VMs users Target, Normal and Malicious users to study their effect on the allocation outcome. Moreover, we consider different VMs arrival times to show each VM type arrivals' effect on allocation security. Our work also presents a solution from the CSPs perspective, which is to obtain the most secure allocation under different scenarios for all users.

4 SYSTEM MODEL AND PROBLEM STATEMENT

We present the system model we assume in this paper and subsequently present the problem we address.

4.1 System Model

The system consists of a set P of $k+1$ physical machines, labelled $PM_0, PM_1, \ldots$ which remains unchanged in the lifetime of the allocation process. Each $PM_i, 1 \leq i \leq k$ has the same set of resources but in varying quantities, e.g., one *PM* may have more storage than another, i.e., we assume the system to be heterogeneous. We assume the existence of a "null" *PM* ($\perp$) that keeps all the unallocated VMs in the system and corresponds to PM_0. We denote by $R(PM_i)$, the amount of physical resources available on PM_i. On the other hand, there is a set V of virtual machines, labelled $VM^1, \ldots$. Each VM^j has the same set of resource type requirements but in varying amounts and we assume that all the resource needs of a *VM* can be met by any *PM*[2]. We denote by $N(VM^j)$, the amount of resources needed by VM^j.

The set V is partitioned into three sets: (i) set T of target VMs, (ii) set M of malicious VMs and (iii) set N of normal VMs, with the following constraints:

- $V = T \cup M \cup N$
- $T \cap M = \emptyset \wedge T \cap N = \emptyset \wedge M \cap N = \emptyset$

A target *VM* is a *VM* that has proven to be legitimate, which means this *VM* has sensitive data that could be compromised by other VMs. The target *VM* is classified as a critical *VM* before and during the *VM* allocation by the CSP, according to their *VM* behaviour analysis. Similarly, based on its behaviour, a *VM* can be classed as a malicious *VM* that behaves suspiciously, according to the CSP behaviour analysis. If a *VM* is behaving suspiciously, then it is considered a malicious *VM*, until it is proven otherwise. If the *VM* is considered malicious, then it is considered a risk to the target VMs. A normal *VM* is a *VM* that is neither a target nor a malicious *VM*.

We model the *VM* allocation as a function $A : V \rightarrow P$, i.e., an allocation is an assignment of VMs to PMs. An unallocated *VM* v is one such that $A(v) = \perp$ (allocated to PM_0), otherwise it is allocated. A *VM*

[2]We assume the system to be heterogeneous in terms of resource availability.

can be allocated to a *PM* if the resources available at the *PM*, meaning the *PM* can meet the resource requirements of the *VM*, i.e., if VM^i is allocated to $PM_j, j \neq 0$, then PM_j can satisfy the resource requirements of VM^i, i.e., $A(VM^i) = PM_j, j \neq 0 \Rightarrow R(PM_j) \geq N(VM^j)$. The system tries to allocate any unallocated VM $VM^u \in PM_0$ to some $PM_j, j > 0$.

An allocation space $\mathcal{A}$ is the set of all possible allocations. We can then view the allocation system as a transition system $(\mathcal{A}, I, \mathcal{M})$, with I being the set of all possible initial allocations and $\mathcal{M}$ being the set of transitions between allocations. The transition from an allocation A_i to an allocation A_j is called a *migration*. The initial allocation, where all VMs are on PM_0, is always secure. We assume migrations are atomic, i.e., all *VM* movements take place at the same time. A system execution is an infinite sequence of allocations $A_0 \cdot A_1 \ldots$, where $A_0 \in I$ and $(A_i, A_{i+1}) \in \mathcal{M}$. If the sequence is finite, it can be made infinite by infinitely repeating the final allocation. The set of VMs that are migrated during a migration (A_i, A_{i+1}) in an execution is given by

$$Move(A_i, A_{i+1}) = \{v | A_i(v) \neq A_{i+1}(v)\}$$

We say an allocation A is *secure* if $\forall m \in M, \forall t \in T : A(m) \neq A(t) \neq \perp$, i.e., an allocation is secure if no malicious *VM* is co-located with a target *VM*. An allocation that is not secure is termed as a *co-resident* allocation. The set of PMs at which co-residency occurs is denoted by

$$CoRe(A) = \{p \in P \setminus \{PM_0\} | A(t) = A(m) = p, t \in T, m \in M\}$$

We say that a migration is secure if both the start and the end allocations are secure. Whenever there are unallocated VMs in the system, *VM* migrations will occur and the number of migrations must be kept to a minimum to reduce downtime of allocated VMs, i.e., $Move(A_i, A_{i+1})$ needs to be minimized.

4.2 Problem Formalization

One variant of the problem we study can be expressed as follows:

Definition 4.1 (VM Migration with no co-residency)**.** Given a set P of PMs, a set U of (unallocated) VMs, a set L of allocated VMs, a secure allocation A_i, obtain a secure allocation A_{i+1} such that (i) all VMs $v \in U$ are allocated and (ii) $Move(A_i, A_{i+1})$ is minimized.

