Toward Secure VMs Allocation: Analysis of VMs Allocation Behaviours in the Cloud Computing Environments

Mansour Aldawood[1], Arshad Jhumka[1], and Suhaib A. Fahmy[2,3]

[1] Department of Computer Science, University of Warwick, Coventry, UK
{m.aldawood, h.a.jhumka}@warwick.ac.uk
[2] King Abdullah University of Science and Technology (KAUST), Thuwal, Saudi Arabia
[3] School of Engineering, University of Warwick, Coventry, UK
{suhaib.fahmy}@kaust.edu.sa

Abstract. Side-channel attacks (SCAs) is a potential threat in cloud computing environments (CCEs) as it allows the malicious VMs to capture private information from the target VMs when they share the same PM. This malicious co-residency of VMs is an outcome of the VMs allocation algorithm behaviour, which is responsible for allocating the VMs to a specific PM based on defined allocation objectives. Earlier studies tackled the SCAs, through specific solutions, by focusing on either formulating VMs allocation algorithms or modifying the architecture of the CCEs to mitigate the threats of SCAs. However, most of them are oriented to specific situations and assumptions, leading to malicious co-residency when applied to other scopes or situations. In this paper, we presented the solution from a different holistic perspective by examining the allocation behaviours of different algorithms and other properties that affect and lead to obtaining a secure VMs allocation. The examinations are performed under different scenarios and structures for each behaviour to understand the possible situations that lead to secure VMs allocation. In addition, we develop deterministic security-aware VMs allocation algorithm that aim to allocate the VMs securely to reduce the potential threats from malicious co-residency in CCEs.

Keywords: Cloud computing · Virtual machines secure allocation · Side-channel attacks.

1 Introduction

Cloud computing users can utilise the computing resources or services offered by the cloud service providers (CSPs) through the network and on-demand basis. These services include servers, storage, networks, applications and other services. The virtualisation technique is the core of cloud computing systems, where it enables the abstraction and sharing of computing resources accessible across a network by a group of users. Virtualisation enables a group of virtual machines

(VMs) belonging to different users to share the physical machines (PMs) while running separately. In the traditional on-premises computing data centre, a PM will be dedicated to a single-purpose application, while in cloud computing, many applications belongings to different users can be hosted on a single PM [19].

The virtualisation allows the CSPs to maximise the utilisation of the cloud resources while minimising the cost of operating the cloud infrastructure. Moreover, it enhances the cloud users utilisation of computing resources based on their needs to avoid resources wastage; thus, the provisioning of resources is elastic, based on user requirements. As a consequence, the physical resources are shared among users, and the security threats for the CCEs have invariably shifted as the types of threats that arise when a malicious user shares the resources with a target user [3]. In other words, 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. This attack will have an impact extending from the application level to the hardware level and becoming more prevalent due to the range of side channels [7].

Thus, to overcome the problem of SCAs, 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. Otherwise stated, the VMs allocation algorithm responsible for allocating the VMs into specific PMs needs to be security-oriented to defend against the SCAs threats. In general, the VMs allocation's objective depends on the desired outcome of the allocation process, for instance, reducing the power consumption of the PMs. In some cases, the allocation objective is related to network traffic control, which allocates the related VMs on the same network subnet [23]. In comparison, this paper focuses on the VMs allocation algorithms that aim to allocate the VMs securely in CCEs to defend against SCAs.

The previous researches tackling the SCAs and malicious VMs co-residency in CCEs focused on either finding a solution logically on the VMs level or physically on the PMs level. While in this paper we will investigate the behaviour of various state-of-the-art VMs allocation algorithms and their effect on producing secure VMs allocations. In our earlier work [2], we proposed the study of secure VMs allocation behaviours, however, the study was preliminary and did not cover more situations and properties that effect obtaining secure VMs allocations.

As such, the following points summarise the intended outcomes of this research:

1. Investigate the behaviour of various state-of-the-art VMs allocation algorithms and their effect on producing secure allocations. These are (i) Round Robin [6], (ii) Random [4] (iii) previously selected servers first (PSSF) [18] and (iv) Secure Random Stacking (SRS) [2]. Each of these algorithms has unique allocation behaviours. Hence, we consider three VMs allocation behaviours: (i) stacking, (ii) spreading and (iii) Random.
2. Develop and evaluate deterministic secure VMs allocation algorithm called Secure Stacking (SS) that aim to reduce the chance of malicious co-residency while using fewer available PMs in the cloud system.

3. Examine the effect of defined properties on producing secure VMs allocations. These properties are: (i) The impact of VMs arrival based on their type. For instance, study the impact of the arrival of malicious VMs before the target VMs. (ii) The impact of the number of VMs based on their classified type. For example, the arrival of many malicious VMs while the number of target VMs is small. (iii) The impact of the structure of the available resources of the PMs or demanded resources of the VMs. (iv) The impact of the VMs allocation behaviours algorithms on the VMs migration number and PMs usage. In other words, studying the effect of the number of VMs migrated and the number of PMs utilised during the allocation will aim to achieve secure VMs allocation.

Generally, the extensive examination of the allocations behaviours under different properties shows that the stacking-based behaviours algorithms are more likely to produce secure allocations than those with spreading-based or random-based allocation behaviours algorithms. As such, our stacking-based algorithm are significantly better as they produce secure allocations more than the compared algorithms under the same examined situations. Furthermore, our results show that VMs arrival time has a considerable impact on the secure allocations, where the arrival of target or malicious VMs earlier than the rest of VMs often leads to malicious co-residency avoidance. Lastly, our stacking-based algorithm show the lowest PMs usage among the compared algorithms, by significant amounts, under most examined situations, leading to utilising fewer PMs and therefore fewer power consumption of the available resources. Moreover, the number VMs migration is the lowest among the examined algorithms. Hence, leading to the high availability of the VMs in cloud systems by avoiding many interruptions resulting from the VMs migration while enhancing the state of the secure allocations.

The following sections of the paper are structured as follows; In section 2, we will introduce the related works and domains tackling the SCA problem in CCEs. Subsequently, in section 3, we will develop and examine the considered secure VMs allocation model in CCEs. Afterwards, in section 4, we will present our secure VMs allocation algorithm. Then, in section 5, we will present an extensive evaluation and comparison with other VMs allocation behaviours. Finally, in section 6, we will conclude our work by summarising the key findings and propose possible future works direction.

2 Related Work

This section reviews previous researches tackling the SCAs and malicious VMs co-residency in CCEs. The areas that tackled SCAs focused on either finding a solution logically on the VMs level or physically on the PMs level. The section divides the previous research into six domains aiming to secure the VMs allocation process in CCEs from different perspectives.

The first domain focuses on grouping the VMs based on defined requirements through the VMs allocation. Then, these requirements cluster the VMs

into groups to achieve secure VMs allocation. In [22], they proposed a group-based allocation policy to optimise the resources and obtain a secure VMs allocation by establishing group instances for the VMs. These groups have specific requirements such as group size limit or resource availability which defines the number of distinct users in each group, not only the number of VMs. This user size limitation potentially enhances the secure allocation of the VMs. In [17, 21, 30, 9, 29], other grouping approaches focus on isolating the allocated VMs into groups based on defined requirements such as network dependency or attributes that define VMs users.

