WARWICK
THE UNIVERSITY OF WARWICK

Secure Virtual Machines Allocation in Cloud Computing Environments

by

Mansour Aldawood

Thesis

Submitted to the University of Warwick

in partial fulfilment of the requirements

for admission to the degree of

Doctor of Philosophy

Department of Computer Science

October 2021

Contents

List of Tables

List of Figures

Acknowledgments

I would like to thank my supervisor Dr Arshad Jhumka for his valuable support and encouragement during my PhD. He continuously guided and encouraged me to overcome the difficulties of my research. Also, I would like to thank my co-supervisor, Dr Suhaib Fahmy, for providing guidance and feedback throughout this project. Also, I am grateful to all of my friends, lab-mates, and staff at the computer science department who contributed with their time to discuss, help and facilitate my PhD life.

Moreover, thanks to my Father, Mother, Brothers and Sisters who provided me with emotional support during my PhD.

Above all, a special thanks to my wife, Kawther and my sons, Fahad and Mohammed, who shared this experience with me and deserved endless gratitude. Their love, patience and encouragement made this journey possible to achieve.

Declarations

The author confirm that this thesis has not been submitted for a degree at another university. Moreover, parts of this thesis have either been previously published by the author or submitted under review in the following.

The following are the papers that the author of the thesis has published in joint with the authors‘ supervisors:

[7] Mansour Aldawood, Arshad Jhumka, and Suhaib A Fahmy. Sit here: Placing virtual machines securely in cloud environments. In *CLOSER*, pages 248–259, 2021

[6] Mansour Aldawood and Arshad Jhumka. Secure allocation for graph-based virtual machines in cloud environments. In *2021 18th International Conference on Privacy, Security and Trust (PST)*. IEEE, Accepted on 22 October 2021

The following is the paper that the author of the thesis has submitted and is under review, in joint with the authors‘ supervisors:

[8] Mansour Aldawood, Arshad Jhumka, and Suhaib A Fahmy. The study of secure vms allocation behaviours in the cloud computing environments. In *Communications in Computer and Information Science (CCIS)*. Under Review, Submitted to Springer, 2021

Abstract

A Cloud Computing Environment (CCE) leverages the advantages offered by virtualisation to enable the sharing of computing resources among cloud users elastically and based on the user requirements. Hence, virtual machines (VMs) can share physical resources within the same physical machine (PM).

However, resource sharing is exposed to potential security threats that can lead to a malicious co-residency, or multitenancy, between the co-located VMs. The malicious co-residency happens when a malicious VM is co-located with a critical, or target, VM on the same PM, leading to side-channel attacks (SCAs), widely recognised as a potential threat in CCEs. Specifically, the SCAs allow the malicious VMs to capture private information from the target VMs by co-locating with them on the same PM. The co-location 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. As such, the VMs allocation behaviours can potentially lead to a malicious co-residency; hence, it is significant that the implemented VMs allocation algorithms need to be made secure.

Most of the earlier studies tackled the malicious co-residency, which leads to 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. While in our work, we presented the solution from a different holistic perspective by studying the allocation behaviours and other properties that affect and lead to obtaining a secure VMs allocation. In addition, we develop a secure VMs allocation model that aims to minimise the malicious co-residency under various situations and constraints. Furthermore, we introduce an evaluation of our model using an optimisation-based approach by utilising a linear programming technique to capture the behaviour of the optimal VMs allocation. Moreover, based on the optimisation-based outcomes, we develop security-aware VMs allocation and VMs migration algorithms that aim to allocate the VMs securely to reduce the potential threats from malicious co-residency.

Therefore, to accomplish our objectives, we utilise state of the art tools and simulations such as PuLP and CloudSim to examine and implement the VMs allocation algorithms. Moreover, we perform an extensive examination of selected VMs allocation behaviours, which are stacking-based, random-based and spreading-based. The examinations are performed under different scenarios and structures for each behaviour to understand the possible situations that lead to secure VMs allocation.

Hence, we show that the stacking-based behaviours algorithms are more likely to produce secure allocations than those with spreading-based or random-based allocation behaviours algorithms. Accordingly, our stacking-based algorithms are significantly better as they produce secure allocations more than the compared algorithms under the same examined situations. Moreover, our results show that VMs arrival time has a significant impact producing secure allocations, where the arrival of target or malicious VMs earlier than the rest of VMs often minimises the malicious co-residency occurrence. In addition, the high available resources diversity between the available resources of PMs yields to produce more secure allocations as it offers more allocation options for the allocation algorithms and thus more flexibility. Furthermore, our stacking-based algorithms 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. Lastly, the number of VMs migration is the lowest among the examined algorithms, leading to the higher availability of the VMs in cloud systems by avoiding many interruptions resulting from the VMs migration while enhancing the state of the secure allocations.

Acronyms

BPP	Bin-Packing Problem.
CCE	Cloud Computing Enviroment.
CIS	Cloud Information Service.
CloudSim	Cloud Simulation.
CSP	Cloud Service Provider.
DC	Data Centre.
DoS	Denial-of-Service.
FPGA	Field-Programmable Gate Array.
FR	Fullness Ratio.
GbSRS	Graph-based Secure Random Stacking.
GbSS	Graph-based Secure Stacking.
HFR	Highest Fullness Ratio.
IaaS	Infrastructure as a Service.
IDS	Intrusion Detection System.
LP	Linear Programming.
MIP	Mixed Integer Programming.
NetCloudSim	Network Cloud Simulation.
OS	Operating System.

OSA Optimisation-based Secure Allocation.

PaaS Platform as a Service.

PM Physical Machine.

PSSF Previously Selected Servers First.

Rand Random.

RR Round Robin.

SaaS Software as a Service.

SCA Side-Channel Attack.

SLA Service-level Agreement.

SRS Secure Random Stacking.

SS Secure Stacking.

VIM Virtual Infrastructure Management.

VM Virtual Machine.

XaaS Everything as a Service.

Chapter 1

Introduction to Cloud Computing Environments

1.1 Introduction

The paradigm of Cloud Computing Enviroment (CCE) focuses on 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., CPUs, RAMs and storage) and make them accessible across a network by a pool of users. The sharing of computing resources is enabled by virtualization technology, which is a technique that provides an extensive distributed computing resource dynamically. The virtualisation technology allows cloud users to access the physical resources hosted on Physical Machine (PM) through Virtual Machine (VM). Otherwise stated, virtualisation enables the sharing of physical computing resources among cloud users efficiently and based on demand to allow the high utilisation of the physical resources.

As such, in CCEs, the resource requirements of an executing workload fluctuate; for instance, the CPUs requirements may need more resources to execute certain tasks. Thus, the provided physical resources can be dynamically allocated to or reclaimed from the users based on the defined resources needs. Hence, the provisioning of resources is elastic, based on the defined user requirements. Hence, Cloud Service Provider (CSP) can dynamically provide a comprehensive distributed computing resource to the cloud users according to their utilisation needs. Therefore, the allocation of resources is flexible and requires less time and management than traditional on-premise allocation [40].

1.2 The Threats from Malicious VMs Co-residency in CCEs

The CSPs are enabling multiple users to share a common computing platform with dynamically available resources. This dynamic and shared resources allocation invariably means that a PM can potentially share its resources among a set of distinct users or VMs, in what is known as VM co-location. As a consequence of VMs co-location, rather than VMs having dedicated resources, the cloud users will share the same set of physical resources, leading to the so-called VMs co-location threats. In other words, the VMs co-location will potentially open the doors for unprecedented security threats, ranging from data-level to system-level threats resulting from these new computing environments [21].

1.2.1 Threat of Side-Channel Attacks (SCAs)

The types of threats from VMs co-location range from data confidentiality breaches to denial-of-service attacks. Thus, VMs co-location, though enabling efficient resource sharing, is creating unwanted side-channels, which can be sources of potential Side-Channel Attack (SCA) [52]. The side-channels are (unwanted) communication channels between processes that may leak sensitive outputs from a process, among many others [97]. Hence, the SCAs are the attacks that utilise the normal outputs gained from the computer system where the communication channels between processes occurring. The SCAs can occur when a malicious VM shares the same PM with a target VM, a VM that contains sensitive information. Specifically, when VMs are co-resident (or co-located) on the 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. This attack depends on collecting related information about the target VMs, such as execution time through a cache-based attack, then analysing the collected bits of information for profiling and performing the attack.

1.2.2 Defend Against SCAs through VMs allocation

The impact of SCAs expanded from software to hardware level and becoming more prevalent due to the range of side channels that could potentially be exposed [14, 82]. 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 [65]. In comparison, this thesis focuses on the VMs allocation algorithms that aim to allocate the VMs securely in CCEs to defend against SCAs.

1.3 Problem Statement

The thesis aims to study the malicious co-residency problem in CCEs among cloud users, which leads to SCAs. In CCEs, when VMs are co-resident on the same PM, and some of the VMs are categorised as malicious VMs, this co-location is considered a malicious co-residency. Thus, the malicious co-residency means that the malicious VMs and the target VMs sharing the same PM. The malicious VM can obtain and analyse characteristics of other VMs sharing the same PM to deduce private information such as cryptographic keys. Moreover, the SCAs is considered a passive attack; therefore, it is difficult for the CSPs to defend against it and relatively easy for the attacker to perform it without being detected.

As such, the malicious VM requires a co-residency with the target VM to perform its attack. With an inefficient allocation algorithm, the malicious user can achieve such a goal depends on the VMs allocation algorithm behaviour that the CSPs utilises to allocate the VMs. In other words, the behaviour of the VMs allocation algorithm contributes significantly to either achieving malicious co-residency or secure VMs allocation.

1.4 Research Objectives

There are three main objectives of the thesis: Examine and capture the patterns from the optimised VMs allocation outcomes using an optimisation-based tool named PuLP [70]. This tool is designed to solve an optimisation problem represented in a mathematical model to produce an optimal solution. As such, the tool will examine the proposed secure VMs allocation model subject to a set of constraints to produce many potential acceptable solutions. Then, the examined outcomes lead to developing secure VMs allocation algorithms that aim to reduce the impact of malicious co-residency in CCEs and mitigate the risk of SCAs. Moreover, investigate selected VMs allocation behaviours and defined properties that affect obtaining secure VMs allocation. The following points summarise the intended outcomes of this research:

1. Develop a secure VMs allocation model that aims to allocate the VMs securely in CCEs under resource constraints while reducing the utilised PMs. Moreover, we evaluate the behaviour of the proposed model using an optimisation-based tool called PuLP [70]. The model evaluation aims to capture the optimal secure VMs allocation patterns produced by PuLP. The model evaluation aims to capture the optimal secure VMs allocation patterns produced by PuLP. The term “optimal” is used by PuLP to indicate that the produced solution complies with the defined objective and constraints (otherwise, the solution is not regarded as being optimal). The examined optimal allocation is selected out of many possible allocations under a defined set of configurations. Thus, the selected optimal allocation is bounded to the properties of the defined model constraints, resources of the VMs, PMs and allocation scenarios. The optimal allocation only exists when the set of the defined constraints of the model are met. For instance, the available resources of the PMs can accommodate the required resources by the VMs.

2. Develop and evaluate secure VMs allocation algorithms that aim to reduce the chance of malicious co-residency while using fewer available PMs in the cloud system. Furthermore, we extended our algorithms to consider the dependent VMs represented as graphs to examine the effect

of their relation on producing secure VMs allocations.

3. Investigate the behaviour of various state-of-the-art VMs allocation algorithms and their effect on producing secure allocations. These are (i) Round Robin, (ii) Random and (iii) previously selected servers first (PSSF) algorithms. Each of these algorithms has unique allocation behaviours. Hence, we consider three VMs allocation behaviours: (i) stacking, (ii) spreading and (iii) Random.

4. 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 on obtaining a malicious co-residency. (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 on the overall malicious co-residency. (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.

1.5 Research Methodology

The thesis follows a research methodology based on quantitative analysis of the gathered data from extensive performed experiments. As such, we utilise and evaluate mathematical modelling to understand the behaviour of the optimal VMs allocation algorithms. Subsequently, we performed extensive empirical experiments to understand the VMs allocation behaviours that lead to secure allocations under various cloud structures. In summation, the research methodology of this thesis contains the following:

1. Taxonomy of related secure VMs allocation algorithms: we performed a deep investigation of the previous solutions related to secure VMs allocation in CCEs. The objective of the investigation is to obtain an abstract

solution based on the previous literature by learning and classifying the main aspects that affect the security of the VMs allocation. The classification includes studying the effect of each level of the cloud system, the attacker behaviour, attack impact, and the proposed countermeasures.

2. Evaluation of mathematical model: we utilise a linear programming solver to evaluate our proposed secure VMs allocation mathematical model. The objective of utilising the solver is to make a better decision for a specific situation under a set of constraints related to the secure VMs allocation. Hence, to be able to capture the behaviour of the optimal solutions for obtaining secure VMs allocation. Knowing how to perform such activity, which captures the allocation behaviour of the optimal situation, helps develop the algorithms intended for obtaining a secure allocation.

3. Simulation of presented algorithms: we utilise real-world cloud computing traces from the Microsoft Azure data centre, which provides realistic traces of the VMs from an existing cloud provider to examine our algorithms under actual heterogeneous VMs traces. Moreover, we utilise a powerful state-of-the-art simulation tool to simulate different aspects related to the CCEs architecture called CloudSim. It is an open-source cloud simulation environment that builds based on cloud system workloads that aim to simulate the provisioning of cloud computing systems which contributes to providing a valuable estimate of the expected outcome of certain situations and an initial understanding of the behaviour of the allocation process.