However, as can be inferred, this cannot be guaranteed at all times, especially when resources are scarce. So, we present a second weaker variant, which we focus on in this paper.

Definition 4.2 (VM Migration with minimal co-residency)**.** Given a set P of PMs, a set U of (unallocated) VMs, a set M of allocated VMs, an allocation A_i, obtain an allocation A_{i+1} such that (i) all VMs $v \in U$ are allocated and (ii) $Move(A_i, A_{i+1})$ is minimised and (iii) the number of PMs at which co-residency occurs is minimized, i.e., minimise $CoRe(A_{i+1})$.

Denoting the minimum number of PMs by p, an allocation A that satisfies Definition 4.2 is said to be p-secure. An allocation A that is secure is thus 0-secure[3].

5 ALGORITHM FOR SECURE ALLOCATION

The main objective of this paper is to address the secure VM placement, i.e., develop a secure allocation algorithm while minimising the VM migrations and the number of PMs where co-residency occurs (i.e., Definition 4.2). In addition, we will endeavour to keep the number of PMs used as small as possible. Using a stacking-based algorithm (e.g., bin-packing) will allow the use of a small number of PMs. However, such algorithms are not known to be secure.

As such, we propose our security-aware heuristic, a variant of bin-packing, called Secure Random Stacking (SRS), which is shown in Algorithm 1. Algorithm 1 (SRS) allocates VMs randomly in a stacking fashion and migrates them from one PM to another if the possibility of VM migration exists. Similar to a bin-packing algorithm, e.g., (Korf, 2002), the SRS algorithm aims to allocate the VMs into the selected PMs while using a smaller number of available PMs and to maintain a secure allocation. The following sections will describe the SRS allocation algorithm and associated functions in detail.

5.1 The Process of SRS Algorithm

The general idea of SRS is to return as many secure allocations as possible within a given time limit, then checks for the malicious co-residency for these allocations (p-secure). If malicious co-residency reaches the minimum level, then this allocation is considered a final allocation. If the malicious co-residency does not reach the minimum level, then another allocation will be generated until a time limit is reached. In our case, the time is set to 2000ms, the algorithm will terminate after this time reached, and the allocation with lowest malicious co-residency is selected as the final

[3]Henceforth, whenever we say secure, we mean p-secure.

Algorithm 1: Secure Random Stacking (SRS).

```
   Input:
   V: Set of unallocated VMs
   P: Set of PMs in the cloud
   Output: mSa: Most Secure Allocation
1  Function vmMigration(electedPMs,P):
2      vmsToMigrateList = {}
3      migrationAllocationList = {}
4      for pm in electedPMs do
5          vmsToMigrateList.add(getMaliciousVms() ∪
             getNormalVms())
6      end
7      for vm_i in vmsToMigrateList do
8          pm_m ← getRandomPM(getHighestFRPMs(vm_i,P))
9          if coResidencyCheck(vm_i, pm_m)≠ True then
10             migrationAllocationList.add(Assign(vm_i, pm_m))
11         end
12     end
13     return migrationAllocationList
14 Function oneAllocation(V,P):
15     oneAllocationList = {}
16     ElectedPMs = {}
17     do
18         ElectedPMs.add(getHighestFRPMs(VM_i,P))
19         PM_j ← getRandomPM(ElectedPMs)
20         if (coResidencyCheck(VM_i, PM_j) ≠ True) then
               // Assign VM to the selected PM
21             oneAllocationList.add(Assign(VM_i, PM_j))
22         end
23         else
24             oneAllocationList.add(vmMigration(ElectedPMs,P)) if
                 (coResidencyCheck(VM_i, PM_j) ≠ True) then
25                 oneAllocationList.add(Assign(VM_i, PM_j))
26             end
27         end
           // If assigning the VM to the elected PMs after
              the migration failed
28         if Assign(VM_i, PM_j) = Null then
29             for PM_k in P do
30                 if (isPMsuitable(VM_i, PM_k)=True) then
31                     oneAllocationList.add(Assign(VM_i, PM_k))
32                 end
33             end
34         end
35     while V ≠ ∅
36     return oneAllocationList
37 allAllocationList = {}
38 oneAllocation = Null
39 oneAllocation ← oneAllocation(V,P)
40 i= getInfectedPMsNumber(oneAllocation)
41 if i ≠ 0 then
42     do
43         allAllocationList.add(oneAllocation)
44         oneAllocation ← oneAllocation(V,P)
45         i= getInfectedPMsNumber(oneAllocation)
46         allAllocationList.add(oneAllocation)
47     while i ≠ 0 ∪ simulationTime ≤ 2000ms
48     mSa ← getLowestInfectedAllocation(allAllocationList)
49 end
50 else
51     mSa ← oneAllocation
52 end
53 return mSa
```

allocation. The time threshold and the minimum level of co-residency can be adjusted based on the performance requirements for the allocation.