The second domain considers another form of grouping as it depends on allocating the VMs based on either profiling the VMs or based on security compliance requirements. For example, in [10], a method was proposed to allocate the VMs by maintaining the same security standard as the co-located VMs, such as the ISO standard. Similarly, in [5, 1] , their scheme validates the level of security of each upcoming VMs, which must comply with specific compliance regulation. In comparison, some works focus on generalising the VMs allocation based on profiling the users and integrating the outcome with the existing placement constraints [25].

In the third domain, the solutions focus on allocating the VMs for a defined time to reduce the amount of sensitive data leakage through the SCAs. The SCAs happens when malicious VMs co-locate for a certain amount of time with target VMs, then initiate the attack by capturing related information, through the leakage channels, about the target VMs activities. Thus, these proposals focuses on reducing the amount of data leakage due to the SCAs by considering the time factor for VMs residing on specific PMs [27, 24].

The fourth domain focuses on developing algorithms that manipulate the cloud system's scheduling component. For example, in [8], introduce an algorithm that deliberately delayed the start-up time for VMs to reduce the chance of co-residency with malicious VMs. While in [32], they focus on migrating the VMs frequently by using an incentive approach, by providing better PMs with more free resources to stimulate the users to migrate their VMs. Thus, to move the VMs periodically to reduce the probability of malicious co-residency. Further, in [4, 18], their algorithms randomly select and allocate VMs into specific PMs to reduce the chance of co-residency.

The fifth domain follows an optimisation-based approach to secure VMs allocation while utilising existing solutions related to ideal situations. For instance, in [11], they proposed a solution that depends on utilising an optimisation-based problem called Dolphin Partner, prioritising the VMs with the most efficient energy-aware and memory-aware utilisation. Their work focuses on the parameters that potentially cause a failure for the VM, for example, the memory utilisation of the VM. Thus, if the current utilisation of a VM is high, it is more prone to failure and considered a less secure. Similarly, in [12], they proposed a solution to enhance the security and performance of the VMs allocation process using the firefly algorithm to produce an optimal secure allocation. Moreover, [16] presented a secure VMs allocation algorithm based on multi-objective opti-

misation by extending the First-Fit algorithm, which provides an allocation that satisfies the resource constraint.

The last domain focuses on the secure VMs allocation on the hardware level of the cloud system. The mechanism includes partitioning memory caches or CPU processing threads to defend against the software level of SCAs. It requires a modification in the hardware level, which includes either changing the mechanism of an existing components or adding new ones [20, 14, 31].

3 Problem Formulation

We will define the model of the secure VMs allocation in CCEs, including the definition of the model's objective and its assumed constraints. The model's objective is to obtain a secure VMs allocation to defend against SCAs by minimising the malicious co-residency. Moreover, it includes defining the objective of the VMs allocation, which is producing a secure VMs allocation under different situations while reducing the utilised PMs. In addition, we defined the threats of malicious users, including the attacker's behaviour and the impact of the attack that the SCAs could leave on the compromised system. Furthermore, we will define cloud users based on their behaviour analysis, thus, classifying them into specific types.

3.1 Threat Model

In CCEs, resource allocation is flexible and enables multiple users to share a common computing platform dynamically. When VMs are co-resident or co-located on the same PM, a malicious VM can analyse characteristics of another target VM, e.g., analysing the operations timing properties, to infer various information such as cryptographic keys through SCAs. For instance, the SCA can occur through a cache-based channel by utilising the sharing capabilities of the cache levels. In other words, the malicious VMs can analyse the execution time of the VMs co-locating on the same PM and subsequently conduct the attack. This analysis starts by capturing the execution data of the target VMs, then analysing them to formulate an attacks model using a machine learning-based approach [26].

Achieving Malicious Co-residency From the malicious user perspective, the first step of conducting SCA is to achieve a co-residency with the target VMs, leading to a malicious co-residency. Achieving such a goal depends on the VMs allocation algorithm that the CSP utilises to allocate the VMs. Alternatively stated, the behaviour of the VMs allocation algorithm contributes significantly to achieving malicious co-residency by the malicious VM. Therefore, the malicious user needs to understand how the CSPs allocates the VMs to formulate the attacks based on this knowledge. For instance, in [28], they studied the possibility of achieving a malicious co-residency based on the allocation algorithms on different public CCEs, such as Amazon or Google. Their study concluded that

the malicious user could reach this goal simply and cheaply due to the vulnerabilities of the VMs allocation algorithms. Hence, the malicious co-residency can be achieved due to the limitation of the allocation algorithms not considering the severity and impact.

Capturing Execution Time Data After the malicious co-residency occurs between the malicious VMs and the target VMs, the malicious user will initiate the SCA. In this work, we assumed the cache-based attack as the considered attack model conducted by the malicious user. It starts by utilising the vulnerabilities of the shared cache among VMs allocated on the same PM. The malicious VMs can perform the attack by measuring the execution time of the load operations of the shared caches on the PM level. If a specific operation utilises a considerable amount of time to load, compared to the other operations, the attacker will deduce a current encrypted operation executing on the physical machine from a co-resided VM [7].

Overall, the SCAs collect information from normal operations output, such as execution time on cache levels. Furthermore, the collected information has no major impacts when treated separately. However, with sophisticated tools that can classify and cluster irrelevant data to meaningful information, such as machine learning tools, this process can lead to major SCAs. Otherwise stated, the extracted information will help to profile the activities of the VMs co-located on the same PM and define the vulnerable state of the target VMs. For example, in [15], they utilise a machine learning-based approach to conduct a cache-based attack by profiling the activities of cloud users. They captured data features resulting from cache-based access that represents different types of applications. Their approach showed that the captured information could be collected regardless of synchronising the cache access between the malicious and target VMs.

3.2 Secure VMs Allocation Model

The main objective of the proposed secure VMs allocation model is to obtain a secure allocation where the target VMs and malicious VMs not sharing the same PM, thus, defending against SCAs. Moreover, the proposed model aims to find allocations where the number of used PMs is minimised. Therefore, our model following a stacking-based VMs allocation behaviour such as Bin-Packing problem (BPP) [13].

Definition of Variables and Functions The following are the variables and functions definitions of the model:

1. $P = PM_1 \dots PM_k$: ***Set of physical machines.***
2. $R(PM_j)$: ***Available resources of a physical machine (j).***
3. $V = VM^1 \dots VM^n$: ***Set of virtual machines.***
4. $N(VM^i)$: ***Required resources of a virtual machine (i).***
5. T: ***Set of virtual machines classified as a Target.***

6. M: ***Set of virtual machines classified as a Malicious.***
7. N: ***Set of virtual machines classified as a Normal.***
8. $A_u : V \rightarrow P$: ***VM allocation function to a PM.***
9. $Move(A_u, A_{u+1})$: ***Set of VMs that are migrated during transition from (A_u) to (A_{u+1}).***
10. $CoRe(A_u)$: ***Set of PMs at which malicious co-residency occurs.***

Objective Function Formulation In summation, we will define the objective function and its constraints of the proposed model as an approximation of BPP. Our main objective is to obtain a secure VMs allocation while reducing the number of used PMs. In other words, the priority is not reducing the utilised PMs, the priority to obtain a secure VMs allocation; however, it is a constraint to influence the allocation algorithm if it is only possible while maintaining secure allocations. Thus, our objective function is described as follow:

The objective is ***Minimize***

$$\sum_{i=1}^{n} CoRe(A_u) * x_{ij} \qquad \forall_{i \in V,\ j \in P,\ \mathbf{for}(j=1 \ldots k)} \tag{1}$$