1.6 Contributions of the Thesis

As stated, this thesis has three main objectives: capturing the patterns for the optimal VMs allocation model using an optimisation-based approach, developing secure VMs allocation algorithms that aim to reduce the impact of malicious co-residency in CCEs and studying the VMs allocation behaviours and their effect on reducing the chance of malicious VMs co-residency. Therefore, we summarise the key contributions of this work as follow:

1. We present a taxonomy of the previous related secure VMs allocation algorithms and approaches in CCEs. Then, we study and classify them to provide a framework of the attacker behaviours and the attack impact on different levels of the cloud architecture. Afterwards, we develop a secure VMs allocation model that aims to securely allocate the VMs while reducing the utilise PMs and minimising the VMs migration. As such, we develop and evaluate a linear programming-based algorithm called OSA to produce optimal secure VMs allocations using PuLP solver under a given set of constraints. Moreover, we propose a learning model framework that classifies the VMs into types based on their behavioural analysis.

2. We develop and evaluate algorithms that follow a stacking-based behaviour called SS and SRS to produce secure VMs allocation in CCEs. Furthermore, we develop a VMs migration algorithm that aims to enhance the proposed secure VMs allocation and maintain the VMs allocation secure as possible. As a result of this contribution, we publish part of our work that aimed to present the SRS and investigate the non-heterogeneous VMs allocation behaviours that are more likely to lead to a secure allocation [7]. Our results showed that the spreading VMs allocation behaviours yield more malicious allocations than stacking and random. Moreover, the stacking algorithms yield less VMs migration than random or spreading. Furthermore, the VMs arrival has a significant impact on producing secure allocation. In addition, the development and evaluation of the SS algorithm have been submitted for possible publication [8].

3. We develop a graph-based model and algorithms that consider the dependent VMs represented as graphs to examine their effect on the graph-based allocation. Consequently, we introduce two algorithms called GbSS and GbSRS, which allocates the VMs based on the graph-based model. We publish part of this work that aimed to present the GbSRS algorithm and represent dependent VMs allocation. This work aimed to evaluate the effect of heterogeneous VMs allocation, VMs migration and PMs usages

under the Fat-tree architecture [6]. This work showed that data centre architecture has a significant impact on random behaviours algorithm as it shows more secure allocations than anticipated. Also, the stacking behaviours produce more secure allocations than spreading or random behaviours. Additionally, the VMs migration is the worst under the spreading behaviours, while the stacking based is considered the best among the examined behaviours.

4. We extensively evaluate the proposed algorithms behaviours under various situations and scenarios to examine their behaviours on obtaining secure allocations. As such, we compare and evaluate the stacking, spreading and random VMs allocation behaviours. Moreover, we examine the impact of various properties that affect the proposed allocations behaviours, including different PMs heterogeneity levels, diversity of VMs demanded resources, various VMs arrival times and under different numbers of VMs according to their classified type.

1.7 Thesis Outcomes

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 algorithms 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. Additionally, the high available resources diversity of the PMs yields to produce more secure allocations as it offers more allocation options for the allocation algorithms and thus more flexibility.

Lastly, our stacking-based algorithms 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.

1.8 Thesis Structure

The following chapters of the thesis are structured as follows; in chapter 2, we will present background about cloud computing and the attack model considered in this thesis, which is the SCA. In chapter 3, we will introduce a taxonomy of the related works and domains tackling the SCA problem in CCEs. Subsequently, in chapter 4, we will develop and examine the considered secure VMs allocation model in CCEs. Afterwards, in chapter 5, we will present our secure VMs allocation algorithms and extensive evaluation and comparison with other VMs allocation behaviours. Then, in chapter 6, we will extend our algorithms by considering the graph-based VMs allocation; as such, we will develop, evaluate, compare algorithms based on different data centre structures. Finally, in chapter 7, we will conclude our work by summarising the key findings and propose possible future works direction.

Chapter 2

Background of Side-Channel Attacks in Cloud Computing Environments

Preface

This chapter will introduce a background about cloud computing and its related components and services, including the service models and the deployment models of the cloud systems. Moreover, it will explain the virtualisation technology that considers the core of the cloud systems, enabling the sharing of the resources among the cloud users. This sharing of cloud resources leads to security issues which this chapter will discuss as well.

Specifically, we will describe the SCAs in CCEs and generally under other domains. It includes the nature of this attack, how it performs in CCEs, and examples of this attack. Also, we show the severity of this attack on cloud users and how they can be impacted due to the amount of leakage of information that causes potentially harmful attacks.

2.1 Cloud Computing Environment

This section will present a brief description of the cloud computing environment, including cloud components and services. Moreover, we will explain the most common cloud service models and cloud service deployments. Further, we will explain the virtualisation technology, the core of the cloud system, and its vital role in sharing cloud environments' resources among cloud users. Lastly, we will highlight in general some of the security issues occurring due to the sharing of resources offered by the cloud system.

2.1.1 Cloud Computing Introduction

The cloud computing paradigm is the deployment of a broad range of services such as web applications, enterprise systems, or large-scale analytics applications. These services enable the abstraction, pooling, and scalable sharing of computing resources (e.g., CPU, storage), accessible across a network, by a pool of users. In other words, according to NIST: "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [68].

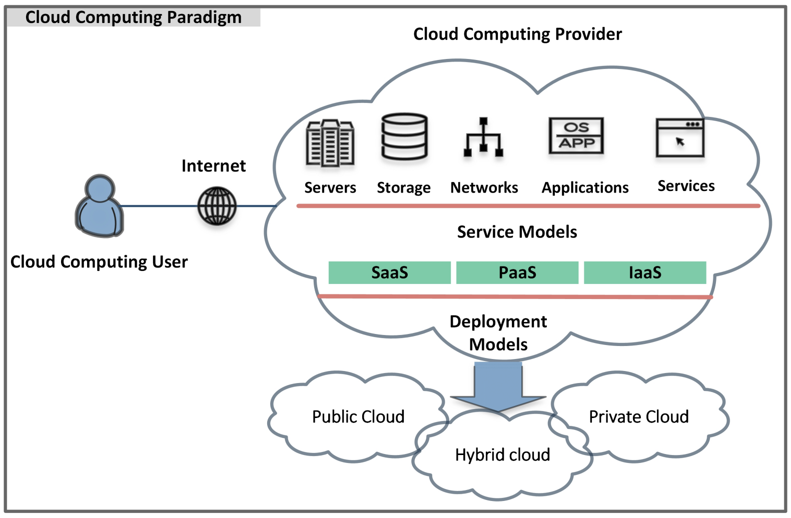


Figure 2.1: Cloud Computing Paradigm.

As illustrated in Figure 2.1, cloud computing users can utilise the computing resources or services offered by the cloud providers through the network and on-demand basis. These services include servers, storage, networks, applications and other services.

The utilisation of the services depends on the user needs; some users may only require applications for their businesses without the hustle of the underline infrastructure. For example, email-based applications, storage-based applications or other specified required applications. At the same time, some may need to customise their underline infrastructure while hosting their application on the cloud system. For instance, some users need to customise the operating systems that are hosting their applications. Thus, the CSPs offers service models that fit the need of the cloud users. These service models are Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS) and other services specified for a particular role [49]. The service models will be explained in Section 2.1.2.

Moreover, the CSPs classifies the deployment models into many types depending on the cloud users' needs. The deployment models are public, private, hybrid clouds and other types [79]. We will only describe the most common types in Section 2.1.3: public, private, and hybrid clouds.

The benefits of utilising the cloud computing services and models are due to the flexibility and agility they offer for business growth compared to traditional on-premiss computing. Cloud computing offers many benefits for users, including the flexibility of managing the resources while efficiently utilise them. Moreover, it maximises the efficiency of utilising the resources by offering the resources on-demand and as needed. Alternatively stated, building an on-premises infrastructure using the traditional methods leads to unnecessary cost and physical space allocation for the hardware equipments, including servers, storage and networks. In contrast, in cloud computing the resources, including hardware equipments, are utilised as a service remotely based on the user needs, thus reducing the cost of building and managing them. Also, the scalability of the resources in cloud computing systems is more agile and efficient than traditional on-premises computing. In other words, information technology systems are the enabler for the business to grow fast and agile; thus, the scalability of cloud computing offers the ability to add or enhance new resources in a short time and less cost than the traditional one [20].

On the other hand, the security of using cloud computing is debatable due to the security issues on them as they offer sharing of the resources among users. Despite the great benefits they offer in data availability by preventing them from being lost or damaged, they suffer from security issues related to data confidentially due to the sharing of computing resources. In Section 2.2, we will explain more about the security issues of utilising CCEs.

2.1.2 Cloud Computing Service Models

In this section, we will explain the cloud service models, including the most common types: SaaS, PaaS, and IaaS. The service models identify the responsibilities of the resources managed by the cloud provider and those managed by the cloud users. Moreover, we will highlight some of the modern service models which considered an extension of the mentioned ones.

Software as a Service

The SaaS are the most common service models utilised as they offer applications as service to the users without the need to manage them or install them on local devices. There are well known commercial applications utilising this model, for example, Slack[1], which is a communication platform utilised by institutions for better communications and efficient knowledge management. Also, Google Workspace[2], which is a group of applications for particular purposes, including emails other applications.

The advantage of utilizing this model is to reduce the time needed for managing and maintaining the application infrastructure because the CSP is responsible for managing them on all levels. On the other hand, data security is a matter of concern in this model. Because the underline infrastructure of the data is managed and processed entirely by the CSPs, which may require moving them from one location to another for better functionality, it may relocate sensitive data leading to Service-level Agreement (SLA) breaches.

Platform as a Service

The PaaS is similar to SaaS but provides additional components to manage the applications and customise them accordingly. Thus, this model provides a framework for building and managing particular applications with their related components without maintaining the underlying infrastructure, including operating systems, servers, storage, and network. For example, Google App Engine[3], which focus on building a platform for developing and customising

[1]Slack: `https://slack.com`
[2]Google Workspace: `https://workspace.google.com`
[3]Google App Engine: `https://cloud.google.com/appengine`

applications, and AWS Elastic Beanstalk[4].

This model has the benefits of providing a development environment for the hosted applications to allow the cloud users to customise and enhance their applications on the same platform. Thus, this model offers a more scalable environment for application hosting and customising than the SaaS model.

Infrastructure as a Service

The IaaS is the most scalable service model as it offers a virtual infrastructure for cloud users to allow them to manage their infrastructure and customise it based on their needs remotely through the internet. Therefore, this model allows the users to have the ability to customise and access the underline infrastructure, including operating systems, servers, storage, networks and the virtualisation layer. Otherwise stated, the CSP in this model allows users to maintain their computing resources by granting them the ability to customise their infrastructure constraints, for example, customising their security controls over the computing recourse.

The advantage of utilising the IaaS model is the scalability of customising the required computing resources while maintaining them based on the user needs. Moreover, the control over the underline infrastructure and the data processing on the computing resources allows the user to design their security controls based on the level of data sensitivity. However, and even with these offered abilities, the security threats of this model exist in different forms and levels. For example, the threats may occur from the other virtual machines that share the same physical computing resources in the public cloud deployments model, which we will discuss later in this chapter. Microsoft Azure[5] and Amazon Web Services[6] are well-known CSP that offers this type of service model.

X as a Service

In recent years, the services migrated or utilised to the cloud systems have grown and therefore offered as a service by the cloud providers. The services

[4]AWS Elastic Beanstalk: `https://aws.amazon.com/elasticbeanstalk`
[5]Microsoft Azure: `https://azure.microsoft.com`
[6]Amazon Web Services: `https://aws.amazon.com`

include network, storage or application, and advanced services such as a disaster recovery solution. Thus, the services offered by CSPs is generalised to Everything as a Service (XaaS), which refers to any tool or technology offered as a service, to describe and formalise the diversity of services models. The XaaS help the business to reduce their operations and service costs compared to the three models. Because it allows cloud customers to buy the needed services without the hassle of paying the subscription for the entire platform, it helps the business adopt new trends by speeding up the provisioning of new applications and technologies it needs [27].

2.1.3 Cloud Computing Deployment Models

This section will provide an overview of the main cloud deployment models: public, private, hybrid, and other deployment models. The deployment models is an indication of the users who can access the cloud resources.

Public Cloud

Our work, in this thesis, is based on the public cloud deployment model where any user can access it and utilise its resources. According to NIST [68], the public cloud is open for general use by any user, and it can be managed or owned by a private or public origination. For example, a government or business.

Private Cloud

On the other hand, the private cloud is dedicated to a single organisation or particular group of users. However, this deployment model can be managed by a third party origination but will be utilised for private institutions.

Other Deployment Models

Many deployment models emerged for the past years that are specified for groups of users who share the same interests, called community cloud. Alternatively, in some areas, a deployment model is a hybrid between the public and the private cloud based on the user needs.

2.1.4 Virtualisation Technology

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. In other words, virtualisation is the enabler for sharing the physical computing resources among cloud users efficiently and on-demand basis to allow the high utilisation of the physical resources. As shown in Figure 2.2, virtualisation enables a group of virtual machines belonging to different users to share the physical machines while running separately. In traditional on-premises computing, a physical machine will be dedicated to a single-purpose application, while in cloud computing, many applications belongings to different users can be run on a single physical machine.

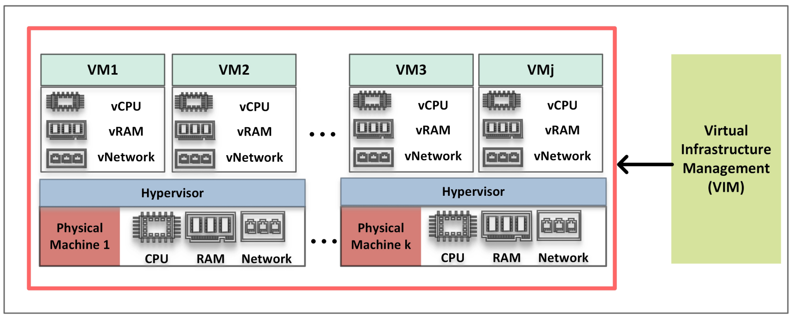


Figure 2.2: Virtualisation and VIM in cloud computing.