The SRS algorithm has two main inputs: (i) the unallocated set of VMs, denoted as V and (ii) the set of the available physical machines, denoted as P. The output, denoted as the *mSa*, is the most secure allocation found for the available set of resources. The SRS algorithm start, at line 39, when the first allocation produced using the *one allocation function*. The details of the *one allocation* and *VM migration* functions will be described in detail in later sections. Afterwards, this first allocation is passed to a function, at line 40, to calculate the number of PMs used in this allocation to compute malicious co-residency. If there is no co-residency (0-secure), the allocation will be selected and considered the most secure allocation for the given resources. Otherwise, the produced allocation will be saved for later comparison with other produced allocations. The algorithm, at this stage, cannot determine if this allocation is the best or worst secure allocation unless it compares the produced allocation with other allocations. Therefore, after this step, another allocation is produced and the previous steps repeated, until one of two stopping conditions applies: (i) the algorithm is able to produce an allocation with no co-residency or (ii) the algorithm reaches a timeout limit. We use a 2000*ms* limit for our algorithm because it produced the best performance trade-off for the SRS algorithm. The time limit is adjustable based on the requirements from the used algorithm, and other constraints on the CSP. Otherwise, if the time limit is reached and the minimum co-residency is greater than zero, the algorithm will compare the existing produced allocations and select the one with lowest co-residency, at line 48.

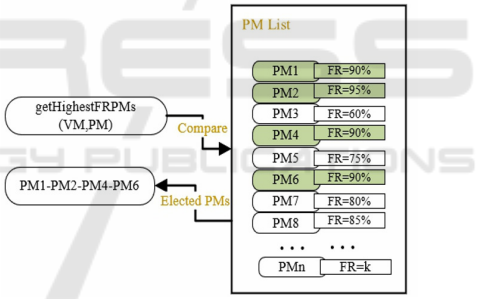


Figure 2: The Fullness Ratio (FR) function.

5.2 The One Allocation Function

The *one allocation function* is responsible for many roles which are: allocate all the unallocated VMs into PMs while maintain the secure allocation constraints, reducing the number of used PMs and reducing the VM migrations. The key factor here is the Fullness Ratio (FR) function, at line 18 denoted as **getHighestFRPMs**. This function allows only the PMs that have resources with a high FR compared to the VM demanded resources to be selected as elected PMs for the allocation. In other words, what are the PMs that if selected, will be filled (FR %) in a way this fullness will be high FR? For example, in figure 2, The PM1, PM2, PM4 and PM6 are the elected PMs for two reasons. Firstly, they have high FR among the available PMs. Secondly, they are the PMs that repre-

sent the two highest FR% among the other PMs FR%. The PM8 still has high FR but not selected as the algorithm will only choose the two highest FR%. The reason for selecting the two highest FR% PMs is to allow more elected PMs to be available. Increasing this number, higher than two, could potentially lead to changing the stacking behavior of the SRS algorithm. The motivation for the FR step is to keep the VMs stacked while selecting the PMs, which leads to a perfect match between the VM and PM selection in the matter of resources. Thus, reducing the number of used PMs during the allocation, which results in allowing more space for incoming VMs to allocate securely. The score of the FR depends on the current situation of the available PM resources, and the VM required resources and the time that VM arrives. Here we calculate the FR based on the RAM resources for the VM and the PMs. The next step of SRS is to select one of the elected PMs as a candidate for allocating the VM_i, at line 19. It starts by selecting the PM_j randomly among the elected PMs and assigns it as a candidate for the next step. The reason for the random selection is to keep predicting the behaviour of the allocation process hard for the malicious user to obtain. Even if the malicious user manages to know that the algorithm is following a stacking behaviour, it will be costly to obtain a malicious co-residency and to know the exact PMs with target users as it will need to launch many VMs periodically to be able to achieve SCA. Afterwards, at line 20, the algorithm will check if the selected PM_j will produce a malicious co-residency. This step is essential to trigger the *VM migration function*, which is explained in the next section. The **coResidencyCheck** function motivated by the learning model that learns the behaviour of VMs and classifies them accordingly to their types. If there is no malicious co-residency the assignment of VM_i to PM_j is performed by the function **Assign**, at line 21, and added to the one allocation list. The **Assign** function will override any previous allocation commitment for the same VM. Meaning if the function is accessed again by the same VM, and the same PM selected, it will result in selecting this new assignment as final. The process of adding to the one allocation list will ensure a unique VM allocation resulted from the **Assign** function. However, in the next step, if there is a malicious co-residency on the selected PM, at line 24, the VM migration will trigger. One of the objectives of SRS is to minimize the number of VM migrations. Because migrating the VM from one PM to another results in some downtime, even for the live migration. The process of VM migration requires moving the VM current state while the VM is running. Furthermore, at a particular stage, copying the VM state and restoring it in the destination PM requires a slight downtime. That result in an unwanted interruption to most cloud users, which SRS tries to avoid by only selecting to migrate the VMs that are allocated on the elected PMs. After the VM migration is performed, the VM allocation produced from the migration will be added to the *one allocation list* to reflect the new changes, if any. Then the algorithm will repeat the malicious co-residency check after the migration. Finally, at line 28, if performing the initial selection and VM migration leads to a failed assignment, then the algorithm will assign the VM_i to any suitable PM. The one allocation function will continue until all the unallocated VMs find an assignment.