Subject to:

$$\sum_{j=1}^{k} y_j \leq |P| \qquad \forall_{j \in P} \tag{2}$$

$$\sum_{i=1}^{n} N(VM^i) * x_{ij} \leq R(PM_j) * y_j \qquad \forall_{i \in V,\ j \in P,\ \mathbf{for}(j=1 \ldots k)} \tag{3}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad \forall_{i \in V,\ j \in P,\ \mathbf{for}(j=1 \ldots k)} \tag{4}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall_{\mathbf{for}(i=1 \ldots n),\ \mathbf{for}(j=1 \ldots k)} \tag{5}$$

$$y_j \in \{0, 1\}, \qquad \forall_{\mathbf{for}(j=1 \ldots k)} \tag{6}$$

Starting from Eq. (1), which aims to minimise the malicious co-residency of a selected possible allocation, i.e., for each possible VMs allocation of requested VMs and available PMs, the objective is to select a possible allocation that yields to produce an allocation with minimum malicious co-residency. The first constraint in Eq.(2) is to make sure that the selected number of PMs is reduced as much as possible. As we stated earlier, we aim to allocate the VMs into selected PMs while minimising the used PMs as an objective, but in our model, we utilised this objective as a constraint. Because our goal is to obtain a secure VMs allocation while reducing the number of used PMs as much as possible. The second constraint in Eq.(3) will verify that the requested resources of the selected VMs are not exceeding the available resources of the available PMs.

The third one, in Eq.(4), verifies that each VM is allocated once on a single PM to prevent duplicated allocations. The equations, Eq.(5 to 6) are defining the decisions variables needed for selecting the best possible allocations, which are x_{ij} and y_j. The x_{ij} responsible for selecting the best allocation that results in obtaining a minimum malicious co-residency. The x_{ij} is an integer value of either 0 or 1, where one means the allocation is selected and zero otherwise. The y_j responsible for selecting the allocations with fewer possible numbers of PMs.

VMs Migration In case of a VM migration triggered, we formulate the following equation as a constraint of the objective function:

$$|Move(A_u, A_{u+1})| \leq \beta \tag{7}$$

This equation denotes that the number of VMs in a set of VMs that are selected for a VM migration, $Move(A_u, A_{u+1})$, is less than or equal to a defined threshold, β. In other words, for each transition from A_u to A_{u+1}, the number of VMs selected, for migration, should not exceed a certain defined threshold. Defining the threshold depends on several aspects that determine how many VMs can be selected, such as an service level agreement (SLA) that forces some VMs to be allocated on a PM at all times. In this case, these VMs will not be selected for VMs migration even if they are eligible; therefore, the number of VMs migrating is reduced.

VMs Learning Model As stated in our earlier work [2], we introduce a learning module which aims to classify the VMs based on their behaviours. The analysis of VMs behaviour is crucial for CSPs to identify VMs with suspicious behaviour and isolate them from other VMs. Briefly, the analysis of the learning model produces a categorisation of the VMs into three types; these types are target, malicious and normal VMs. Formally stated, the set V, the set of all VMs available in CCEs, is partitioned into three sets: (i) set T of target VMs, (ii) set M of malicious VMs and (iii) set N of normal VMs.

4 Secure VMs Allocation Algorithm

We propose a deterministic security-aware heuristic, a variant of bin-packing, called Secure Stacking (SS), which is shown in Algorithm 1. Mainly, SS aims to allocate VMs in a stacking fashion and migrates them from one PM to another if the possibility of VM migration exists. Like a BPP, the SS 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 motivation behind utilising a smaller number of PMs is to avoid VMs migration during the allocation, which leads to unwanted service interruptions of the VMs during the migration process.

4.1 Secure Stacking Algorithm (SS)

The SS algorithm has two main inputs: (i) the unallocated set of VMs, denoted as **V** and (ii) the set of the available PMs, denoted as **P**. The output, denoted as the **A**, is the secure allocation produced for the available set of resources.

Algorithm 1: Secure Stacking (SS) VMs Allocation

```
   Input: V: Set of unallocated VMs, P: Set of PMs
   Output: A: Secure Allocation
 1 sortedPMsList ← ∅
 2 COR ← Flase
 3 for v in V do
 4 |   sortedPMsList ← getSortedFRPMs(v,P)
   |   // First try to allocate v
 5 |   for p in sortedPMs do
 6 |   |   COR ← getCORvmCheck(v,p.getVMslist())
 7 |   |   if COR ≠ True then
 8 |   |   |   A ← Assign(v,p)
 9 |   |   end
10 |   end
11 |   if v.getPM() = ∅ // Second try, migrate VMs then retry allocate v
12 |    then
13 |   |   vmMigration(sortedPMsList,P)
14 |   |   Repeat steps from 5 - 10
15 |   end
16 |   if v.getPM() = ∅ // Third try, allocate v in any available P
17 |    then
18 |   |   for p in P do
19 |   |   |   if p.suitablePM(v) = True then
20 |   |   |   |   A ← Assign(v,p)
21 |   |   |   end
22 |   |   end
23 |   end
24 end
25 return A
```

The SS algorithm starts, at line 3, by allocating the VMs, in the set of unallocated VMs in the set of the available PMs. It goes through three trials of allocating the VMs, and each trial has its specific functions. From line 5 to line 10, the first try aims to allocate the VMs securely in a stacking fashion without triggering the VMs migration. Meaning the SS will try to obtain a secure VMs allocation while meeting the required resources constraints without changing the structure of the current VMs allocation, i.e., triggering VMs migrations. On the second try, from line 11 to line 15, the SS will try again to obtain a secure VM allocation for the same VM; however, this time will trigger the VM migrations, thus changing the current structure of the allocated VMs. On the third try, from line 16 to line 23, the SS reach the point to allocate the VM to any available PM, regardless of the security constraints. Meaning the priority at this point is to obtain a VM allocation to any suitable PM.

Fullness Ratio In line 4, the SS will sort the available PMs based on their fullness ratio (FR) by comparing the require resources of the VM, denoted as v, with the available PMs resources. In other words, the SS prioritised the PMs for an allocation based on the fullness of each PM, which means that each PM will be filled differently after the allocation of the upcoming VM. Thus, triggering *getSortedFRPMs(v,P)* that compares each requires resource from the VM (v) to the available PMs resources. Then, we will sort the PMs based on the FR and produce a list of the sorted PMs, called *sortedPMsList*.

The main objective of the FR function, denoted as *getSortedFRPMs(v,P)*, is to sort the available PMs resources according to their FR compares to the VM required resources that needs to be allocated. Therefore, the FR function is comparing the remaining resources of each PM with demanded resources of the upcoming VM. This comparison results in a sorted PMs list based on how much the PM will be filled after the allocation if the allocation occurs. Hence the SS algorithm following a stacking-based behaviour in allocating the VMs. We consider the RAM size and the CPU cores with their sizes are the essential resources to be validated during the FR calculation. Thus, we will explain how the calculation of the FR for the multidimensional resources (MR) is performed in the FR function

The detailed calculation of MR described as follow:

$$MR = (VM_{ram} \div PM_{ram}) + (VM_{cpu} \div PM_{cpu}) \tag{8}$$

The idea is to compare the VM with the PM, considering the RAM and CPU specifications. In Eq(8), firstly, the FR function divides the required RAM of the VM by the available RAM of the PM. If the results of this part are exceeding one, then it means that the required RAM is more than the available RAM of PM; thus, this PM is discarded. Then, repeat the same division of the RAM part with the CPU part. Moreover, finally sum the result of the two parts, the RAM and CPU calculation. If the result is equal to two, then this PM is a perfect match for the VM. In other words, this PM will be the first PM selected for possible allocation. The results represent how much the PM will be utilised for the unallocated VM; thus, it will be prioritised based on this result.