The benefits of virtualisation can be classified into three aspects; isolation, consolidation and migration of workloads. First, isolation allows multiple users to run their virtual machines with a specific purpose into shared physical machines separately and securely. Otherwise stated, the processes of the virtual machines are isolated, acting as if they running on single dedicated hardware. Second, consolidation, which means consolidating different workloads into a single physical computing resource. For example, two processes from different virtual sources can be run together on a single physical CPU but separate threads or cores based on the resource's allocation requirements. This feature allows to manage the hardware resources efficiently and reduce the resources wastage and the cost of implementation. Third, migration refers to moving the allocated virtual resources from one physical resource to another. For instance, migrating a user application to run on a different CPU, or network, or migrating the entire virtual machine to a different physical machine. This feature keeps the high availability of the cloud resources against any system failure or disruption [84].

Types of Hypervisor

As shown in Figure 2.2, the hypervisor is software that allows the virtualisation technology to be applied. As such, the hypervisor is responsible for allocating a PM's physical resources to multiple VMs virtual resources, for example, allocating the physical CPUs, RAMs, networks and storage.

There are two types of hypervisors, the first one, called type 1 hypervisor, which is the most common method by implementing the hypervisor software on the PM as a bare metal server. Then, after implementing the software, or hypervisor, it will allow separating the physical resources of the PMs into multiple divided resources, which can be utilised and shared by VMs. There are many commercial and open-source CSPs of the hypervisor that uses this implementation method, for example, VMware ESXI [29] and Microsoft Hyper-V [51]. The second one, called type 2 hypervisor, depends on implementing it on the OS directly instead of the bare metal server. In other words, the hypervisor software is installed as an application, and then it can allocate computing resources from the hosted OS to the VMs created by this hypervisor, for example, VMware Workstation [90]. It is less common because it has poor performance than the first type, as it depends on the hosted OS, not on the bare metal computing resources, i.e., direct connection to the PM [32]. In the thesis, we only consider the first type of hypervisor.

Virtual Infrastructure Management (VIM)

As stated, the hypervisor is responsible for virtualising the physical resources of a PM to be shared by many virtual instances. In comparison, the Virtual Infrastructure Management (VIM) manages the cloud system's virtual infrastructure resources, including the VMs allocations and migrations across the virtualised data centre. As shown in Figure 2.2, the VIM is a component that has visibility and control over the entire virtualised infrastructure of the cloud environment. The VIM is responsible for allocating the VMs, or virtual resources, into a specific PM according to the defined allocation process implemented in the cloud system. It is also responsible for migrating the virtual machines, or other virtual components, from physical entity to another. As such, the VIM can manage and control the entire virtualise computing resources of the virtual infrastructure such as VMs, network, storage, and the virtualisation layer. This holistic control over the infrastructure contributes to optimising resource utilisation and mentoring from potential overhead performance [86].

An example of the VIM is the OpenStack platform. OpenStack is an open-source VIM platform designed to manage, control, and provision physical computing resources as virtual resources under a single CSP domain. It includes managing the life-cycle of virtual resources by controlling the allocation and scheduling of the virtual resources on the entire dedicated virtual infrastructure[7].

Containerisation

The virtualisation technology that we defined earlier focuses on virtualising the physical resources, by the hypervisor, into a set of VMs sharing the underline physical resources. On the other hand, another technology works similarly to virtualisation but on a different level, called containerisation. However, instead of virtualising the physical resources of the PM, it will virtualise the processes of an OS and allow multiple applications with their dependencies to work concurrently and share the same OS. The containerisation technology is the packaging of applications with their required code and dependencies running on OS, called containers. The containers can be running on a single OS sharing the processes while working separately and efficiently [56].

For example, Docker is a containerisation open-source platform that allows the application with their dependencies and libraries running on OS kernel as a container. It has many features that allow the development of scalable, agile, and automated application containers[8]. Another example is Kubernetes, an open-source container that allows the automation and management of software development as a container[9]. The two examples work differently but achieve the same goal of containerising the applications into the shared OS.

[7]OpenStack: `https://www.openstack.org/`
[8]Docker: `https://www.docker.com/`
[9]Kubernetes: `https://kubernetes.io/`

Adopting containers for executing applications has many benefits than adopting applications running on top of VMs, for instance, the speed of development. The containers need only an OS to host the application process without worrying about the hustle of the underline layer of physical infrastructure and network connectivity. In comparison, deploying the application on top of a VM require more overhead deployment and licencing cost that can be avoided. Moreover, it allows smooth portability and integration of the application containers to another platform on the cloud system. Because each container has its application code and dependencies packaged in the containers, it can run with compatibility issues[10].

2.2 Security Issues in Cloud Computing

The paradigm of cloud computing enables multiple users to share a common computing platform where resources are dynamically available. This sharing of resources invariably means that a PM, or any other hardware resources, can potentially share its resources among a set of distinct users (or VM) in what is known as VMs co-location, or resources sharing. As a consequence of the physical resources sharing, 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 range from data confidentiality breaches to denial-of-service attacks [52].

2.2.1 Traditional and Cloud Computing Security

The business migration to cloud computing brought many security challenges on data security, multi tenancy, and on security standards of cloud providers. For instance, the lack of data processing visibility, location and control, while in traditional on-premises data centres, the data processing and location are known and under control. Moreover, the lack of trust in the security operations conducted by the CSPs, for example, unpatched systems or misconfigured ones, could lead to a potential data leakage over the processing of the shared computing resources. In contrast, in the traditional data centres, the system

[10]IBM Container: `https://www.ibm.com/cloud/learn/containerization`

owner controls the security operations and the configuration of the recovery process. However, the benefits of relying on CSP security are reducing the operational cost and overhead of human resources required for conducting security operations. Furthermore, on the systems level, the sharing of commuting resources, such as PMs, or processes, multi tenancy, is leading security threats either on PMs level, network or in storage. Consequently, the benefits of migrating the computing resources to the cloud systems outweigh their security drawbacks due to the sharing of resources [47].

2.2.2 Threats to Cloud Computing

This section will highlight some of the top threats that are challenging in the cloud environment nowadays. The threats are obtained from Cloud Security Alliance (CSA) group, which focuses on providing the institutions with the recent risks and vulnerabilities facing the CCEs[11]. The CSA report classifies the threats into either internal or external threats depending on the threats' nature. For example, the internal threats include data breaches, careless data handling, misconfiguration or communications ports or abuse of systems such as VPN. The external threats include cloud accounts hijacking and compromise, abuse of access given to third-party vendors, which leads to an insider attack, or the threats that arise from former CSPs who gain intimate knowledge about cloud configurations. We will state these threats in detail in the following parts alongside the proposed mitigations of the threats.

Internal Threats

The internal threats include:

- Data Breaches: An insider attack could gain users data due to the vulnerability of misconfiguration access control and the system's firewall from the server-side. Moreover, on the hypervisor level, the default setting of the hypervisors trust mechanism allows for this kind of data breach.

[11]Cloud Security Alliance (CSA) Report: `https://cloudsecurityalliance.org/artifacts/top-threats-egregious-11-deep-dive`

- Internal Systems Abuse: The poor management of internal systems such as VPN or any network-based function leads to unwanted privilege breaches. This misconfiguration of privileges gives the attacker visibility over cloud systems configurations and data.
- Data Processing: The careless data handling when stored on cloud databases without the acceptable methods for securing these data while processing or the systems they reside on. This vulnerability exposes highly sensitive data of the originations, leading to critical threats.
- Ports and APIs: The access misconfiguration, the lack of proper cloud architecture solutions and ports communications lead to unauthorised access to services and systems. The vulnerability over APIs access leads to exposing the customer data processing.

External Threats

The external threats include:

- Intimate Knowledge of CSP: The vulnerability of relying on the default setting of cloud systems could lead to security breaches. For example, a former CSP employee with intimate knowledge about the configurations and architecture of the cloud systems leads to unauthorised access or data leakage.
- Third-party Access: Relying on third-party applications for some functionality of the provided service, such as web applications, with poor control leads to data breaches. The vulnerability of inadequate access control management and trust with third-party providers leads to critical data leakage.
- Accounts hijacking: The poor identity management for cloud-based service leads to customer accounts and data misuse.
- Malicious Containers: The inadequate anti-malware executions over the cloud containers allow some malicious users to utilise them for crypto-currency mining.

Threats Mitigation

According to the CSA report, the mitigation of the mentioned threats includes three main parts: preventive, detective, and corrective mitigations. For instance, they propose several actions on either the internal or external classified threats on data breaches. The preventive actions include monitoring and controlling the data flow in the cloud systems. The data flow includes securing documenting the process and inventory of the life cycle of the data from data creation to disposal. The detective action includes a periodic data risk assessment to identify the access rules of the data, who and on which level the data manipulating, storing and disposal of the data. The data breaches' corrective actions include the management of cloud architecture to control the life-cycle of the data processing over cloud systems. Additionally, the identification process of misconfiguration for the cloud systems leads to unauthorised data access. Furthermore, the training of systems and applications admin and raise the awareness about the critically of data that processes within the cloud systems for better management of them.

Moreover, to prevent abusing the access over systems and applications, the security requirements identification is critical for defining the limited use of the applications and controlling the misconfiguration. As a detective action for these threats, applying the periodic testing of the applications and systems against the integrity of their data processing, availability, and confidentiality. There are more in the CSA report, which includes case studies of real-world organisations with their security challenges and how these challenges are mitigated.

2.2.3 The Importance of Could Computing Security

The importance of securing the systems of the CCEs is related to the negative impact on organisations after data breaches or systems failure. The security breaches on cloud systems could potentially lead to a significant loss for the originations hosting their data and applications on the cloud. For example, the breaches of data or misusing them by internal actors lead to a financial settlement, loss of customers and negative repudiation, hence, losing the business. As such, the need for applying security controls, compliance and risk assessments over cloud processes, data and applications is a critical act. As such, cloud security controls and standards must contribute to organisational growth and continuity. Also, they protect cloud systems against breaches and failure by establishing disaster recovery configurations. Moreover, they contribute to patching the misconfigured systems and processes and preventing unauthorised access by conducting risk assessments.

2.2.4 Deployment Model Threats

The threats are different based on the deployment model of the cloud system that consumers utilise. Specifically, in the IaaS deployment model, security threats arise on VMs, virtual networks, PMs, or hypervisor levels. While in PaaS, and in addition to the inhered security threats in IaaS, the security threats focus on the service architecture or the API related components. The SaaS threats arise on web application components that causing vulnerability or interruption to the web application configuration [10].

IaaS threats

The IaaS threats cover the VMs level, including malware or viruses that can affect the systems and need to be maintained by the cloud user. Moreover, the VMs image threats are the ones that cloud users utilise to either create a model VM or duplicate an existing one without creating a new VM every time. Alternatively stated, the VM image is a configurable model of a VM that can be duplicated and utilised multiple times by the cloud user. For example, if the user wishes to create many database VMs, only one image VMs can be created and configured with the OS, security standards and any other configurations. The rest of the VMs can be duplicated from the image VM, and only a few changes can be made to them, such as changing their IP address or domain name. Thus, this image is a great security concern if it is compromised or injected with viruses, as it can easily cascade to many VMs.

Moreover, the threats on virtual networks are the ones that can be exploited because of the sharing of network resources among cloud users. For instance, the VMs sharing the same PMs resources often share the same network physical resources, potentially leading to threats on the network levels. The threats could occur by deducing information about network traffic using software based on tracking such data, which leads to network-based attacks.

Furthermore, the PMs hosting the co-located VMs and allowing computing resources, such as RAM and CPU, to be shared among users are potential threats. These co-located VMs are not isolated physically but logically, allowing for potential threats based on data leakage on the cache-based or RAM pages level. Otherwise stated, the shared VMs are utilising the physical computing resources simultaneously without physical isolation. However, this lack of isolation leads to security threats on these shared physical resources, which is a leakage channel that can be exploited by a side-channel attack, which we will explain in detail in Section 2.3. In addition, the threats on hypervisor levels, or the virtualisation technology, are responsible for allocating the resources and sharing them among cloud users. The threats on this level may lead to compromise; all the VMs are in control under the hypervisor, which makes it critical to secure.

PaaS and SaaS threats

The PaaS and SaaS threats are mainly related to the application platform configurations, specifically to the service-oriented architecture or API configuration. Specifically, the site script related attacks such cross-site scripting, injection attacks such as SQL injection or threats related to authentication between entities of the application platform. Moreover, the APIs with a lack of security controls and access potentially leads to exploit threats to this level of the cloud system. Moreover, the threats on data location, integrity, availability and confidentially are the most concern on these two levels.

2.3 Side-Channel Attacks in Cloud Computing Environments

This thesis focuses on the security threats on the IaaS model, specifically on VMs sharing the PMs computing resources. Therefore, we will present the SCAs background that resulted from sharing VMs on the same PMs, thus; sharing the same physical resources. In other words, VMs co-location, though enabling efficient resource sharing, is creating unwanted side channels, which can be sources of potential 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 [97]. SCAs will have an impact extending from the application level to the hardware level and becoming more prevalent due to the range of side channels [14, 82]. In this section, we will explain this attack in more detail.

2.3.1 SCA Overview

Generally, in the traditional attack model, two machines are communicated through an encrypted channel, and an attacker aims to compromise these encrypted communications. As illustrated in Figure 2.3, we have machine A and B machines exchanging encrypted messages, and a malicious user tries to capture and decrypt the communication. The malicious user's focus is the message sent between the two machines and the operations related to decrypt this message.

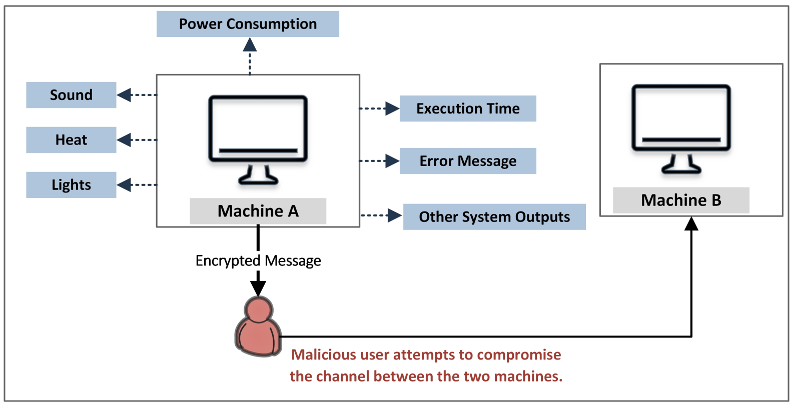


Figure 2.3: Overview of Side-channel Attacks.