5.3 The VM Migration Function

The goal of the *one allocation function* is to produce a secure allocation. A secure migration must leave the system in a secure allocation. The migration function attempts to maintain that secure status, whenever possible. The *migration function* receives the elected PMs to select their VMs for migration and the available PMs set, denoted as P. The reason for this step is to select the minimum number of VMs for migration, thus reducing the VM movements. The VMs selected for migration, at line 5, will include all the malicious VMs and all normal VMs. This step will allow more space for the target VMs, by migrating the normal VMs to allow more space, and thus, reduces the chance of co-residency. Also, selecting the malicious VMs, will increase the difficulty for the malicious VM user to achieve SCA. Hence, reducing the possibility of being allocated with a target VM. Finally, the selected VMs are allocated to specific PMs according to the same steps of one allocation function.

6 RESULTS AND DISCUSSION

In this section, we evaluate the algorithms, using the CloudSim simulator (Calheiros et al., 2011). CloudSim is an open-source cloud simulation environment based on cloud system workloads. Cloud simulators are useful to provide a solution and projection of the real world scenarios.

This work examines the effect of the secure VM allocation, VM migration, and PMs used during the allocation. Also, we compare against well-known allocation algorithms such as Round Robin (RR) (Balharith and Alhaidari, 2019) and Random (Rand) algorithms (Azar et al., 2014). Furthermore, we compare against one of the recently developed algorithms,

called Previously Selected Servers First (PSSF). (Han et al., 2015b). We consider these algorithms as they capture different approaches and behaviours of VM allocations. Our proposed SRS algorithm allocates VMs in a stacking fashion, while the Random algorithm allocates VMs randomly, and RR allocates through spreading. The PSSF follow a unique behaviour that depends on spreading the VMs of the same user if they exceed 3 VMs on the same PM. The Random, RR and PSSF algorithms do not check the type of the VMs during VM allocation, as part of the **coResidencyCheck** function, as SRS does. Therefore, and in order to trigger the VM migration, we have added this function to all the considered algorithms. However, the migration function performed for each algorithm is based on its initial allocation. This action will preserve the behaviour of each algorithm from being changed after the migration.

6.1 Arrival of VMs

In this experiment, we consider three arrival times (launch times), to show the effect of VM arrival time, based on its type, on the malicious co-residency. The three arrival times are *M(t)*, *T(t)* and *N(t)*. The *M(t)* is the time that the malicious VM is arrived. The definition applies to *T(t)* and *N(t)* for target VM and normal VM, respectively. In the experiment, we study all the possibilities of arrival time for each type of VM. The notation $M(t) < T(t) < N(t)$ means that the malicious VM arrives and is allocated in a time before the arrival of target and normal VMs. We also consider when the target and normal VMs arrive at the initial time.

6.2 Experimental Setup

The number of VMs ranges from 20-120, increasing by 20 VMs in each experiment. The number of PMs is 24 in each experiment. We consider three types of PMs structure, or level of PMs heterogeneity, High, Medium and Low heterogeneous PMs. Meaning the resources of the PMs are structured based on the classification of PMs heterogeneity, as follows:

1. **HighHetPMs.** The first eight PMs can host the VMs as following and in this order (2VM-4VM-6VM-8VM-2VM-4VM-6VM-8VM). Then this order repeated until it reaches 24 PMs to accommodate up to 120 VMs.
2. **MedHetPM.** Here it will be as following (4VM-4VM-4VM-4VM-4VM-6VM-6VM-8VM). Then this order repeated as above.
3. **LowHetPMs.** Here it will be as following (4VM-4VM-4VM-4VM-6VM-6VM-6VM-6VM). Then this order repeated as above.

The resource requirements of the VMs are similar with 1 GB vRAM (Virtual RAM), 1 vCPU and 500 MB vStorage. On the other hand, the resources available for each PM are heterogeneous, as described above. There are four types of PMs used for this setup: (i) 2 GB RAM and 2 CPU, (ii) 4 GB RAM and 4 CPU, (iii) 6 GB RAM and 6 CPU, and (iv) 8 GB RAM and 8 CPU. The CPU is space-shared, meaning that each CPU can only accommodate one vCPU at each time. We investigate every combination of numbers of VMs with each type of PMs under each VMs arrival times. We also consider the number of each VMs type in each experiment, for example, if we have a 20 VMs, how many of these VMs are malicious VM, target VMs, or normal VMs. We distributed these possibilities in our simulation, and due to the limited space for this paper, we are not able to show the effect for each VM type number.

6.2.1 External Workload

We used a real workload while conducting the experiments in order to mimic real cloud computing scenarios as much as possible. The used workload is published by the Karlsruhe Institue of Technology ForHLR II System (Mehmet, S, 2018).