First Try we will start with the first try and explain its main functions, from lines 5 to 10. In line 5, the SS will try the first PM, denoted as p, out of the produced FR PMs to allocate the unallocated VM on it, denoted as v. The p, at this stage, represent the highest FR of the available PMs. Meaning, this p, if selected, will yield to be filled more than the other available PMs. Thus, this step is contributing to the stacking behaviour of the SS allocation algorithm. Then, in line 6, the SS will check if allocating the VM, v, yields a malicious co-residency. According to the presented learning model, in [2], each VM in the CCEs classified into either target, malicious or normal VM. Thus, at this step, the SS will compare the upcoming VM with the allocated VMs on the selected PM, if any, for malicious co-residency. If the malicious co-residency will occurs after allocating the VM, this PM will be discarded from the allocation and moved

on to the following PM. The result of this checking is preserved in a Boolean variable denoted as *COR* after triggering the *getCORvmCheck(v,p.getVMslist())* function. This function is essential to the SS as it will be the main responsible for triggering the VMs migration in the second try. The last step of the first try, in line 8, is to assign the v into the selected p if previous conditions are met.

Second Try In lines 11 to 15, the second try starts if the first try failed to obtain a secure allocation for the unallocated VM. As indicated in line 11, the second try is triggered if the v is not allocated to a PM yet. The primary step of this try, in line 13, is the VM migration function, which is changing the structure of the current VMs allocation by moving the allocated VMs, if possible, to another PMs to obtain a secure VM allocation. The VMs migration aims to migrate a few VMs to obtain a secure allocation for the unallocated VM and enhance, or keep, the current secure allocation state. In other words, we aim to migrate the VMs while reducing the number of VMs migrated and maximising, or maintaining, the current security state of allocated VMs.

The *vmMigration(sortedPMsList,P)* receives the list PMs to select their VMs for migration and the available PMs set, to select the destination PM after migration. The selection step, selecting the PMs list, aims to select the minimum number of VMs for migration, thus reducing the VM movements. For example, the SS algorithm utilises the VMs migration function by sending the list of the sorted PMs, to migrate their VMs. This list of PMs has a low number of VMs compared to all the available PMs. Thus, the VMs allocated on these VMs will be minimal; hence the VMs selected for migration will be minimised. We consider this way of selecting the VMs for migration to allow few effective migrations that potentially leads to a more secure allocation and fewer VMs interruptions resulted from the migration. Although the number of VMs selected for migration will be higher at some point in time, specifically when there is a high number of PMs available with high (FR%).

Third Try The last step of the SS algorithm is started, in lines 16 to 23, if the previous two steps failed to obtain a secure VM allocation. At this step, the SS will allocate v to any available PM regardless of the malicious co-residency allocation constraints. In other words, as long as the selected PM has enough available resources, it will be selected to host the unallocated VM. Afterwards, in line 25, all the assignments will be registered in **A** and returned as a final allocation.

5 Evaluation

We will present a detailed evaluation of the proposed algorithms under different PMs and VMs structures and different allocation scenarios. As such, we study the effect of VMs allocation behaviour on obtaining a secure allocations. The behaviours are stacking, spreading and random behaviour. We investigate the

factors affecting the outcome towards obtaining a secure allocation. These are; the PMs heterogeneity level, the diversity of available resources, the VMs arrival time for each type of VMs considered in this work and the number of VMs according to their classified type. Additionally, we study the effect of VMs migration and the efficient PMs usage for the proposed algorithms on the overall outcome of a secure allocation.

5.1 Allocation Behaviours Comparison

In this part, we will introduce the VMs allocation behaviours algorithms to compare our SS algorithm with them, and each of them has a unique allocation behaviour. The first one is spreading behaviour, which means that the allocation algorithm will allocate VMs to the whole span of PMs. An example of the spreading behaviour is the round-robin algorithm, denoted as RR, described in [6]. The second one is random behaviour (Random), which aims to allocate the VMs randomly as long as the candidate PM is suitable. In [4], they presented a random-based algorithm aiming to allocate the VMs randomly. The third considered allocation behaviour, called the PSSF, is a combination of spreading and random behaviour. This behaviour algorithm, described in [18], depends on spreading the VMs of the same user if they exceeded three VMs on the same PM and, eligible PMs are selected randomly if they have less than three VMs of the same user. Moreover, we include our previous algorithm, SRS, in the comparison of this work [2].

Furthermore, we modify the three algorithms, RR, Random and PSSF, by making them aware of the learning model outcome. Otherwise stated, we have added the co-residency detection function while keeping their allocation behaviour the same. These algorithms will allocate the VMs as they have been doing unless there is a malicious co-residency in the allocation.

5.2 Experimental Setup

We will explain the simulation environment utilised in this work, the structure of the PMs resources considered during the allocation process, the VMs structure including the VMs arrival times and the structure of VMs type.

Simulation Environment Similar to our earlier work [2], we utilise CloudSim, a cloud computing simulation environment, to examine the proposed VMs allocation algorithms and compare them.

VMs and PMs Number The VMs range from 20-120, increasing by 20 VMs in each experiment. The number of PMs is 24 in each experiment, where the sum of available resources of the PMs can accommodate up to 120 VMs. Thus, the experiments will start by allocating the VMs with vast available resources; then, the resources get limited until it reaches 120 VMs. 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. 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.

VMs Arrival Time We consider three arrival times (launch times), to show the effect of VMs 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 same definition applies to *T(t)* and *N(t)* for target VM and normal VM, respectively.

Table 1: VMs Arrival Time Types

Tries No.	VMs Order	Description
1	GNMT	G(N), G(M), G(T)
2	SNMT	S(N), S(M), S(T)
3	Mixed NMT	S(NMT), G(N), S(NMT), G(M), S(NMT), G(T), S(NMT)

As shown in Table 1, we study some of the possibilities of VMs arrival time based on each type of VMs. For instance, in try 1, we study when a group of normal VMs arrives, then a group of malicious VMs arrives, then a group of target VMs arrives last, denoted as GNMT. Furthermore, in try 2, the VMs will arrive a single instead of a group, meaning one normal VM arrives, followed by malicious, followed by target, denoted as SNMT. Lastly, in the try 3, we study the arrival time as a mixed of single and group arrivals. The size of each group, the seven groups of the mixed order type, divided equally to each group. The motivation behind designing the arrival times in this sequence is to mimic the real-world scenario of VMs arrival as much as possible.

VMs Type Structure Table 2 considers seven possible situations where each VMs type number might reach for each experiment. Moreover, each VMs type number, tries 1 for instance, will be examined for its secure VMs allocation level and how it performs under this defined configuration. To explain, if we consider 20 VMs, then this VMs type number will be structured 7 times, as described in Table 2, and examined for each situation. The seven tries are because we have three VMs types considered, and $2^3 = 8$ possible situations. However, we discarded the one where the VMs type number are zeros from these eight possible situations.