Unlike the traditional attack model, the SCA aims to compromise the target system through the information gained from this system's "normal" outputs, not only from the message sent by this system. As shown in Figure 2.3, each machine or system is designed with certain executions protocols that produce a normal system output. For instance, a heat from a machine, light or sound when certain tasks executed, and it could be an error message appears once a wrong input was entered. Moreover, the malicious user could monitor and analyse the execution time of certain tasks or the power consumption to profile and compromise the target system. The profiling process includes gathering data related to systems sharing the same computing resources.

These outputs considered properties leaked while executing tasks, which are considered a normal output and are not considered a security threat. However, recent researchers have become interested in the possibility of attacks that utilise these system properties normal outputs, or side-channels.

Side-Channels Definition

We can say that the side-channels are normal outputs, unintended, from a system or machine. Malicious users utilise these outputs to understand the system behaviour and therefore perform an attack. The nature of these attacks is not related to the encrypted message between two parties, or machines but rather the amount of data gained from the side-channels outputs.

Side-Channels Attack Definition

From the above, we can say that the SCA is the threat based on the leaked data gathered from the system implementation outputs rather than flaws in the implemented algorithm. Therefore, generally, the need for security system implementations is as important as the need for secure algorithms.

2.3.2 Classifications of SCAs

The SCA is classified into three types based on the nature of the attacks and the way of attacking the target system [97]. The first is based on the attacking nature, either active or passive attack. The second is based on the source of the side-channels properties that exploited physical or logical property. The third is based on the location of the attacker from the compromised system.

Passive or Active

The passive SCA attacks refer to the attacks that are not noticeable or cause interference to the execution of the target system. It mainly focuses on gathering private information from the target system during the execution from the leaked side channels. In contrast, the attacker may need to modify the current system to gain the required attacking goal in an active SCA attack, and this modification can be noticeable in the system behaviour or performance. This type of attack depends on the impact of the attack on the target system. For example, a Denial-of-Service (DoS) attack is considered an active attack because it leads to a disruption to the system's availability. Therefore, its effect can be seen clearly on the target system, while the passive attack can not be noticed or captured.

Physical or Logical Property

The physical or logical property refers to the source of the side-channels leakage, leading to the attack. For instance, the physical property could be the monitor and analysis of the power consumption of the target system or the physical movements of a mobile device. The logical property could be the target system's data usage statistics, including the computing resources usages.

Attacker Location

The location of the attacker refers to the proximity to the target system. For instance, a local attack (insider attack) refers to the attack that occurs while the malicious user is temporary in possession or close to the target system. In comparison, the remote attack is based on software execution on the target system to capture target information.

2.3.3 Attacker Behaviour and Scenario

In this section, we will describe the scenario of the SCA and the attacker behaviour. Specifically, we only focus on the passive attack aspects as part of the attack model considered in this thesis. The passive SCA attack has three main components: target system, side-channels, and malicious user [82]. The target system refers to the systems that represent an interest to the malicious users because of the value of their data or the impact on them when a failure occurs. Moreover, during the execution of particular tasks on the target systems, the side-channels will start leaking potential important information, which will be interested in the malicious user. Furthermore, the malicious user will observe and capture this important information from the leaked side-channel and plan the attack.

We can classify the attacker behaviour into two main steps; training step and attack step. In the training step, the attacker starts profiling the events that have been captured from the target systems. In other words, the SCA depends on cumulative gathered information from the target system at different times. Thus, collecting these data in order to profile the target system behaviour. During the collection process, the malicious user may utilise a machine learning-based technique to classify the collected data into meaningful information. This information includes usage information about a certain aspect of the systems, for example, the execution time of some tasks at a particular time. The malicious user will be able to build a profile or template from the events that potentially lead to an attack, which is the attack step. This step depends on repeating the previous steps to enhance the created profile and its accuracy about the target system behaviour, hence, increasing the chance of an accurate attack model.

2.3.4 SCA in Cloud Computing

The SCA in cloud computing follows the same behaviour of other system models, which focuses on capturing the information leaked from the target systems and formulating the attack based on them. In this work, we are focusing on SCA resulting from sharing the resources between cloud users. Specificity in CCEs, as shown in Figure 2.4, the malicious user will attempt to co-locate their VMs (the red one) on the same PMs of the target one (the green one) to perform the attack. Changing the VMs location from one PM to another could occur by performing normal unsuspicious actions to the CSP, including increasing VMs current resources, many VMs creation or deletion at a short time. These actions could lead the CSP allocator to reschedule the current allocation of the VMs to different VMs, which increase the chances for the malicious user to achieve its target.

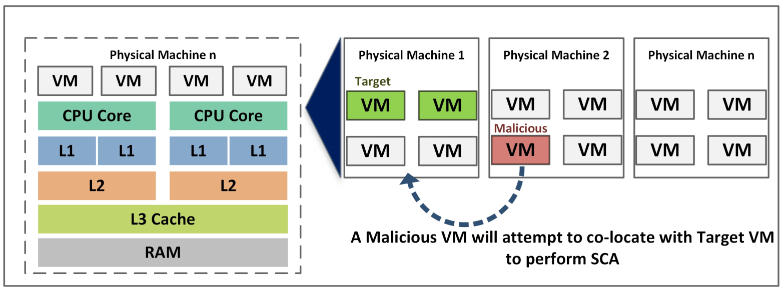


Figure 2.4: Side-channel Attacks in Cloud Computing Systems on Shared Cache Level.

After the co-location is achieved, the malicious user will try exploiting the shared resources. As shown in Figure 2.4, the malicious user can perform SCA from the VM level by merely sharing the same physical resources with the target VMs, in this case, sharing the CPU cores or CPU caches. This type of attack behaviour is called cross VM SCA, which focuses on exploiting the weaknesses on the shared computing resources that leak private information, such as the execution time of other VMs. The execution time analysis attack, known as cache-based side-channel attacks, utilises the sharing of L2 and L3 caches among hosted VMs or threads. It is performed by measuring the execution time of the load operations of the shared caches on the physical machine. If a specific operation took a long time to load compared to the other operations, the malicious user would profile this operation for future classification. The response time of operations could be due to encrypted one or running highly intensive tasks requiring high response time.

As such, the malicious user will be able to extract information from collected bits even if they are captured at different times to profile the target VMs behaviour and therefore compromise their VMs. In the following section, we will present some examples explaining exactly how some SCA are performing in a shared environment.

2.3.5 The Importance of SCAs in Cloud Computing

The SCAs are critical due to their passive nature, where the malicious users can know features about the cloud systems from data gained through side-channels leakage. These features are extracted from the leakage channel of the hosted systems, and the data extraction process is passive and hard to be detected.

Moreover, the demands over cloud services have expanded over the past years, where the business requires more of its services running on cloud systems. In addition, this attack takes advantage of the cloud systems' sharing capability, which leads to a potential threat over this type of attack. As we explained earlier, in Section 1, the data breaches in cloud systems lead to significant losses for the business owners legally and financially. Consequently, the impact of SCAs can affect the application level and the hardware level of the target systems, it is becoming more prevalent due to the various channels that can be compromised. Therefore, due to its passive behaviour, sharing capability, and increased demands over cloud service, the defence against SCA threats is becoming more critical [14].

2.3.6 SCA Examples

In this section, we will present some of the examples related to SCA in shared computing environments.

Prime-Probe

A well-known form of cache-based attack is called Prime-Probe attack. It performs in two stages; the first stage is prime, which means that the attacker VM will perform a read-memory operation on all memory blocks that lead to placing all the attacker data in the cache. The target VM will perform a heavily encrypted operation at a specific time, requiring utilising the full cache. In this case, the attacker data will be evicted from the cache and replaced with the target VM data. The attacker will repeat the read-memory operation and calculate the time if it takes longer; it indicates a sensitive operation executing at this time [95].

Power Consumption Analysis

The power consumption analysis depends on tracking the power footprint of a VM to know its level of security and weaknesses. Moreover, the extracted information will help to define the vulnerable state of the VM. If the power consumption is significantly high at a precise moment, it will indicate that the

VM performs a sensitive or essential operation, and the attacker can interrupt or extract this information [48].

Meltdown and Spectre

Another form of side-channel attack is called Meltdown and Spectre. These are SCAs recently discovered that take advantage of the out-of-order execution and speculative execution of the most modern processors to allow users to overcome memory isolation and read the entire shared memory from a single machine. Firstly, the Meltdown attack, the main reason for this attack is the out of order executions of CPU, which is a feature deployed in modern processors to maximise the performance of processing units. The basic idea of out-of-order execution is to allow the processor to run some instructions in an out-of-order manner instead of waiting for the previous instruction to be ready and its data available for execution. Therefore, for example, if we have an (IF) statement followed by an instruction (ADD operation), the processor will execute the (ADD operation) in the background and store it in the CPU cache without waiting for the (IF) statement to be valid, to have a better response time. Secondly, a Spectre attack will utilise the CPU's speculative execution for operation and try to predict its outcome using branch predicting. It will guess the outcome ahead for repetitive operations and store it in the CPU cache to better execute it [57, 62].

PORTSMASH

PORTSMASH is a timing-based side-channel attack that utilizes the ports contention during instructions execution on modern processors. This attack relies on exploring the execution time by measuring the time of instructions execution among shared threads. Modern processors have several techniques for executing the instructions to maximize the performance and throughput, such as simultaneous hyperthreading (SMT). The notion behind SMT is to allow the CPU to execute instructions from a different thread and from different logical cores that are all located on the same physical core. The execution of the instructions is performed concurrently and is in an out-of-order manner. PORTSMASH attack happens at a single physical core with shared logical cores that utilizes the SMT execution. SMT allows instructions to execute in an out-of-order ordered manner from different threads. The following example illustrates the idea of PORTSMASH. Assume two users A and B shared a logical core on the same physical core. If A executes instructions on a dedicated port, port-0, and B executes instructions on another port, port-1. Then there will be no port contention, thus, no PORTSMASH attack. However, if A and B alternate between port-1, then the time of execution can happen in this scenario. Moreover, some instructions execute only on a specific port. For instance, latency instruction NOP-0 only execute on port-0. Also, NOP-1 is only executed on port-1. Therefore, if A use port-0 for sending a zero bit and port-1 for sending one bit. Then B uses port-0 and measure the time-0, then uses the port-1 and measure the time-1. If time-1 is greater than time-0, it is one bit; otherwise, it is a zero bit, which means that if the execution of the instruction takes relatively much time than usual, it is an indicator that the other thread occupying the port at this time, which is considered a leakage channel on the ports level. This type of information helps the attacker profile the shared port activities and orchestrate the attack using this leaked information [9].

Summary

This chapter presents background about the cloud computing environment, including the service and deployment models. Moreover, we present an explanation about the virtualisations technology and its crucial parts that contribute to maximising the resources sharing efficiency in the cloud system. Furthermore, we explain the security issues in cloud computing systems resulting from sharing computing resources among cloud users. Lastly, we explain the attack model we consider in this work: the SCA, generally and specifically in cloud systems.

Chapter 3

Related Work and Domain of Thesis

Preface

Generally speaking, in cloud computing, defending against SCAs has been an exciting topic for researchers over the past years. However, securing the VMs allocation is challenging because of the impact of allocation, or migration, on aspects unrelated to the VMs security during the allocation, for instance, resources constraints.

This section aims to review the 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 is the allocation algorithms based on VMs clustering, which includes solutions that aim to find a secure VMs allocation by grouping the VMs according to defined parameters. The second domain focuses on profiling the VMs based on security compliance requirements. This allocation method ensures that each VM hosted on the same PM share the same security compliance level. The third domain highlight the solutions that focus on allocating the VMs, for a defined time on selected PMs, to reduce the amount of sensitive data leakage through the side-channels. The fourth domain focuses on developing algorithms that manipulate the cloud system's scheduling component to allocate secure VMs. Hence, reducing VMs malicious co-residency while maintaining resources and other constraints. The fifth domain focuses on studying the behaviour of allocation policies by examining the ideal situations of an allocation under specific constraints by following an optimisation-based approach. The last domain focuses on the secure VMs allocation on the hardware level of the cloud system by performing a modification on the hardware components.

Moreover, this section will discuss the domain that this thesis focuses on to obtain a secure VMs allocation by presenting a study of the aspects affecting the secure allocation in CCEs. These aspects include the attacker's behaviour and the attack impact. Furthermore, the section will propose the countermeasures proposed by the previous solutions based on the type of attack and the level of virtualisation. The levels of virtualisation compromised by SCAs include the VMs, PMs, and the hypervisor.

3.1 VMs Clustering

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 [61], they proposed a VM placement algorithm that aims to reduce the chance of co-residency with malicious VMs. When a new VM request initiated, it will go through two steps, groups selection and PMs selection. The groups selection, which is selecting the VMs group fitted for the current requested VM using the Round Robin policy (RR). Meaning, the selection process will select the VMs alternatively until reaching the last VM, then repeat the process until all the VMs selected. After the VMs selected and defined into a specific group, the PMs in the group will be selected using the First Fit policy. Their goal is to create randomness by formulating the VMs allocation to selected PMs using the grouping technique. They mentioned that the available number of groups has a significant impact on reducing the chance of co-residency.

In addition, [93] proposed a placement algorithm that depends on the network dependency of the VMs. Meaning, if the VMs share the same network dependency, these VMs are grouped to be allocated on the same PM. Moreover, they quantify the risk of each VM depend on its type, i.e., hosted application. Thus, the security risk of application is considered in their work. They quantify the risk of both VMs and PMs based on the US national vulnerability database (NVD) [17]. The vulnerabilities are a set of software and configuration settings that are potentially affecting the cloud environment. The NVD set is valuable data for quantifying the risk and defining a particular risk score of VMs and PMs. The risk score, in their work, is based on the connection dependency relationship of the upcoming VMs with the existing VMs. If there is no network dependency between them, the score will remain the same. Eventually, the placement of high-risk VMs will be on the PMs scored as high-risk PMs.