6.2.2 Figures Explanation

Figures 3−5 compare four algorithms, SRS, PSSF, Random and RR under three different PMs heterogeneity. Each type of a PM structure, for example **HighHetPMs:** is highlighted vertically with a shading color to demonstrate which part of the PM structure. And each comparison Figure, for example Figures 3a, 3b and 3c compares three different VMs arrivals times. Therefore, for example, when the VMs equal 20, it will be simulated for each possible arrival time under each type of PMs heterogeneity and each possible VM type number.

6.3 Result of Malicious Co-residency

We calculate the percentage of PMs with Malicious allocations, denoted as (M_{pms}), as follow:

$$M_{pms} = \frac{I_{pms}}{T_{pms}} \tag{1}$$

Where the (I_{pms}) specify the infected used PMs, and the (T_{pms}) specify the total used PMs for an allocation.

6.3.1 VM Arrival: $M(t) < T(t) < N(t)$

In Figure 3a, the PSSF and RR are the worst for the M_{pms} due to the spreading behaviour and because the

malicious VMs arrived first which helped to spread the VMs. Nevertheless, PSSF only shows weakness when the number of VMs increases and the available recourse on the PMs start limiting. Moreover, the RR suffers more from increasing M_{pms} when the PMs are **HighHetPMs**, compared to other PMs structures. This is because the PMs available for allocation filled more quickly than others PM structure, which leaves fewer options. SRS and Random are the best in this situation. The SRS shows the least M_{pms} due to the stacking behaviour.

6.3.2 VM Arrival: $N(t) < M(t) < T(t)$

In Figure 3b, this considers the most challenging case for any allocation algorithm, as the target and malicious VMs arrives at the end, when most of the resources are already utilized. The options for a secure allocation become challenging for this case. However, and except for PSSF, most of the algorithms performed well even with limited resources. For PSSF, due to its constraint of keeping only three users on the same PM, lead to spread target and malicious, which result in higher M_{pms}. This because the normal VMs, for each experiment, arrives first, and spread their VMs on the available PMs. Thus fewer available PMs, when the malicious and target VMs arrives. Also, The SRS shows the least M_{pms} among others.

6.3.3 VM Arrival: $T(t) < N(t) < M(t)$

In Figure 3c, and according to their paper as well, this is the best case for PSSF. When the target VMs arrives before other types, the result of the M_{pms} improved significantly. However, for RR, this is the worst case. This because of the same reason mentioned in Figure 3a but with the opposite case. A notable case for the Random, as it performed worse here than other situation due to the arrival of malicious VMs at the end. The SRS shows the least M_{pms} among others.

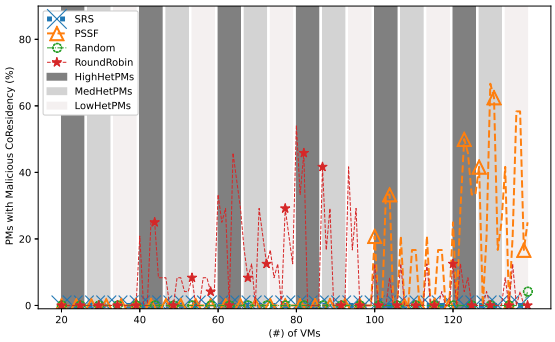
6.4 Result of VM Migration

The percentage of VMs migrations, denoted as (Mig_{vms}), is defined as follow:

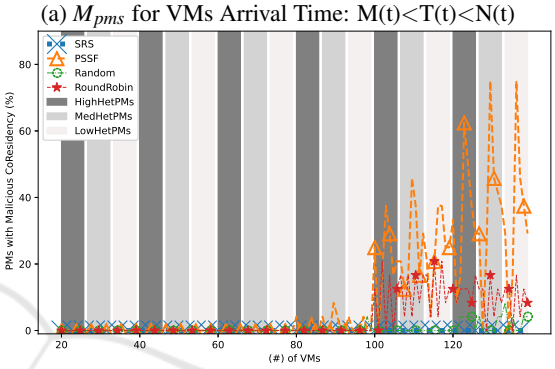
$$Mig_{vms} = \frac{S_{vms}}{T_{vms}} \tag{2}$$

Where the (S_{vms}) specify the VMs selected and migrated from one PM to another, and the (T_{vms}) specify the total VMs for an allocation.