PMs Heterogeneity levels We consider three types of PMs structure, or level of PMs heterogeneity, High, Medium and Low heterogeneous PMs. Meaning the

Table 2: VMs Type structure

Tries No.	% Malicious VMs	%Normal VMs	%Target VMs
1	25	25	50
2	25	50	25
3	26	37	37
4	50	25	25
5	37	26	37
6	37	37	26
7	33	34	33

resources of the PMs are structured based on the classification of PMs heterogeneity, as explained in [2].

5.3 Results of Malicious Co-residency Respect to VMs Type and under Limited Resources Availability

Continue to our work in [2], we will have a closer look at the M_{pms} concerning the VMs type number, the VMs arrival and PMs heterogeneity. Here, we only will show the results when the resources are limited, which means when the number of VMs equal 120 VMs. The M_{pms} is the percentage of PMs with malicious co-residency, calculated as follow:

$$M_{pms} = \frac{I_{pms}}{U_{pms}} \tag{9}$$

Where the (I_{pms}) specify the infected used PMs, and the (U_{pms}) specify the total used PMs for an allocation. We only will show the results when the resources are limited, which means when the number of VMs equal 120 VMs.

Malicious Co-residency for Group VMs Arrival under Limited Resources In Figure 1, this case 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 utilised.

From VMs type number perspective, the PSSF algorithm often suffers from high M_{pms} when the number of malicious VMs or targets VMs higher than the other types. In some cases, the higher number of malicious and target VMs leads to a high M_{pms}. This outcome is because this case considers the group VMs arrivals, meaning a group of VMs, possibly belonging to the same user, will be allocated simultaneously. Since the PSSF spreads the VMs of the same user, and if the user is a malicious one, then the chance of malicious co-residency occurring is very high for such allocation behaviour. The same applies when many VMs belong to a target user or arrive simultaneously with a considerably high number of VMs. For RR, spreading the VMs is negatively impacting the M_{pms} as it is

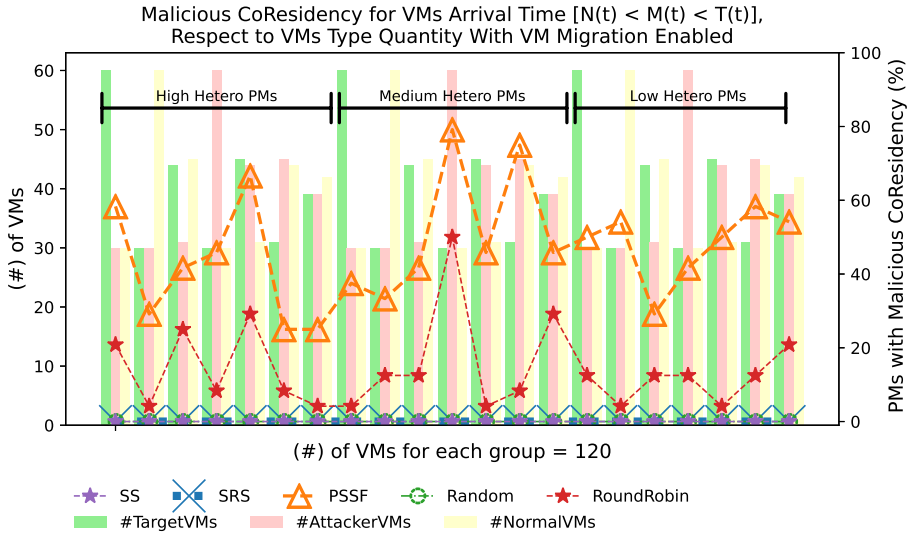

Fig. 1: Malicious Co-residency under GNMT Arrival Time, When Available Resources Limited.

considered among the worst of compared algorithms. The reason for the high M_{pms} is the same as we described in the PSSF algorithm, as they share the spreading behaviour of allocating the VMs. Overall, the SS, SRS, and Random algorithms are best when the VMs arrive in groups. The stacking of the VMs reduces the number of used PMs during the allocation and creates a perfect match between the required resources and the available recourse, which is what SS and SRS perform. Thus, avoiding the chance of producing allocations with high M_{pms}.

From VMs arrival time perspective, the constraint of the PSSF algorithm that keeps only three users on the same PM leads to spread target and malicious VMs, which results in higher M_{pms}. Thus, if the malicious user launched many VMs, it will be easier to obtain a malicious co-residency with the target user. Also, because the normal VMs, for each experiment, arrives first and spread their VMs on the available PMs. Hence, fewer available PMs when the malicious and target VMs arrives, which is also applies to RR algorithm.

From PMs heterogeneity perspective, when comparing the effect of PMs heterogeneity level, the low heterogeneous PMs structure often leads to a better result of M_{pms} than the other PMs structure for PSSF and RR.

Malicious Co-residency for Single VMs Arrival under Limited Resources In Figure 2, the single VMs arrivals lead to better results comparing to the group or mixed VMs arrivals.

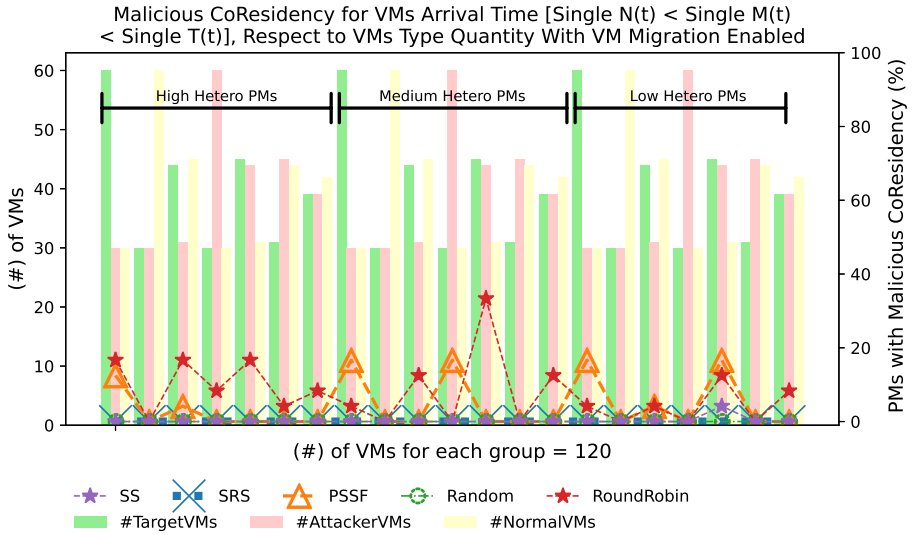

Fig. 2: Malicious Co-residency under SNMT Arrival Time, When Available Resources Limited.

The effect of the VMs number, according to their type, is similar to the group arrivals, for all the algorithms. Briefly, the higher number of either malicious or target VMs, and in some cases when both are high, leads to a higher chance of malicious co-residency occurrence.

Moreover, when the VMs arrived separately, the M_{pms} decreases significantly for the compared algorithms even when the available recourse is limited. This outcome happens because it is easier for the allocator to obtain a secure allocation for a single VM, according to its type. However, when a group of VMs of the same type arrives, it is not easy to produce a secure allocation. For example, when a group of target VMs arrives and the malicious VMs already allocated to most of the available PMs, it will be challenging for the algorithms to obtain a secure allocation, especially if the algorithm follows a spreading behaviour. On the other hand, a single target VM arriving makes it easier to obtain the secure allocation because the available PMs options that lead to secure allocation is potentially higher, even for the spreading based algorithms.