Furthermore, [16] proposed a framework for placing the VMs into PMs under conflict-free constraints. They assume that each VM has a conflict or non-conflict with other VMs and, the PMs grouped based on the VMs that resides on them. The conflict-free attributes of each VM hosted on a PM will define the grouping attributes of that PM. Thus, their method ensures that each group of PMs share the same attribute value of conflict. They assume that this attribute value will be assigned by either the cloud provider or the user. The cloud provider assigns it depending on the sensitivity of each VM's data processing on the cloud. At the same time, the user can assign it based on specific security requirements, for example, a geographic constraint. The mechanism of the method starts by assigning an attribute value and the level of the data sensitivity of the VM and afterwards, grouping the VMs that share the same attribute value and then placing them on specific groups of PMs with the same conflict value.

Similarly, in [92], their solution was initiated by creating an isolation group for VMs. Each group shares the same aggressive conflict of interest (ACIR). Then, place a VM in one of these groups if there is no conflict by most VMs in this chosen group. It is a VM-to-VM relation calculated by capturing the behaviour of the VMs. The strength factor that defines placement security depends on having fewer conflicts VMs on the selected group. The more conflicted VMs co-located on the same group, the less secure the allocation will be. They require that an agent be installed on all physical machines to capture the behaviour of the VMs to aid in calculating the ACIR between users.

In addition to this, [5] introduced a framework for VMs allocation on the cloud that aims to group VMs based on constraint requirements. These constraints consider multiple factors, which are cloud provider constraints, customer risk and security requirements. The first constraint is allocating the VMs that share the same security constraint on the exact PMs, or groups of PMs share the same requirements. The second one is considering the VMs reachability requirements with other VMs in the same VM group. They strict the VMs reachability by specifying a maximum distance between each connected VMs within the same group. These VMs may be allocated on different PMs, thus specifying this control reduces the network traffic threats. Lastly, they define a constraint to the communication between VMs in different security groups. If a VM wishes to communicate to another VM classified into a different group, this communication is conducted under a policy for access control.

Additionally, the work of [60] introduced a mechanism to identify the dependency between VMs using the network connection information collected from the VMs operating system. According to the collected data, each VM gave a vulnerability score based on the NVD [17]. They define grades for each VM risk, which if it becomes more than zero, the VM is high-risk according to the defined threshold. Based on the obtained score, the VMs with a high dependency network will be placed in a high-risk group of VMs, and the VMs with a low dependency network will place in a low-risk group of VMs. In other words, their work classifies the VMs according to the network reachability requirements.

Further, [1] proposed a VM allocation that allocates the VMs into specific PM if there are no adversary VMs on the selected PM. They assume that the adversary VMs list submitted by the legit VMs to the CSP to allocate legit VMs according to this constraint. The adversaries, in their work, are not specifically malicious users; instead, it is a security requirement to ensure that the legit VMs are allocated away from adversary VMs. For example, the adversary could be a list of VMs that are geographically different or following different security standards. Moreover, their algorithm will migrate VMs to allow more space while considering the resources of the VMs requesting the migration.

Lastly, [64], proposed a group-based allocation policy to create a balance between optimizing the resources and obtaining a secure VMs allocation. Their solution is focused on establishing group instances for the VMs, and these groups have specific requirements such as group size limit or resource availability. The group size 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. Moreover, their work allows the CSP to adjust the balance between obtaining a secure VM group or an efficient resource of a VM group. This VMs allocation approach mainly depends on various factors related to the security requirements of the VMs and the availability of resources in the cloud system.

3.2 Security Profiling and Compliance

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. These allocation methods ensure that each VM hosted on the same PM share the same security compliance level. The security compliance of VMs is the security specification and configuration that must be applied, within a particular domain, to each VM sharing the computing resources with other VMs.

The work in [12] introduced a placement scheme that validates the level of security of each new arriving VMs. The VMs must comply with specific compliance regulations and they must adopt the Health Insurance Portability and Accountability Act (HIPA) to comply with their algorithm [28]. This act investigates the level of security for the VMs, PMs, networks and storage as primary components. Therefore, it will go through this compliance for a new VM request to ensure it fits the security group's level.

Furthermore, [3] utilises a profiling model for detecting the VMs with malicious behaviour, then invoke the allocation or reallocation process accordingly. This model, called the SIR model, focuses on analysing the spreading of infectious diseases in the computational biology field. Based on the proposed profiling, they present two processes of VMs allocation and migration. These are a secure-based selection (SBS) method and secure-based placement (SBP).

They check all PMs against the selected VMs, and if a PM has the same security level as the candidate VM, and the power consumption of the PM will not exceed the maximum threshold after migration, then the VM should be migrated. Furthermore, they assume that the live migration between PMs is available to allow the migration of VMs while detecting the PMs overutilisation and underutilisation are enabled. Moreover, a selection method to defines the VMs that are a candidate for migration.

Also, [19], proposed a method to allocate the VMs by maintaining the same security standard level as the other shared VMs. Furthermore, the upcoming VMs will not compromise the security metrics of the existing VMs. For example, if a group of VMs comply with the ISO 27004 standard as the upcoming VM, this allocation will occur. Additionally, they create a notion of the randomness of the VMs placed on a PM by periodically changing the selection and placement process. For example, one process called (RAND FREE) handles the requests for placement randomly, and for each VM request, the algorithm will choose the least loaded PM from the compatible list of PMs. They defined many policies to allow the allocation algorithm to select PMs randomly between them.

Moreover, [73] proposed an algorithm that depends on a user security profile defined by the users. The security profile specification will integrate with the existing placement constraints to improve the placement process's security and group the VMs to the suitable PMs. The profile constraints inputs are gathered from either the customer security requirements or each VM reputation score calculated by the cloud provider.

In the same domain, some works consider the risk assessments and management to be implemented in the allocation process of the VMs. The risk assessments involve quantifying the risk on various layers of the cloud systems, such as the risk of VMs, PMs or hypervisors and allocating the VMs accordingly. For instance, the work in [41] proposed an isolated zone to separate the low-risk VMs from the high-risk ones based on four security dimensions that mainly affect the cloud system. These are VMs risk, hypervisor risk, co-residency risk and the network risk of the PMs. They also proposed placement strategies to reduce the risk of VMs placement. The first strategy is to place the VMs on the same network as the hosted PMs to reduce the risk of network connectivity. The second is to place the high-risk VMs in isolated zones. The third is to place the low-risk VMs that does not share the same network connections with the VMs in the isolated zone in a low-risk PM. Lastly, place each VM in the same PM to share the same network connectivity. They assume that the resources needed for the migration and placement process will always be available for the CSPs.

Moreover, in [53], they proposed a VM allocation strategy based on the risk management approach by quantifying and predicting the risk of PMs' future failure risk. Their work aimed to identify and predict the PMs' reliability in hosting the upcoming VMs. Therefore, mitigating the risks of allocating the VMs on hosts will potentially suffer from a service failure. Their strategy is based on a culture algorithm that optimises the VMs allocation by increasing the selected PMs' relatability. The method quality depends on collecting and monitoring the PMs' former and current resources status and activities.

3.3 Time-Triggered Allocation

The third domain focuses on allocating the VMs for a defined time to reduce the amount of sensitive data leakage through the SCAs. The SCA 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, this domain focuses on reducing the amount of data leakage due to the SCAs by considering the time factor for VMs residing on specific PMs.

The work in [83] proposed an information leakage model between co-resident VMs by defining three concepts. The first concept defines the relation between the VMs with each other, as it defines the VMs as either friendly VMs or unknown VMs. The concept of friendly VMs means that these VMs, defined as friendly VMs, shared the same PM in the past and did not trigger the intrusion detection system. The second concept defines the relation between the VMs and the hosted PMs as trusted hosts. In this case, the PMs who are considered friendly are the PMs who hosted the requested VM without triggering intrusion detection. The third concept is defining the co-residency time threshold between VMs; therefore, if a VM is placed with another VM for a time equal to or greater than the time set as a threshold, without triggering the intrusion detection system, these VMs will be considered friendly. The same rule is applied to the relation between VMs and PMs. They assume that the intrusion detection system is implemented on the VMs, to detect malicious behaviour.

Moreover, [72] introduced a placement algorithm to eliminate the information leakage without making changes to the hardware, hypervisor or even the OS of the VM. Time is an essential factor in their method, which means that the total time of two VMs co-resident directly relates to the amount of the leaking of private information. Their method depends on placing the VMs to selected PMs based on three-level of priorities. The first one considering the possibility of having replicated VMs for the target user, which means either the target user has more than one VM on the same PM or has replicated VMs dedicated to specific tasks, such as load balancing the database VMs. Moreover, considering collaborating malicious VMs, the malicious user has more than one VM or replicated VMs. In this case, information leakage will be considerably higher than the leakage when only one VM for each VM type, target or malicious, co-reside with each other. Secondly, considering the possibility of having replicated VMs for the target user. However, there are no replicated malicious VMs, or collaborate malicious users. In this case, the leakage is three-time higher compared to the normal situation when there is only one VM for each type. Thirdly, the worst-case scenario is replicating and collaborating malicious VMs allocated on the same PM with the target VMs. In this case, the leakage is six times higher than the typical situation when there is only one VM for each type. The proposed allocation process will consider these factors and allocate the VMs accordingly to reduce leaked information.

3.4 Co-residency Mitigation

The fourth domain focuses on reducing VMs co-residency with malicious VMs by focusing on the different allocation algorithm behaviour. Alternatively stated, focusing on developing an algorithm that manipulates the scheduling component of the cloud system. For example, in [15], they introduced an algorithm that deliberately delayed the start-up time for VMs to reduce the chance of co-residency with malicious VMs. Their algorithm assigns the VMs request to a queue before the placement to delay the assignment of the VMs. The delay aims to make the prediction of the arrival of the target VMs difficult for the malicious VMs to deduce. However, some attackers may have the ability to compromise the scheduling algorithm of CSP, and they can predict the location of the target VMs accordingly. Afterwards, and from the defined queue, their algorithm will choose a VM randomly from the queue and place it into a suitable PM. This random behaviour will increase the cost of the malicious user trying to predict the location of the target VM.

Furthermore, [96] propped a model for encouraging cloud users to migrate their VMs to a different PM to avoid co-residency threats from malicious VMs. Their idea is to migrate 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. However, migrating the VMs is not preferable to the users since it causes a downtime during the migration, even for the live migration (LM), leading to a service interruption.

On other direction, in [43], they studied the efficiency and coverage of the malicious VMs when they attempt to co-locate with the target VMs by examining four virtual machine allocation policies: the least VM, the most VM, the random VM, and the round-robin policies. According to their paper, efficiency is the percentage of the malicious VMs succeeding in co-locating with the target VMs at a specific time divided by the total number of malicious VMs. In comparison, the coverage is the percentage of the target VMs co-located with at least one of the malicious VMs at a specific time divided by the total number of the target VMs. They found out that the least VM allocation policy is the most secure policy as it will be hard to achieve co-residency, mainly when the malicious and the target users launch their VMs in a short period. The round-robin policy is the least secure one as it is easy for the malicious user to predict the following location of the target VMs. In connection to their previous work, they investigated the same VMs allocation policies and found out that the over subscription and VMs configuration can significantly affect the efficiency rates and coverage rates. They stated that, in this case, the most VM allocation policy is the most secure one.

Further, based on these findings, they present a VM allocation algorithm called previously selected servers first (PSSF). Their algorithm allocates the VMs belonging to the same user on a selected PM if the number of VMs belonging to this user exceeds three VMs. 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 [44], [46]. Continuing their previous work, they aim to identify the difference between the legal user and the malicious user and classify their behaviours to maximize the cost for a malicious to achieve co-residency.

Moreover, they identified the best behaviour for a malicious user to achieve co-residency with the target VMs. The malicious requires to use of multiple accounts, and each of these accounts creates one VM only. In this case, the PSSF policy will be less effective in preventing the malicious co-residency. As such, they classified the accounts into three categories, low, medium and high risk accounts. Whenever a new account is created, it will be classified as a medium risk until it is classified as either high or low risk. The malicious users will try to change their behaviour from medium to low risk to achieve co-residency. The main idea is to make it costly for them to change their classification from medium to low risk [45].

Additionally, in [75], they proposed an approach to reduce the co-residency with malicious VMs in CCEs. Their method aims to prevent the possibility of SCAs or cover channel attacks conducted by malicious VMs. Their work evaluated the allocation of VMs in two main aspects: the probability of co-residency and its coverage. Also, developed an algorithm called CRRD that focuses on reducing the chance of VMs co-residency by considering the number of VMs, users and PMs during the allocation. When new VMs initiated, the scheduler will check if the number of VMs have not exceeded a certain threshold. If the threshold has not been exceeded, the scheduler will spread the VMs for the same user across the available PMs to achieve load balancing allocation. However, if the threshold has exceeded, the scheduler will allocate the VMs for this user in previously selected PMs. This process concern more about the VMs side; however, from the PMs side, the algorithm filters the available PMs based on their availability of computing resources to accommodate the requested VMs workload.

Moreover, in [2], they introduced an algorithm that aims to reduce the malicious co-residency by allocating VMs with other VMs that previously shared the exact PMs. Their algorithm, called "previously co-located users first", has two main objectives: reducing the malicious co-residency and maximizing the used computing resources. The proposed solution assumes that the cloud provider does not know the type of users, whether malicious or regular. Also, it has no information about the number of VMs arriving in future time. Besides, it assumes that the malicious user has the control and the ability to obtain leaked information through side channels, and it can collaborate with another malicious user to gain more benefits from the leakage channels. The algorithm starts by checking if the user launches VMs in the past and if not, the scheduler will assign the VM to the eligible user PMs list. This list satisfies the condition of the availability of resources. On the other hand, if the user has previously launched the VMs, the scheduler will attempt to allocate the VMs with other previously co-located VMs.