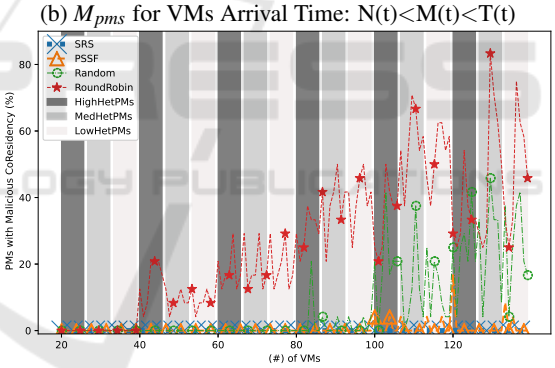
In the Figures 4a, 4b and 4c, the percentage of VMs migrations (Mig_{vms}) is an indication of the processing needed to obtain a secure allocation. We can see clearly, in Figure 4c that the processing needed is



(a) M_{pms} for VMs Arrival Time: M(t)<T(t)<N(t)



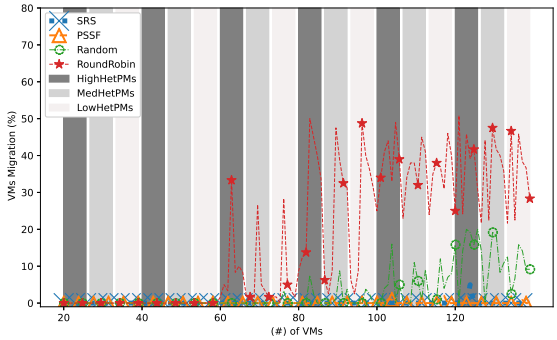
(b) M_{pms} for VMs Arrival Time: N(t)<M(t)<T(t)



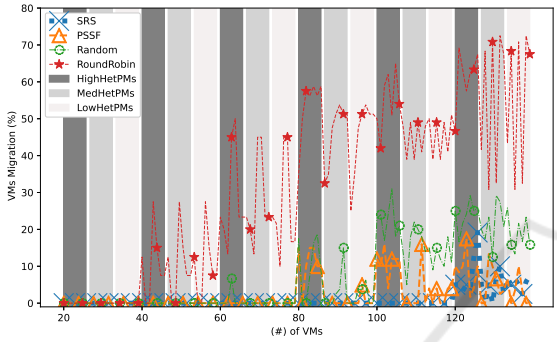
(c) M_{pms} for VMs Arrival Time: T(t)<N(t)<M(t)

Figure 3: Comparison of percentage of the malicious co-residency under three arrivals times for $M(t)$,$T(t)$ and $N(t)$.

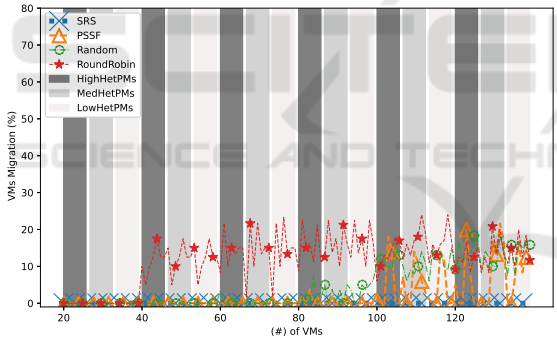
far less for all algorithms compare to the other VMs arrivals. In general, the RR needed more VM migrations to obtain the resulted level of M_{pms} compares to other algorithms. The SRS only suffers from extensive VM migration in the situation of Figure 4b, as this is the most challenging situation, as we described earlier. However, the Mig_{vms} is still considerably low compared to RR and Random, and is similar to PSSF. Another notable case, the Mig_{vms} is quite high for SRS in **HighHetPMs** compares to other structure. Because of the number of options for available PMs, limiting quickly, and therefore, VMs migration triggered more than other structures.



(a) Mig_{vms} for VMs Arrival Time: M(t) < T(t) < N(t)

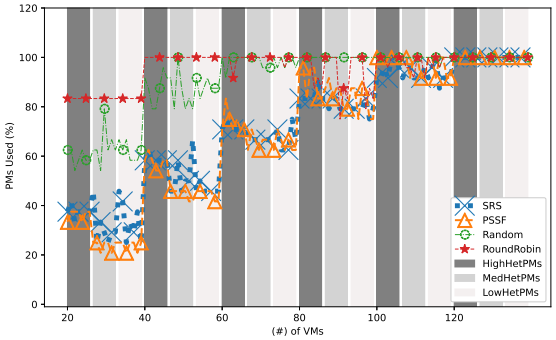


(b) Mig_{vms} for VMs Arrival Time: N(t) < M(t) < T(t)

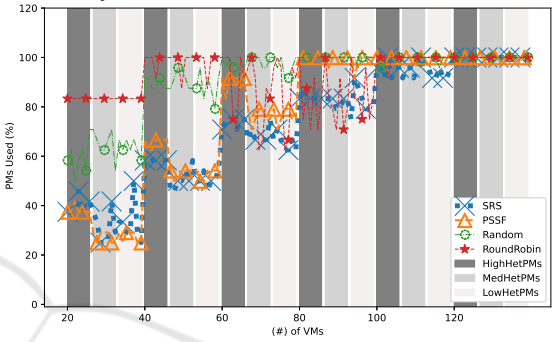


(c) Mig_{vms} for VMs Arrival Time: T(t) < N(t) < M(t)

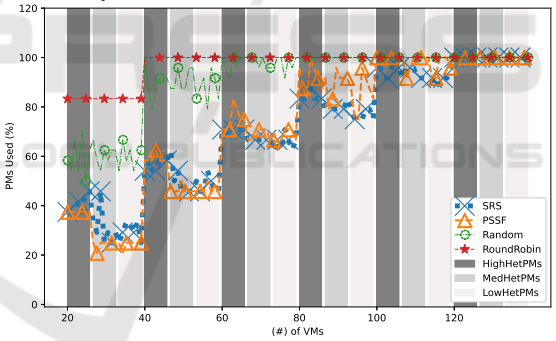
Figure 4: Comparison of percentage of the VMs migrations under three arrivals times for *M(t)*,*T(t)* and *N(t)*.