The high heterogeneous PMs structure often leads to a better result of M_{pms} than the other PMs structure for all algorithms due to the high diversity of the structure of the resources. Ultimately, the impact of VMs number continues to be the same on the single VMs arrival.

Malicious Co-residency for Mixed VMs Arrival under Limited Resources In Figure 3, the outcomes of the M_{pms} is better than the group VMs arrival as this type mixed the group with the single VMs arrivals. Thus the single

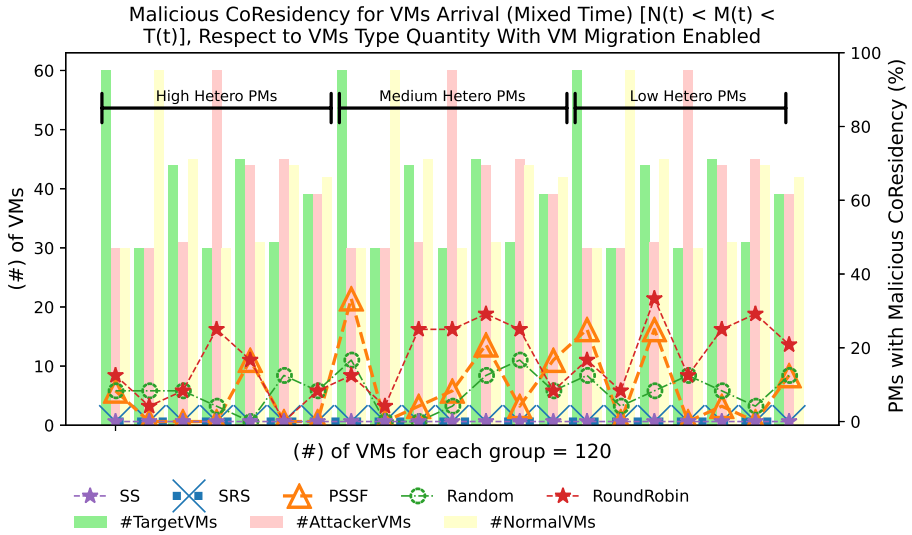

Fig. 3: Malicious Co-residency under Mixed NMT Arrival Time, When Available Resources Limited.

VMs arrival influences the positive impact of obtaining more secure allocations for all the algorithms.

From VMs type number perspective, the PSSF and Random algorithms are showing a clear relationship between the spike number of either target or malicious VMs with the high M_{pms}. Even in the cases where they both have a relatively high number at the same time compared to the total number of VMs, of the experiment. Also, the high number of normal VMs positively leads to low, sometimes none, malicious co-residency. However, this effect disappears when the number of either target or malicious VMs rises. Similarly, the RR algorithm was impacted by the rising number of target and malicious VMs. The SS and SRS continue to produce the best outcome of the compared algorithms over the examined situations.

From VMs arrival perspective, the normal VMs will arrive first, then any VM from the other two types can be allocated with them. Thus, leaving more options and more available PMs for the upcoming VMs when it arrives to obtain secure allocations. However, the single VMs arrival between the groups causing the spike of M_{pms} the same way happened in the previous arrivals.

Furthermore, PMs heterogeneous structure's effect did not seem to have that great difference between the three types in the matter of the number of spikes; however, the high heterogeneous is slightly better in most cases in the matter of producing low M_{pms}.

5.4 Results of VMs Migrations

This section will compare the result of VMs migration for all the compared algorithms under different arrival times. The percentage of VMs migrations, denoted as (Mig_{vms}), is defined as follow:

$$Mig_{vms} = \frac{S_{vms}}{T_{vms}} \tag{10}$$

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.

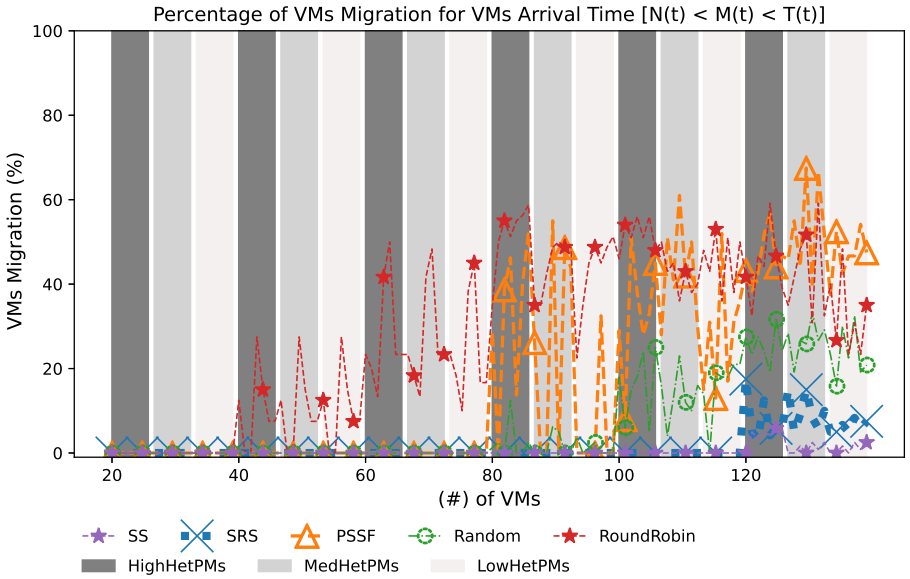

Fig. 4: VMs Migrations under Single VMs arrival, GNMT Arrival Time.

VMs Migrations for Group VMs Arrival In general, as shown in Figure 4, the spreading allocation behaviours algorithms, RR and PSSF, are the worst in Mig_{vms}, especially when the resources are limited. While the random algorithm, have a moderated percentage of VMs migration considering benefits produced by these migrations, which is a lower chance of malicious co-residency. The stacking-based algorithms, SS and SRS, show the lowest percentage of VMs migration among the other algorithms under all the group VMs arrival. However, the SRS algorithm shows high Mig_{vms} compared to the SS algorithm in a few cases when the resources are limited and the malicious and target VMs arrive at last. The SS algorithm is considered to have the best outcome of Mig_{vms} under all the

examined situations. On the other hand, the reason that influences the Mig_{vms} to be low in some cases is the limited options of the available PMs and the limited number of selected VMs for migration. As a consequence of this behaviour, the M_{pms} will produce a higher percentage than other cases of the same algorithm under different arrival times.

Overall, from the algorithm perspective, the benefits for VMs migration are high for the random, SRS and SS algorithms, but for PSSF and RR, the benefits are not significant. For instance, the high Mig_{vms} for the SRS algorithm leads to obtaining secure allocations for all the cases examined. Also, the random algorithms benefit greatly from the VMs migration as it produces many allocations without high M_{pms}. On the other hand, for RR and PSSF, their benefits are not as much as the other algorithms due to their spreading behaviour that limits VMs migration options.

VMs Migrations for Single VMs Arrival In Figure 5, the RR case shows high Mig_{vms} compared to the other algorithms, even when the resources are not limiting. The reason for this behaviour back to two main points; the configurations of VMs arrivals and the behaviour of the algorithm. The VMs arrival structure, in this case, depends on separating the VMs as single based on their type classification. Thus, it is easier for the malicious VMs, or target VMs, to spread access to the entire available PMs at early stages. This spreading brings us to the second reason, which is the behaviour of the algorithm, which depends on spreading the VMs upon their arrivals. Hence, making the Mig_{vms} much higher compare to the other algorithms.