Furthermore, [89], proposed a solution that aims to secure VMs placement and secure load balancing in the cloud environment. They assume that the CSPs control the scheduler and implement an intrusion detection system on the PMs. The CSP can also audit the user's activities and trace back their actions to identify the root cause of any incidents. The malicious user, on the other hand, could perform actions regularly as any other cloud user. Also, the malicious users are considered sophisticated in detecting the target VMs when they reside on the same PM. The malicious users will try to co-locate with the target VMs by launching many VMs or migrating them constantly. Their work enhances the Best-Fit algorithm, which focuses on enhancing the power consumption during the placement. The presented algorithm depends on having a high score of familiarity with the PMs and VMs during selecting PMs and placement of VMs.

Lastly, [11] presented a secure VMs placement that randomly selects and allocates VMs into specific PMs to reduce the chance of co-residency. Their method initially depends on having a small number of PMs open for an allocation; then, the allocation algorithm will randomly select a PM from the available PMs. Afterwards, and if the current PMs can not accommodate any new VMs, their algorithm will utilise new PMs and make them available for an allocation. Thus, their idea is to limit the available PMs and make finding a secure allocation under limited available spaces challenging. Indeed, the more the algorithm has many available PMs, the more secure allocation will produce.

3.5 Optimisation-Based Allocation

The fifth domain focuses on studying the behaviour of allocation policies by examining the ideal situations of an allocation under specific constraints. This domain follows an optimisation-based approach to secure VMs allocation while utilising existing solutions related to ideal situations. For instance, the work in [26] proposed a solution to enhance the security and performance of the VMs allocation process using the firefly algorithm. It is an optimisation-driven solution aiming to find the optimal allocation that co-locates the malicious VMs away from the regular VMs on the same PM.

Furthermore,[42] presented a secure VM allocation algorithm based on multi-objective optimisation. This solution improves the First-Fit algorithm, which provides an allocation that satisfies the resource constraint. The main objective is to produce many possible placements, and the algorithm chooses among them according to a set of defined rules. Accordingly, the highly ranked placement will be chosen and placed in a list as candidate allocation.

Moreover, in [25], their work depends on utilising an optimisation-based problem called Dolphin Partner for obtaining a secure VMs allocation. Their proposal is divided into two steps: prioritising the VMs with the most efficient energy-aware utilisation and the VMs with the most efficient memory aware utilisation. Then, from this extracted VMs list, their algorithm will analyse and select the allocations that yield fewer malicious VMs in them. Thus, their method aims to combine the efficient usage of resources while producing a secure VMs allocation. Moreover, they focus on the parameters that potentially cause a failure for the VM, for example, the memory utilisation of the VM. If the current utilisation of the VM is high, it is more prone to failure and considered a less secure VM than the VM with efficient memory utilisation. Alternatively stated, the malicious VM will have a better chance of attacking this VM, for example, by DoS, than a VM with efficient memory resources.

3.6 Secure Hardware Components

The last domain focuses on the secure VMs allocation on the hardware level of the cloud system. This domain requires a modification in the hardware level, and this modification includes either changing the mechanism of an existing component or adding a new one. For example, logically partition the cache into separate threads to minimise the effect of SCA. Alternatively, in some cases, using an existing component, which utilises on a different domain, to secure the processing on shared resources. For example, [81] proposed a solution that requires a change in the hardware to eliminate SCAs. 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. In the same manner, the work in [55] offers a mechanism for partitioning memory caches to defend against the software level of SCAs. They suggest that their scheme does not require a substantial modification to the current configuration of modern operating systems.

Additionally, [88] introduced a secure data processing component using Field-Programmable Gate Array (FPGA). FPGAs is reprogrammable to specific functionality, and they been used as an accelerator for the hardware processing, for instance, on Amazon cloud. Similar to the previous work, in [37], presented a secure FPGAs in order to minimise the reliance on CSPs for protecting the processing of the VMs in CCEs.

In different direction, in [95], introduced a system called CloudRadar to detect the cache-based SCA in CCEs. Their method operates by combining two main events to protect the VMs processing against malicious attacks through SCA. These events are: identifying when the VM is executing encrypted operations by utilising signature-based detection and monitor the abnormal activities that may occur on shared caches by anomaly-based detection. Furthermore, a VM migration may be triggered to protect the target VMs from a potential SCA from these two events. Furthermore, the proposed system can be implemented as a part of the monitor system of the CSPs to monitor the performance behaviour of the VMs and, therefore, model their activities for abnormal detection.

3.7 The Domain of The Thesis

The thesis aims to develop a secure VMs allocation algorithm in CCEs to defend against SCAs by mitigating the chance of malicious co-residency. The development of the algorithm starts by studying the aspects that affect VMs allocation security of the VMs allocation. Alternatively stated, develop an algorithm that minimises the malicious co-residency on shared resources and, therefore, minimises SCA's impact. As shown in the previous section, the earlier studies tackled the malicious co-residency by proposing specific solutions, for example, proposing VMs allocation algorithms or modifying the architecture of the CCEs to mitigate the threats of SCAs under specific constraints and assumptions. As such, their solutions will suffer from malicious co-residency occurrence when applied to other scopes or situations. Therefore, in this thesis, we will propose a solution from a holistic approach by focusing on studying the allocation behaviours that lead to the secure VMs allocation instead of only focusing on the allocation algorithm itself.

As illustrated in Table 3.1, we investigated six domains that directly related to obtaining a secure VMs allocation in CCEs against SCAs.

The first domain, VMs clustering, focuses on clustering, or grouping, the VMs into specific groups based on defined requirements, such as network dependability. In other words, the VMs that share the same network reachability constraints, or having connections between them, then these VMs are preferable to be allocated on the same PM, or at least on the same cluster of PMs. Another example is grouping the VMs based on the constraint of groups due to geographic constraints, or defined security configurations, or VMs behavioural analysis.

The second domain is security profiling and compliance, which is similar to the first one; however, it is not necessarily that the VMs grouped during the allocation process, as it only requires that the VMs follow a certain standard to be allocated on specific PMs. Thus, it is a grouping but with more flexibility than the first domain. For example, a CSP can specify that a group of PMs only host VMs specialising in medical health care and followed the (HIPA) standard [28].

The third domain is time-triggered allocation, which depends on the VMs allocation based on the time these VMs spend sharing the exact PMs. Otherwise stated, it focuses on defining the friendly VMs and the malicious ones based on the sharing time of co-residency. For example, if two VMs co-reside for a time more than a specific threshold without triggering any alarms or conduct malicious actions, it means they are friendly VMs. Otherwise, these VMs considered malicious, and the allocation will try to reallocate the VMs to obtain secure allocation based on these factors.

The fourth one is co-residency mitigation, which depends on developing a secure VMs allocation algorithm based on VMs and PMs configuration and specific factors. For instance, depending on the number of existing VMs and arriving ones, or the configuration of PMs and if they have many resources or limited resources. Also, the arrival of the VMs at a particular time or by changing the VMs allocation behaviour. It is a heuristic approach that aims to secure the VMs allocation based on the current situations rather than known data.

The fifth domain is optimisation-based, which focuses on obtaining a secure VM allocation under ideal situations of the optimised algorithm. Such approaches utilise existing optimal solutions applied to different domains to produce a secure allocation under different constraints. The last domain is secure hardware, which depends on adding a component or changing the hardware level of CCEs infrastructure to secure the processing of VMs when they are share the same PM. For example, secure the operations, or threads, processed on shared CPUs of a shared PM, to minimise the leaked information through side-channel, and therefore, minimising SCAs.

Table 3.1: Domains of the related work

Domain Area	**References**
VM Clustering	[61], [93], [16], [92], [5], [60], [1], [64]
Security Profiling and Compliance	[12], [3], [19], [73], [41], [53]
Time-Triggered Allocation	[83], [72]
Co-residency mitigation	[15], [96], [43], [44], [46], [45], [75], [2], [89], [11]
Optimisation-Based Allocation	[26], [42], [25]
Secure Hardware	[81], [88], [95], [37], [55]

The following section will explain in detail the aspects considered to develop the secure algorithms in this thesis.

3.7.1 Aspects affecting the secure allocation

This section aims to introduce the aspects that affect developing a secure VMs allocation in CCEs, from the perspective of the thesis. We aim to present an abstract perspective about the secure VMs allocation process and the factors that potentially affect the overall security of a proposed allocation. In order to do so, we perform a deep investigation of the previous solutions in the same area, in Table 3.1. The outcome of this investigation is to find an abstract solution based on the previous literature by learning about four main aspects that affect the security of the VMs and expose them to SCAs.

The first one is the level of cloud system, either the VMs or PMs, the hypervisor and the VIM. The VIM is usually the component that responsible of the allocation algorithm and has visibility over a cluster, or clusters, of the cloud infrastructure. The visibility includes the hosted VMs, the available

PMs, available networks, amount of available storage and the communication between these components. For example, in Table 3.1, [1, 3, 12, 16, 19, 41, 44–46, 61, 72, 83, 92, 93] focus on the aspects related to the VMs layer of the cloud systems, while the works in [45, 81] focus on the aspects related to the PMs layer, and [41, 60, 83] focused on the hypervisor layer.

The second aspect is the attacker behaviour, which includes the actions that the malicious VMs perform to co-reside with the target VMs. These actions are usually classified as normal actions. For instance, requesting an increase of the available resources of malicious VMs [43, 45, 83]. If the resources are limited, it could lead to a migration of the malicious VMs to another PM, therefore, a better chance for malicious co-residency.

The third one is to study the expected impact of this attack depends on the assumed scenarios. These attacks could be classified as passive or active attacks depending on the nature of the attack and how it interrupts or compromises the target VMs. Finally, the fourth aspect focuses on the previous researchers proposed countermeasures to defend against this type of attack under each classification. We will discuss these aspects in more detail.

In Figures 3.1 and 3.2, we present a classification of the aspects that affect the VM-level, PM-level and Hypervisor-level malicious co-residency based on the conducted investigation of the literature review. The classification includes the attacker behaviour, attack impact and the proposed countermeasures.

3.7.2 VM Level Attack

This section will introduce the aspects that affect the VMs level malicious co-residency, which are the attacker behaviour, the attack impact and the countermeasures, as illustrated in Figure 3.1.

Attacker Behaviour

Firstly, the attacker behaviour includes the malicious user's actions to achieve malicious co-residency with target VMs. For instance, the attacker could utilise a vulnerability in the operating system (OS) of the VM. This attack happens due to flaws in the configuration of drivers on the OS level. Alternatively, normal operations performed by the OS leaking data that the malicious user can utilise to attempt SCA [1, 41, 61, 93]. Moreover, the malicious user could perform regular actions to force the VMs scheduler to initiate a VM migration, i.e., re-allocating the VMs. These actions, for example, demanding more resources which is not currently available on the hosted PM, or some attacker tries to deliberately peak their VMs performance to the maximum to force the migration to be triggered. This peak of performance, and if the CSP applies a particular SLA related to performance threshold, it will potentially lead to a migration of the VMs. In some cases, the malicious users will try to increase their VMs number in a short period, which may stimulate the scheduler to perform VMs migration [43].

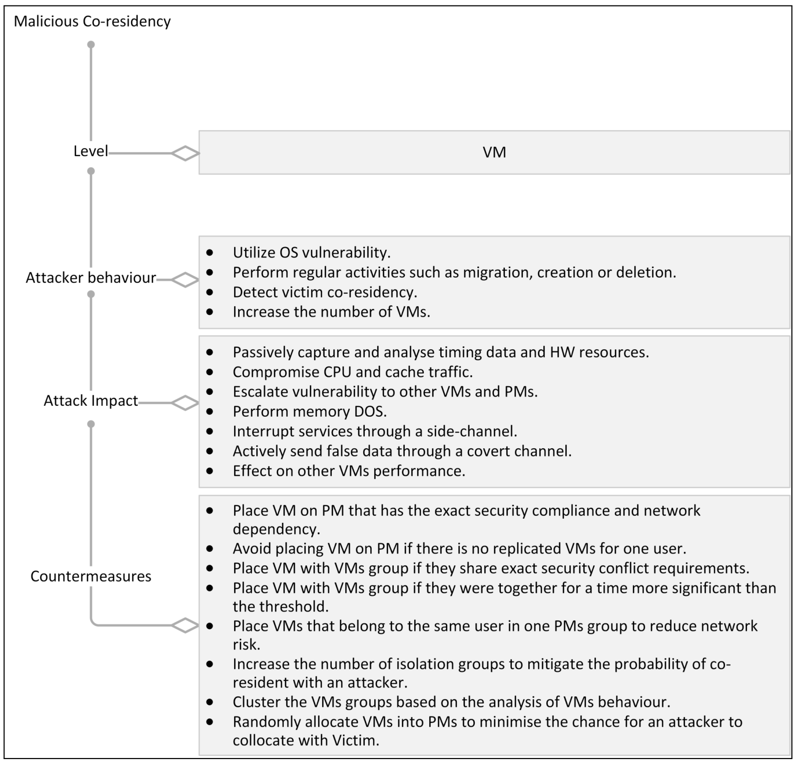

Malicious Co-residency
Level
VM
Attacker behaviour
• Utilize OS vulnerability.
• Perform regular activities such as migration, creation or deletion.
• Detect victim co-residency.
• Increase the number of VMs.
Attack Impact
• Passively capture and analyse timing data and HW resources.
• Compromise CPU and cache traffic.
• Escalate vulnerability to other VMs and PMs.
• Perform memory DOS.
• Interrupt services through a side-channel.
• Actively send false data through a covert channel.
• Effect on other VMs performance.
Countermeasures
• Place VM on PM that has the exact security compliance and network dependency.
• Avoid placing VM on PM if there is no replicated VMs for one user.
• Place VM with VMs group if they share exact security conflict requirements.
• Place VM with VMs group if they were together for a time more significant than the threshold.
• Place VMs that belong to the same user in one PMs group to reduce network risk.
• Increase the number of isolation groups to mitigate the probability of co-resident with an attacker.
• Cluster the VMs groups based on the analysis of VMs behaviour.
• Randomly allocate VMs into PMs to minimise the chance for an attacker to collocate with Victim.