(a) $Usage_{pms}$ for VMs Arrival Time: M(t) < T(t) < N(t)



(b) $Usage_{pms}$ for VMs Arrival Time: N(t) < M(t) < T(t)



(c) $Usage_{pms}$ for VMs Arrival Time: T(t) < N(t) < M(t)

Figure 5: Comparison of percentage of the PMs usage under three arrivals times for *M(t)*,*T(t)* and *N(t)*.

6.5 Result of PMs Usage

To calculate the effect of the number of used PMs for each algorithm, we calculate the percentage of used PMs compared to the total available PMs, denoted as ($Usage_{pms}$), as follow:

$$Usage_{pms} = \frac{Used_{pms}}{T_{pms}} \tag{3}$$

Where the ($Used_{pms}$) specify the used PMs for completing an allocation, and the (T_{pms}) specify the total available PMs.

In the Figures 5a, 5b and 5c, there is an indication of the resource usage, in an efficient manner, towards obtaining a secure allocation. In general, RR is worse due to its spreading behaviour while Random is only better when the available resources are not limited. SRS and PSSF have similar behaviours in almost all situations. A notable case, the $Usage_{pms}$ is always higher for SRS and PSSF in **HighHetPMs** compared to other structures. As the number of options for available PMs is reduced, $Usage_{pms}$ increases. Also, as shown in Figure 5b, SRS makes efficient use of resources, achieving high capacity when the number of VMs is high, compared to other approaches which

approach such capacity at a much lower number of VMs. The process of selecting the most secure allocation, in SRS, focuses on obtaining the most secure allocations, rather than obtaining the ones with the least used PMs. Hence, the $Usage_{pms}$ in SRS is considerably higher than PSSF when the available resources are not limited.

7 CONCLUSION

This paper proposed a secure VM allocation (SRS) to defend against SCA in CCEs. The presented algorithm aims to find a secure allocation by preventing or reducing co-residency of a target VM with a malicious VM. Our results show that VM arrival times have a significant impact on obtaining a secure allocation. Also, the algorithms that follow a stacking behaviour in VM allocations are more likely to return secure allocations than spreading or random-based algorithms. We show that SRS outperforms other schemes in obtaining a secure VM allocation. In future work, we will investigate further other factors that affect secure VM allocations. We also plan on integrating service level agreements (SLAs) into the allocation process.

REFERENCES

Afoulki, Z., Bousquet, A., Rouzaud-Cornabas, J., et al. (2011). A security-aware scheduler for virtual machines on iaas clouds. *Report 2011*.

Agarwal, A. and Duong, T. N. B. (2019). Secure virtual machine placement in cloud data centers. *Future Generation Computer Systems*, 100:210–222.

Ahamed, F., Shahrestani, S., and Javadi, B. (2016). Security aware and energy-efficient virtual machine consolidation in cloud computing systems. In *2016 IEEE Trustcom/BigDataSE/ISPA*, pages 1516–1523. IEEE.

Al-Haj, S., Al-Shaer, E., and Ramasamy, H. V. (2013). Security-aware resource allocation in clouds. In *2013 IEEE International Conference on Services Computing*, pages 400–407. IEEE.

Azar, Y., Kamara, S., Menache, I., Raykova, M., and Shepard, B. (2014). Co-location-resistant clouds. In *Proceedings of the 6th Edition of the ACM Workshop on Cloud Computing Security*, pages 9–20.

Bahrami, M., Malvankar, A., Budhraja, K. K., Kundu, C., Singhal, M., and Kundu, A. (2017). Compliance-aware provisioning of containers on cloud. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pages 696–700. IEEE.

Balharith, T. and Alhaidari, F. (2019). Round robin scheduling algorithm in cpu and cloud computing: a review. In *2019 2nd International Conference on Computer Applications & Information Security (ICCAIS)*, pages 1–7. IEEE.

Bazm, M.-M., Lacoste, M., Südholt, M., and Menaud, J.-M. (2017). Side channels in the cloud: Isolation challenges, attacks, and countermeasures.

Berrima, M., Nasr, A. K., and Ben Rajeb, N. (2016). Co-location resistant strategy with full resources optimization. In *Proceedings of the 2016 ACM on Cloud Computing Security Workshop*, pages 3–10.

Bijon, K., Krishnan, R., and Sandhu, R. (2015). Mitigating multi-tenancy risks in iaas cloud through constraints-driven virtual resource scheduling. In *Proceedings of the 20th ACM Symposium on Access Control Models and Technologies*, pages 63–74.

Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., and Buyya, R. (2011). Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, 41(1):23–50.

Ding, W., Gu, C., Luo, F., Chang, Y., Rugwiro, U., Li, X., and Wen, G. (2018). Dfa-vmp: An efficient and secure virtual machine placement strategy under cloud environment. *Peer-to-Peer Networking and Applications*, 11(2):318–333.

Han, J., Zang, W., Chen, S., and Yu, M. (2017). Reducing security risks of clouds through virtual machine placement. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 275–292. Springer.

Han, J., Zang, W., Liu, L., Chen, S., and Yu, M. (2018). Risk-aware multi-objective optimized virtual machine placement in the cloud. *Journal of Computer Security*, 26(5):707–730.

Han, Y., Alpcan, T., Chan, J., Leckie, C., and Rubinstein, B. I. (2015a). A game theoretical approach to defend against co-resident attacks in cloud computing: Preventing co-residence using semi-supervised learning. *IEEE Transactions on information Forensics and Security*, 11(3):556–570.

Han, Y., Chan, J., Alpcan, T., and Leckie, C. (2015b). Using virtual machine allocation policies to defend against co-resident attacks in cloud computing. *IEEE Transactions on Dependable and Secure Computing*, 14(1):95–108.

Hu, Y., Wong, J., Iszlai, G., and Litoiu, M. (2009). Resource provisioning for cloud computing. In *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research*, pages 101–111.

Jansen, W. A. (2011). Cloud hooks: Security and privacy issues in cloud computing. In *2011 44th Hawaii International Conference on System Sciences*, pages 1–10. IEEE.

Korf, R. E. (2002). A new algorithm for optimal bin packing. In *Aaai/Iaai*, pages 731–736.

Li, M., Zhang, Y., Bai, K., Zang, W., Yu, M., and He, X. (2012). Improving cloud survivability through dependency based virtual machine placement. In *SECRYPT*, pages 321–326.

Liang, X., Gui, X., Jian, A., and Ren, D. (2017). Mitigating cloud co-resident attacks via grouping-based virtual machine placement strategy. In *2017 IEEE 36th*

International Performance Computing and Communications Conference (IPCCC), pages 1–8. IEEE.

Mehmet, S (2018). Karlsruhe Institue of Technology forhlr ii system. https://www.cs.huji.ac.il/labs/parallel/workload/l_kit_fh2/index.html/. Last checked on Dec 01, 2020.

Moon, S.-J., Sekar, V., and Reiter, M. K. (2015). Nomad: Mitigating arbitrary cloud side channels via provider-assisted migration. In *Proceedings of the 22nd acm sigsac conference on computer and communications security*, pages 1595–1606.

Natu, V. and Duong, T. N. B. (2017). Secure virtual machine placement in infrastructure cloud services. In *2017 IEEE 10th Conference on Service-Oriented Computing and Applications (SOCA)*, pages 26–33. IEEE.

Qiu, Y., Shen, Q., Luo, Y., Li, C., and Wu, Z. (2017). A secure virtual machine deployment strategy to reduce co-residency in cloud. In *2017 IEEE Trustcom/BigDataSE/ICESS*, pages 347–354. IEEE.

Sprabery, R., Evchenko, K., Raj, A., Bobba, R. B., Mohan, S., and Campbell, R. H. (2017). A novel scheduling framework leveraging hardware cache partitioning for cache-side-channel elimination in clouds. *arXiv preprint arXiv:1708.09538*.

Spreitzer, R., Moonsamy, V., Korak, T., and Mangard, S. (2017). Systematic classification of side-channel attacks: A case study for mobile devices. *IEEE Communications Surveys & Tutorials*, 20(1):465–488.

Sun, Q., Shen, Q., Li, C., and Wu, Z. (2016). Selance: Secure load balancing of virtual machines in cloud. In *2016 IEEE Trustcom/BigDataSE/ISPA*, pages 662–669. IEEE.

Will, M. A. and Ko, R. K. (2017). Secure fpga as a service—towards secure data processing by physicalizing the cloud. In *2017 IEEE Trustcom/BigDataSE/ICESS*, pages 449–455. IEEE.

Wong, Y. and Shen, Q. (2018). Secure virtual machine placement and load balancing algorithms with high efficiency. In *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, pages 613–620. IEEE.

Yuchi, X. and Shetty, S. (2015). Enabling security-aware virtual machine placement in iaas clouds. In *MILCOM 2015-2015 IEEE Military Communications Conference*, pages 1554–1559. IEEE.

Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1):7–18.

Zhang, Y., Li, M., Bai, K., Yu, M., and Zang, W. (2012). Incentive compatible moving target defense against vm-colocation attacks in clouds. In *IFIP international information security conference*, pages 388–399. Springer.

Zhou, Y. and Feng, D. (2005). Side-channel attacks: Ten years after its publication and the impacts on cryptographic module security testing. *IACR Cryptology ePrint Archive*, 2005:388.