VMs Migrations for Mixed VMs Arrival In Figure 6, the similarity of outcome for VMs migration continues for this type of VMs arrival, where the RR algorithm performs the worse among the compared algorithms due to its spreading behaviour. Similarly, the PSSF shows a high Mig_{vms} only when the resources start limiting, which indicates that obtaining secure allocation at this stage is challenging. Moreover, the Rand algorithm low Mig_{vms} compare to the spreading behaviour algorithms, RR and PSSF. The stacking-based algorithms, SS and SRS, are the best in this time arrivals are they yielding to the lowest Mig_{vms} for all the cases.

5.5 Results of PMs Usage

This section aims to examine the PMs utilisation, ($Usage_{pms}$), during the VMs allocations for all the compared algorithms. The PMs utilisation is also considered an indication of the power consumption for the compared algorithms. We calculate the percentage of used PMs compared to the total available PMs, denoted as ($Usage_{pms}$), as follow:

$$Usage_{pms} = \frac{U_{pms}}{T_{pms}} \tag{11}$$

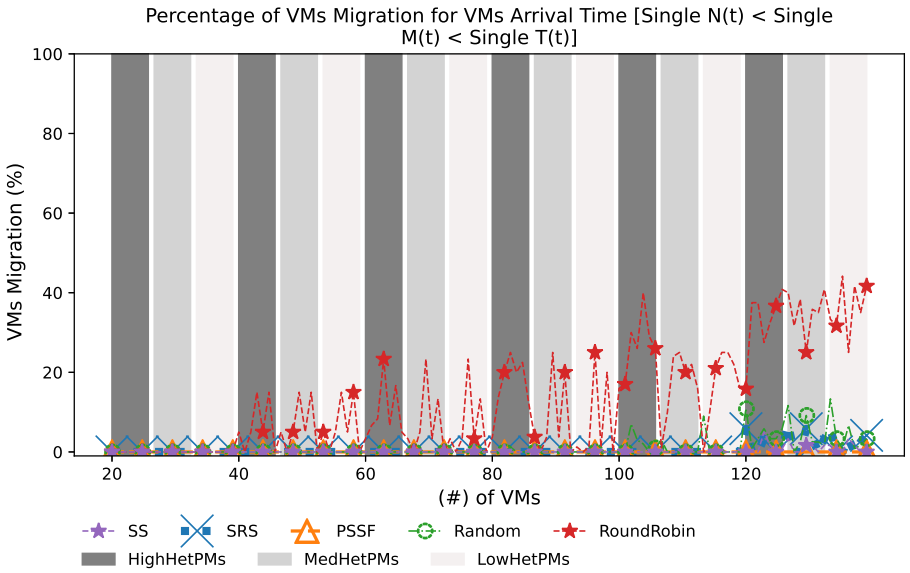

Fig. 5: VMs Migrations under Single VMs arrival, SNMT Arrival Time.

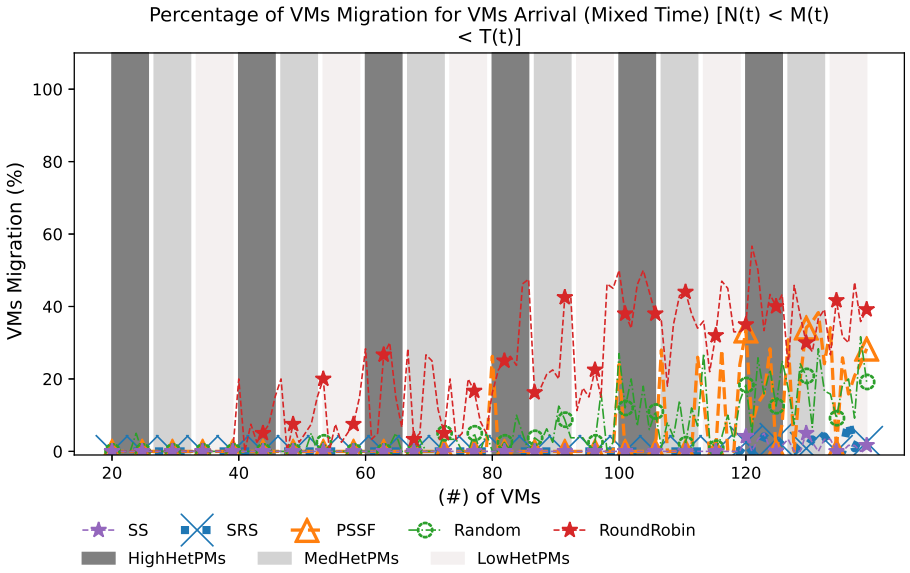

Fig. 6: VMs Migrations under Single VMs arrival, Mixed NMT Arrival Time.

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

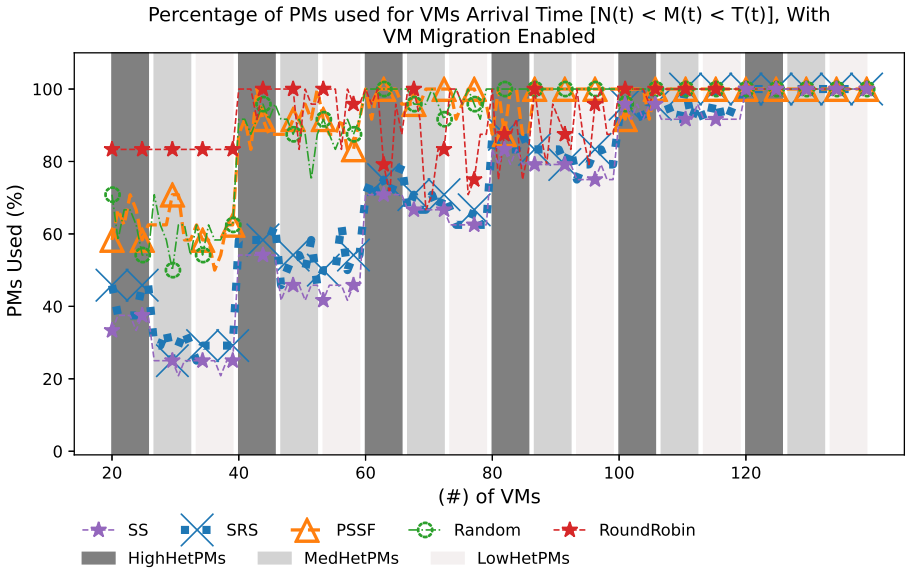

Fig. 7: PMs Usage under GNMT Arrival Time.

VMs Arrival (GNMT) To avoid duplication of similar results, we will only show the case of group VMs arrival, as shown in Figure 7.

Overall, in Figure 7, there is an indication of the resource usage, efficiency towards obtaining a secure allocation. In other words, in our proposed SS algorithm, the $Usage_{pms}$ are the best among the compared algorithms under most cases, even when the resources start limiting. On the other hand, the RR algorithm is generally worse due to its spreading behaviour, while Random and PSSF are only better when the available resources are not limited.

In a notable case, the $Usage_{pms}$ is slightly higher for SS and SRS in high heterogeneous PMs than other PMs structures for the same VMs number. For instance, when the VMs number equal 20, the PMs usage is considered higher than the other types despite the fact that the PMs number is the same for all the structures, but with different available resources. The possible reason is that the high heterogeneous PMs filled early than the other two types due to the design of this PM structure, which leads to utilising more PMs, during the allocation, than medium and low heterogeneous structures.