Figure 3.1: Aspects of VM-level malicious co-residency

In addition to the attacker behaviour, the malicious user aims to detect the existence of co-resident VMs on the same shared PM to initiate the attack [95]. The detection of the target user is not necessarily that this target user is the one that the attacker aims to collect its data. It could be simply a user that performs a heavy operation on the same shared PM. A heavy operation indicates that the VM is currently performing a task requiring encryption of data or a vital task. These major operations could be captured and spotted through a normal side-channel on the cache level. To put it differently, a malicious user can detect if the VMs share the same PM, performing major operations by calculating the time needed to operate on its own VM and comparing the results of the captured times for any spike.

Attack Impact and Consequence

The attack impact of such malicious behaviour depends on whether the attack is from a passive or active source. For instance, and as stated earlier, the attacker can capture data of the operations on the shared PM through its VM. These data include the time needed for completing an operation or sensitive encrypted information. These captured data result from the sharing of computing resources, such as CPUs and caches, among cloud users. Moreover, these data could be information about power consumption, performance data, global kernel data, or asynchronous kernel events [54]. Moreover, the weakness in applying security standards for some VMs allows malicious VMs to compromise them and escalate the attack to other VMs through these none secure VMs.

After collecting the data, the malicious user will profile the target VMs activity, e.g. execution time, with the help of machine learning techniques [39]. Then, the malicious user can actively perform a DoS attack on shared memory by peaking the performance, assuming the CSP allows for memory peaking to maintain performance for the VMs [33]. To clarify, CSPs may allow the VMs to have more than the allocated resources at a peak time of VMs performance to accommodate the sudden rise in resource demand. The allocated extra resources are taken from shared VMs that are not utilising these resources at the time of peaking. This rule allows the CSP to maintain VM's performance and avoid any interruptions affecting the SLAs. However, the attacker could take advantage of such flexibility of resources to either affect shared VMs' performance or interrupt the service performed at peak time. For example, in [87], they perform a SCA by taking advantage of the shared memory and ballooning technology in the cloud. The Ballooning is a driver installed on each VM, used simply by the hypervisor to ensure that each VM has enough resources at a specific moment to perform a particular task that requires more memory, and this is achieved by taking resources from the co-resident VM's resources.

Moreover, data leakage can cascade to other VMs hosted on the shared compromised PM. The malicious user will have information about the resource's utilisation and profile the VMs connectivity accordingly. Furthermore, the attackers can exploit VM image vulnerabilities that propagate its flaws to other VMs and PMs in the environment. From a security perspective, any data breaches or flaws in the systems configurations consider threats to the CCEs and the organisation utilising cloud services, leading to significant financial losses.

Countermeasures

Thirdly, the countermeasures for such attacks mainly focus on the methods and rules of allocating the VMs securely while maintaining other constraints such as resources availability and limited PMs. For example, and to defend against SCA on VM-level, the group of VMs that shares the PMs resources should share the exact security requirements and standards. The requirements include network dependency and reachability for VMs and PMs and the same applied security standard on both VMs and hosted PMs. Moreover, the scheduler should allocate the VMs together for a specific time and did not trigger any malicious behaviours among them; this requires adding an intrusion detection system on both VMs and PMs levels. Many types of intrusion detection systems (IDS) can be applied to the cloud system to secure the VMs processing from malicious intrusion [71].

In some cases, allocating a VM into a PM that contains many VMs belonging to the same user could be less secure. If this user, with many VMs, is a malicious one, then the amount of data that this user can collect will substantially be higher due to replicating malicious processes this user can perform using its many VMs. For the VM scheduler that follows a grouping behaviour, it is preferable to create many VMs groups and divide them according to their defined security standard or based on VMs processing behaviour. The more VMs group existed, the more reduction in the chance of the malicious VMs co-residency to occur.

Lastly, if applicable, following a random VMs allocation behaviour is potentially reducing the effect of malicious co-residency occurrence. The malicious users that depend on detecting the target VMs will be much harder for them to predict the VM scheduler's allocation behaviour and, therefore, achieve malicious co-residency. However, the CSPs allocation policy is limited to other constraints such as resource limitation and SLAs commitments; as such, achieving the randomness in allocation should consider these restrictions.

3.7.3 PM and Hypervisor Level Attack

In PM and hypervisor levels, in Figure 3.2, the attacker behaviour includes the malicious user's actions performed on these two components to achieve malicious co-residency with target VMs. For example, the malicious VMs can observe, capture and analyse the data processed on a shared cache of the shared PM. As mentioned, SCAs depend on capturing data throughout an intermittent time; then, these data can be utilised for profiling and targeting specific processes or VMs. The profiling could perform by utilising a machine learning-based technique to cluster and classify the data [66]. A malicious VM can compromise the hypervisor by collecting data about the scheduling table on PMs. These data contain information about the co-located VMs, including their migration route. It allows the malicious user to predict the behaviour of the scheduling algorithm and, therefore, plan the attack accordingly [83].

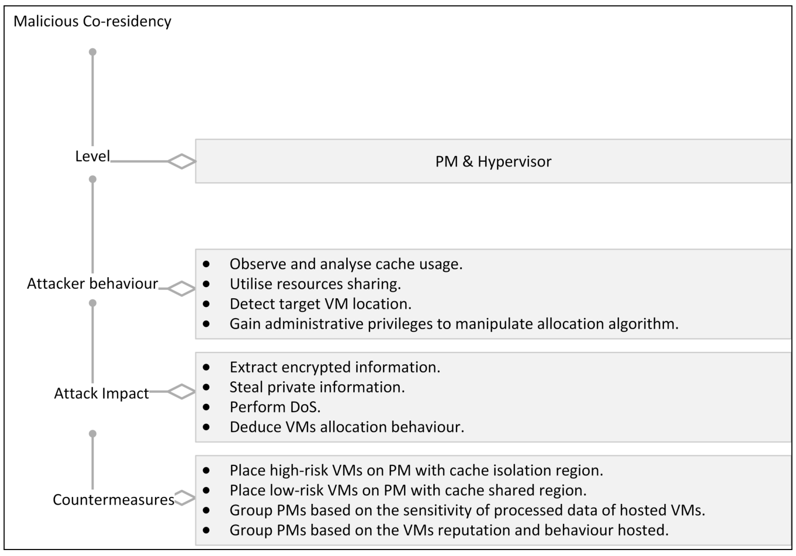

Malicious Co-residency
Level
PM & Hypervisor
Attacker behaviour
• Observe and analyse cache usage.
• Utilise resources sharing.
• Detect target VM location.
• Gain administrative privileges to manipulate allocation algorithm.
Attack Impact
• Extract encrypted information.
• Steal private information.
• Perform DoS.
• Deduce VMs allocation behaviour.
Countermeasures
• Place high-risk VMs on PM with cache isolation region.
• Place low-risk VMs on PM with cache shared region.
• Group PMs based on the sensitivity of processed data of hosted VMs.
• Group PMs based on the VMs reputation and behaviour hosted.

Figure 3.2: Aspects of PM-level malicious co-residency

Similar to VM-level classification, the attack impact can range from passive attacks to active ones. The type of the attacks is similar to the attacks of VM-level classification, but it differs in the mean utilised to perform the attack. For example, extracting the data does not necessarily because there are flaws in the driver configurations of the VMs; it could be because the hosted PM has flaws in hardware configuration. In many cases, it is not a flaw in the hardware configuration, but simply it could be a standard channel utilised by a malicious user, for instance, the shared level of the cache or Ballooning of the shared memory. These are standard configurations used for common purposes; however, these channels leaking important information lead to SCAs.

Lastly, the countermeasures for these levels of attacks include a minor modification on the hardware for some cases, while others require to cluster the PMs according to specific standards. For instance, the available PMs could be clustered and classified according to the severity of the VMs hosted on them. The PMs that host highly sensitive VMs can be equipped with a hardware component to secure its data processing, i.e., make the data encrypted while processing on CPU and cache. The FPGA is an example of using additional components to secure the data processing on the PM level [88, 94]. Other methods focused on modifying the existing hardware components, and these modifications could only be logical, not physical. In other words, modify the cache's shared level structure and divide the cache into multiple caches to create an isolation zone. These isolation zones will separate the processing threads from each other and reduce the amount of leaked information from that channel. After this isolation, the VMs with highly secure information could be placed into PMs with this feature of cache separation to ensure the data a processed securely.

Summary

In this chapter, we present a review of the previous researches that tackling the SCAs and malicious VMs co-residency in CCEs. The review divides the previous researches into six domains which are the VMs clustering scheduling algorithms, the VMs profiling based algorithms, the time triggering based VMs allocation algorithm, the algorithms that aim to reduce the chance of co-residency, the optimisation-based algorithms, and the algorithms that based on hardware level remedy. Moreover, we presented classification of the aspects affecting the secure allocation in CCEs by studying the effect of the attacker's behaviour and the attacker's impact. We also introduced the countermeasures analysis of the SCA based on the type of attack and the level of virtualisation. The levels of virtualisation compromised by SCA include the VMs, PMs, and the hypervisor. This analysis contributed to developing the secure VM allocation model, which will be discussed in detail in the next chapter.

Chapter 4

Developing and Evaluating Secure VMs Allocation Model

Preface

In this work, we aim to develop a secure VMs allocation in CCEs to defend against SCAs. Thus, we defined the aspects that affect the development of the intended secure VMs allocation approach, as explained in Section 3.7.1. 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.

This chapter aims to define the cloud data centre model assumed in this thesis, including its components that affect the development of the cloud system. The components include the PMs and the VMs structure and how they interact in CCEs. Moreover, the virtualisation layer, which is responsible for sharing the resources in the CCEs. In addition, we will define the method of resources allocation and sharing of CCEs computing resources, which allows the cloud user to utilise resources based on their needs.

Further, 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. Additionally, we introduce a definition of cloud users based on their behaviour analysis, thus, classifying them into specific types. 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.

Finally, we present an evaluation of the secure VMs allocation model using an Linear Programming (LP) solver, which aims to study and examine the behaviour of the secure optimal allocations under the defined model constraints. The evaluation includes different scenarios and structures of the CCEs components, which captures the secure optimal VMs allocation patterns.

4.1 Cloud Data Centre Architecture

As stated, the objective of the thesis is to obtain a secure VMs allocation in CCEs to defend against SCAs. Thus, we proposed the cloud data centre model, which considers the public cloud IaaS model, where the users provided infrastructure computing resources as a service, as depicted in Figure 4.1.

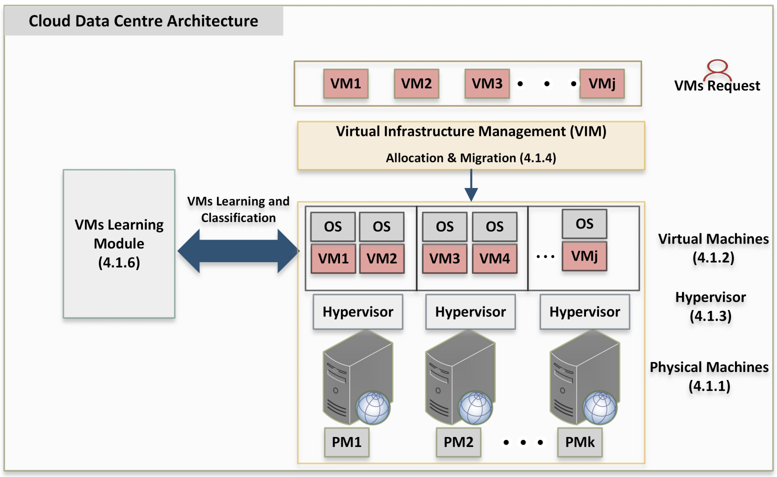

Cloud Data Centre Architecture
VM1
VM2
VM3
VMj
VMs Request
Virtual Infrastructure Management (VIM)
Allocation & Migration (4.1.4)
VMs Learning Module (4.1.6)
VMs Learning and Classification
OS
VM1
VM2
VM3
VM4
VMj
Virtual Machines (4.1.2)
Hypervisor
Hypervisor (4.1.3)
Physical Machines (4.1.1)
PM1
PM2
PMk

Figure 4.1: Assumed Cloud Data Centre Architecture

This service includes computing resources, through virtualization, such as storage, network and servers. Thus, the users have a level of access to the physical resources of the IaaS service model. For example, each virtual computing resource, e.g., CPU, is linked through a driver on the operating system (OS) level to the shared physical CPU [23]. The proposed model consists of different entities that affect obtaining the secure VMs allocation: the VMs, PMs, learning module, virtualisation layer, VIM and virtual machine allocation and migration algorithm. The components will be described briefly in the following sections with their relation to the proposed objective.

4.1.1 Physical Machines

We will start from the lower level of the cloud system, as shown in Figure 4.1, the hardware or physical machines (PMs). Without the virtualization technology, the PMs are bare metal devices dedicated to a single OS with predefined fixed resources. Otherwise stated, the PMs are computers designed to be used by single users for single-purpose with fixed resources and without resource sharing or efficient usage. Each PM is equipped with computing resources such as physical RAMs, physical CPUs and other physical components. We refer to them as physical resources because when defined on the VMs level, they will be virtual resources. Each physical resource of a PM will be shared and allocated, based on demand, for groups of VMs hosted or allocated on this PM. Our model assumes that the PMs are connected and that all PMs have visibility over the cloud system. Hence, all the VMs hosted on the cloud system can be shared by any available PMs and move from one PM to another accordingly.