6 Conclusion

This paper focuses on evaluating the behaviour of the secure VMs allocation algorithms that leads to produce secure allocations in CCEs. Moreover it focuses on obtaining a secure VMs allocation in CCEs to defend against SCAs. As such, we propose a solution to defend against this attack by developing SS algorithm and examining it under different situations with other algorithms. 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 SS outperform other schemes in obtaining a secure VM allocation. In future work, we will be extending the proposed model to include tasks allocation on the hardware level in addition to the VMs level. In other words, it depends on controlling the allocation of tasks on CPUs and caches to allocate them securely and reduce data leakage through the side channels. It includes classifying the tasks according to the user behaviour and allocate their tasks accordingly.

References

1. Ahamed, F., Shahrestani, S., Javadi, B.: Security aware and energy-efficient virtual machine consolidation in cloud computing systems. In: 2016 IEEE Trustcom/BigDataSE/ISPA. pp. 1516–1523. IEEE (2016)
2. Aldawood, M., Jhumka, A., Fahmy, S.A.: Sit here: Placing virtual machines securely in cloud environments. In: CLOSER. pp. 248–259 (2021)
3. Almorsy, M., Grundy, J., Müller, I.: An analysis of the cloud computing security problem. arXiv preprint arXiv:1609.01107 (2016)
4. Azar, Y., Kamara, S., Menache, I., Raykova, M., Shepard, B.: Co-location-resistant clouds. In: Proceedings of the 6th Edition of the ACM Workshop on Cloud Computing Security. pp. 9–20 (2014)
5. Bahrami, M., Malvankar, A., Budhraja, K.K., Kundu, C., Singhal, M., Kundu, A.: Compliance-aware provisioning of containers on cloud. In: 2017 IEEE 10th International Conference on Cloud Computing (CLOUD). pp. 696–700. IEEE (2017)
6. Balharith, T., Alhaidari, F.: Round robin scheduling algorithm in cpu and cloud computing: a review. In: 2019 2nd International Conference on Computer Applications & Information Security (ICCAIS). pp. 1–7. IEEE (2019)
7. Bazm, M.M., Lacoste, M., Südholt, M., Menaud, J.M.: Side channels in the cloud: Isolation challenges, attacks, and countermeasures (2017)
8. Berrima, M., Nasr, A.K., Ben Rajeb, N.: Co-location resistant strategy with full resources optimization. In: Proceedings of the 2016 ACM on Cloud Computing Security Workshop. pp. 3–10 (2016)
9. Bijon, K., Krishnan, R., Sandhu, R.: 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. pp. 63–74 (2015)
10. Caron, E., Le, A.D., Lefray, A., Toinard, C.: Definition of security metrics for the cloud computing and security-aware virtual machine placement algorithms. In: 2013 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery. pp. 125–131. IEEE (2013)

11. Dhanya, D., Arivudainambi, D.: Dolphin partner optimization based secure and qualified virtual machine for resource allocation with streamline security analysis. Peer-to-Peer Networking and Applications **12**(5), 1194–1213 (2019)
12. Ding, W., Gu, C., Luo, F., Chang, Y., Rugwiro, U., Li, X., Wen, G.: Dfa-vmp: An efficient and secure virtual machine placement strategy under cloud environment. Peer-to-Peer Networking and Applications **11**(2), 318–333 (2018)
13. Garey, M.R., Johnson, D.S.: Computers and intractability, vol. 174. freeman San Francisco (1979)
14. Genssler, P.R., Knodel, O., Spallek, R.G.: Securing virtualized fpgas for an untrusted cloud. In: Proceedings of the International Conference on Embedded Systems, Cyber-physical Systems, and Applications (ESCS). pp. 3–9. The Steering Committee of The World Congress in Computer Science, Computer ... (2018)
15. Gulmezoglu, B., Eisenbarth, T., Sunar, B.: Cache-based application detection in the cloud using machine learning. In: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security. pp. 288–300 (2017)
16. Han, J., Zang, W., Liu, L., Chen, S., Yu, M.: Risk-aware multi-objective optimized virtual machine placement in the cloud. Journal of Computer Security **26**(5), 707–730 (2018)
17. Han, J., Zang, W., Chen, S., Yu, M.: Reducing security risks of clouds through virtual machine placement. In: IFIP Annual Conference on Data and Applications Security and Privacy. pp. 275–292. Springer (2017)
18. Han, Y., Chan, J., Alpcan, T., Leckie, C.: 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 (2015)
19. Hu, Y., Wong, J., Iszlai, G., Litoiu, M.: Resource provisioning for cloud computing. In: Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research. pp. 101–111 (2009)
20. Kiriansky, V., Lebedev, I., Amarasinghe, S., Devadas, S., Emer, J.: Dawg: A defense against cache timing attacks in speculative execution processors. In: 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). pp. 974–987. IEEE (2018)
21. Liang, X., Gui, X., Jian, A., Ren, D.: Mitigating cloud co-resident attacks via grouping-based virtual machine placement strategy. In: 2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC). pp. 1–8. IEEE (2017)
22. Long, V.D., Duong, T.N.B.: Group instance: Flexible co-location resistant virtual machine placement in iaas clouds. In: 2020 IEEE 29th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE). pp. 64–69. IEEE (2020)
23. Lopez-Pires, F., Baran, B.: Virtual machine placement literature review. arXiv preprint arXiv:1506.01509 (2015)
24. Moon, S.J., Sekar, V., Reiter, M.K.: Nomad: Mitigating arbitrary cloud side channels via provider-assisted migration. In: Proceedings of the 22nd acm sigsac conference on computer and communications security. pp. 1595–1606 (2015)
25. Natu, V., Duong, T.N.B.: Secure virtual machine placement in infrastructure cloud services. In: 2017 IEEE 10th Conference on Service-Oriented Computing and Applications (SOCA). pp. 26–33. IEEE (2017)
26. Ristenpart, T., Tromer, E., Shacham, H., Savage, S.: Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In: Proceedings of the 16th ACM conference on Computer and communications security. pp. 199–212 (2009)

27. Sun, Q., Shen, Q., Li, C., Wu, Z.: Selance: Secure load balancing of virtual machines in cloud. In: 2016 IEEE Trustcom/BigDataSE/ISPA. pp. 662–669. IEEE (2016)
28. Varadarajan, V., Zhang, Y., Ristenpart, T., Swift, M.: A placement vulnerability study in multi-tenant public clouds. In: 24th {USENIX} Security Symposium ({USENIX} Security 15). pp. 913–928 (2015)
29. Yu, S., Gui, X., Tian, F., Yang, P., Zhao, J.: A security-awareness virtual machine placement scheme in the cloud. In: 2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing. pp. 1078–1083. IEEE (2013)
30. Yuchi, X., Shetty, S.: Enabling security-aware virtual machine placement in iaas clouds. In: MILCOM 2015-2015 IEEE Military Communications Conference. pp. 1554–1559. IEEE (2015)
31. Zhang, T., Zhang, Y., Lee, R.B.: Cloudradar: A real-time side-channel attack detection system in clouds. In: International Symposium on Research in Attacks, Intrusions, and Defenses. pp. 118–140. Springer (2016)
32. Zhang, Y., Li, M., Bai, K., Yu, M., Zang, W.: Incentive compatible moving target defense against vm-colocation attacks in clouds. In: IFIP international information security conference. pp. 388–399. Springer (2012)