4.1.2 Virtual Machines

The VMs is a computer that can run on a shared environment, i.e., sharing the computing resources, such as CPUs and RAMs, with other VMs. It is a software or a virtual system that can mimic a physical computer and run as a guest on hosted PM. Each VM, hosted on a PM, is allocated to a specific amount of the physical resources through virtualisation technology, which will be explained in the following section. As illustrated in Figure 4.1, each VM has a unique OS and specific purpose application from the other VMs who shared the same PM. Moreover, the VMs are equipped with virtual computing resources such as virtual CPUs, virtual RAMs and virtual storage, connected to the physical resources through the virtualisation layer. In essence, the main objective of VMs is to allocate and utilise computing resources efficiently and on-demand to avoid resources wastage.

4.1.3 Virtualisation

The concept of virtualisation is expressive to its role in the cloud computing system, which is to virtualise the physical resources to be shared and utilised by the virtual guests, or VMs. Alternatively stated, virtualisation technology allows the provisioning and sharing of VMs, with their unique OS, on a single PM [91].

Accordingly, the virtualisation technology is feasible by a software component, or layer, called the hypervisor, which aims to integrate the VMs resources with hosted PMs resources. As illustrated in Figure 4.1, the hypervisor is responsible for allocating a PM's physical resources to multiple VMs virtual resources through the virtualisation layer. For instance, the allocated and shared computing resources are the CPUs and RAMs. Moreover, it is responsible for the separation of the VMs logically while running on shred PMs. However, this separation does not avoid SCAs occurrences, as we will explain later in the threat model.

Implementing the hypervisor depends on the CSPs; however, the most common method is implementing the hypervisor software on the PM as a bare metal server. Then, after implementing the software, or hypervisor, it will allow separating the physical resources of the PMs into multiple divided resources, which can be utilised and shared by VMs. There are many commercial and open-source CSPs of the hypervisor that uses this implementation method, for example, VMware ESXI [29] and Microsoft Hyper-V [51]. The second less common method of implementing the hypervisor is implementing it on the OS directly instead of the bare metal server. In other words, the hypervisor software installed as an application, and then it can allocate computing resources from the hosted OS to the VMs created by this hypervisor, for example, VMware Workstation [90]. It is less common because it has poor performance than the first type, as it depends on the hosted OS, not on the bare metal computing resources, i.e., direct connection to the PM [32]. In the thesis, we only consider the first type of hypervisor.

4.1.4 VMs Allocation and Migration

This section will describe VMs allocation and VMs Migration in a virtualised environment, thus in CCEs. Generally speaking, and as illustrated in Figure 4.2, the VIM is responsible for allocating and managing the physical computing resources to the guest VMs according to their requirements. As explained in Section 4.1.3, the hypervisor aims to virtualise the computing resources such as CPUs, RAMs and Networks and make them shared for guest VMs, hence becoming virtual CPU (vCPU), or virtual RAM (vRAM). While the VIM is responsible for allocating the resources for VMs includes creating them, deleting them, and migrating them to another PM, which is called VMs migration [50].

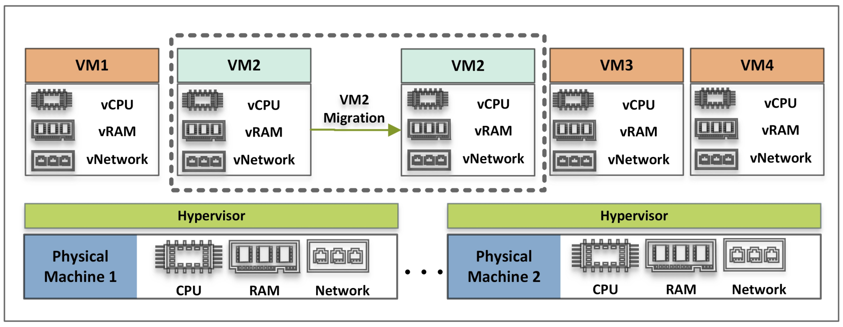


Figure 4.2: VM Allocation and Migration

In the following, we will introduce more details about VMs allocation and migration in a virtualised environment.

VMs Allocation

VMs allocation refers to the steps to create or allocated a VM into a selected PM, i.e., allocating the hardware computing resources, such as CPU or RAM, to the requested VM. For example, in Figure 4.2, VM1 and VM2 are allocated to PM1, while VM3 and VM4 allocated to PM2. The allocation to a specific PM depends entirely on the method of VMs allocation implemented. Otherwise stated, there are two steps of allocating the VMs, selecting the PM suitable for the VM and allocating the resources of the selected PM to the guest VM, vCPU or vRAM.

The first step, which is selecting the PM, depends on implementing the VMs allocation policy in CCEs. The methods are different based on the objective of allocating the VMs, for example, allocating the VMs for power consumption optimisation, resources load balancing, network overhead optimisation or security objectives [65]. In this thesis, we aim to allocate the VMs securely into suitable PMs to reduce the malicious co-residency and, therefore, SCAs, thus, security objective.

The second step is allocating the resources into selected PM, i.e., allocating the demanded VMs resources on shared PM. Each component of the computing resources has its method of sharing and allocating the selected PMs resources, either Time-sharing or Space-sharing. Time-sharing means that the resources are allocated for a time limit, then the same resources will be allocated to another process, or guest VM, based on the method of priority for selecting these processes. On the other hand, Space-sharing allocates the space of computing resources entirely to the guest VMs at all times. The allocated space includes allocating the pages of the RAM or CPU cores [18].

VMs Migration

VMs migration refers to the steps needed to move the current state of a VM from source PM to destination PM. The current state of a VM includes the data on the hosted CPU and hosted memory files that are processing while moving, or migrating, the VM. As shown in Figure 4.2, the VM2 is moved from PM1 to PM2, called VM migration.

Several reasons might trigger VMs migration in CCEs, for instance, a sudden failure to the hosted PM, which leads the guest VMs to be migrated to another PM. Alternatively, for optimisation reasons, preserving the load balance between the available PMs for power consumption efficiency. In our work, we trigger the VMs migration for a security-aware objective to minimise the chance of malicious co-residency during VMs allocation.

There are two types of VMs migration, live and non-live VMs migration. The live VMs migration, which we considered in this work, is moving the current state of the VMs from one PM to another. We will refer to live migration in the thesis as VMs migration. On the other hand, non-live migration starts moving the VM to another PM after stopping the execution of the VM, i.e., turn the VM off. These types of VMs migrations depend on the situations needed and the design of the data centre of CCEs [4].

In some cases, the non-live migration is forced because there is no network connectivity configured that serve the purpose of live VM migration between the source and destination. Alternatively, the hardware configuration of the two PMs is different, which makes it not possible for a VM to migrate its current state, on a CPU, for example, from one PM to a PM with a different configuration of the CPU. However, some cases select this type of VMs migration for SLAs reasons. For example, a VM user wants to ensure that the current state of memory page files is not corrupted or missed during the live VM migration.

As stated in this work, we consider live migration as the migration type implemented in our work. The steps of VM migration start, after triggering the VMs migration, by the initialisation phase, which ensures that the selected PM has enough resources and meets the requirement for VMs demands. This initialisation step mainly depends on the PM's selection process, which depends on the behaviour of the VMs allocation algorithm. The next chapter of this thesis will describe the VMs allocation algorithm and VMs migration algorithm steps.

After selecting the PM, the next step is allocating the resources for the migrated VM on the destination PM to ensure that other VMs does not utilise the allocated resources. The following step is to suspend the VM, preserve its current state to an image file, and copy it to the destination PM, including the data processing on the vCPU registry file and the vRAM pages. The reason to call the image file in this term is that it captures the current state of the migrated VM, exactly like capturing pictures from a camera device.

After the image file is captured and copied to the destination PM, the VM on the source PM will continue to run normally, then the first iteration of moving the VM from source to destination PM will start. The iterations mean that, and since the current state is captured, a log file will be created on the source and destination PM to capture the changes happening on the VM. The log file will be used to update the changes that happened on sources PM to the destination PM. This step will iterate several times until all the logs are updated, and then stop the VM on source PM and copy its remaining logs to the destination PM. Afterwards, after the VM is migrated, it will be committed and synchronised to its previous state before migration [63].

4.1.5 VMs Allocation and Migration Integration

In the previous Sections 4.1.4 and 4.1.4, we described VMs allocation and migration as individual components. In this section, we will describe the integration between them as intended in the thesis. As we mentioned, the objective of this work is to introduce VMs allocation, and therefore VMs migration, to minimize the chance of malicious co-residency on the shared PMs. Thus, we aim to allocate the VMs securely and trigger VMs migration to preserve or enhance the state of the secure allocation. Otherwise stated, and as shown in Figure 4.3, we assume that the VMs migration only triggers upon the arrival of new VMs, forcing the VMs migration to happen. For example, VM9 require an allocation without sharing other VMs; therefore, a VM migration triggers and VM4 and VM5 are migrated to PM2 and PM4, respectively, to comply with the rule of VMs allocation algorithm. This example is only to explain the assumption we made for VMs migration process in the thesis.

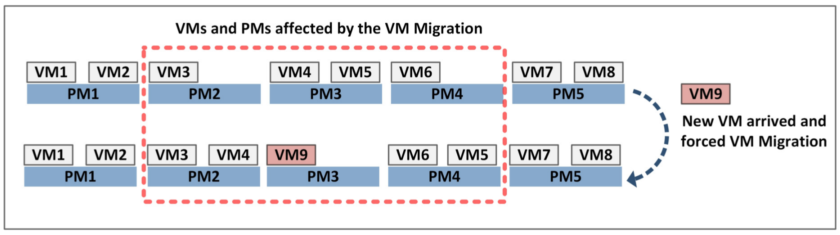


Figure 4.3: The Integration between VM Allocation and Migration Assumed in the Thesis

In this work, the VMs migration will be trigger if the VMs arrival and allocation will lead to producing an allocation with malicious co-residency. For example, in Figure 4.3, we assume that the first allocation produced before migration is secure. Furthermore, the new arrival VM, denoted as VM9, is malicious and will produce an allocation with malicious co-residency between existing VMs, thus, the VM migration will be triggered. Here we described the events that lead to VMs migration assume in this work, and it mainly depends on the objective that we aim to achieve from the VMs allocation, which is secure VMs allocation. However, in the next chapter, extensive details will be presented about selecting the PMs, how the VMs migrated, and which VMs are selected for migration.

4.1.6 VMs Learning Model

As illustrated in Figure 4.1, 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.

Therefore, we consider utilising the method presented in [76], where the authors introduce a machine learning-based technique for analysing the VMs behaviour by capturing specific factors and categorise the VMs accordingly. Their method is based on detecting the malicious behaviour of cloud users by analysing the abnormal activities of their VMs, or other computing components that they utilised. These activities are collected from different cloud system levels, including operating systems, VMs, PMs, or network devices levels. Any abnormal activities will trigger the monitoring system, which is an Intrusion Detection System (IDS) responsible for collecting the events. The type of malicious behaviours can be classified based on the level of cloud systems where the abnormal event occurs. For instance, on the VMs level, multiple VMs are created, deleted, migrated or cloning in short periods. On the network level, unusual ports configuration changes, creations of new ports, persistent host IP querying or deletion of ports group.

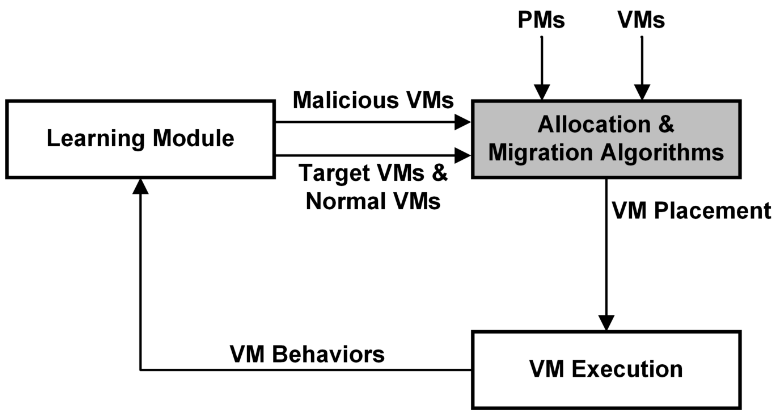


Figure 4.4: VMs Behaviour Learning Model

Moreover, in [45], the authors present a model for analysing the behaviour of VMs by monitoring specific factors that help categorise VMs into specific classes. These are: a user launching a small or large number of VMs at a particular time, or at a periodical time, or keeping at least one VM active at all times or 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 VMs.

As shown in Figure 4.4, we will assume that a CSP analyses the behaviours of VMs, to identify suspicious VMs and 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 have the required awareness of the security threats on the cloud systems. For example, some users may require their VMs to be allocated to specific geographic locations to control their data flows and comply with regulations. Also, some require their VMs to be allocated on PMs that are configured under the same network subnet to optimise network traffic and minimise the risk of network connectivity. The constraints gathering process is not always possible and depends mainly on users' awareness of security threats. However, it is part of many steps performed by the CSP to analyse and profile VMs behaviours.

Moreover, 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 VMs execution to capture their activities and possible suspicious behaviours. 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 high-risk. The CSP should handle these VMs from a security perspective and perform allocations according to the categorisation result while meeting hosting requirements.

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, with the following constraints:

$$V = T \cup M \cup N \tag{4.1}$$

$$T \cap M = \emptyset \land T \cap N = \emptyset \land M \cap N = \emptyset \tag{4.2}$$

The first constraint, in Eq. (4.1), indicates that each VM belonging to any of the defined types belongs to the set of all VMs, which is V. While in Eq. (4.2), it indicates that any VM categorised with a specific type can not be redefined with another type during the allocation process. Otherwise stated, each VM belonging to a particular type group is unique to this specific VM type group and will not be duplicated to other VMs type groups, during the allocation.

We define a target *VM*, a malicious *VM* and a normal *VM* as follow:

Definition 4.1.1 (Target VM (T))**.** 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 produced from the learning component, in Figure 4.4.

Definition 4.1.2 (Malicious VM (M))**.** A *VM* can be classified 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

Definition 4.1.3 (Normal VM (N))**.** A normal *VM* is a *VM* that is classified as neither a target nor a malicious *VM*. Therefore, it's a harmless normal cloud user.

4.2 Threat Description

In CCEs, as we described, resources allocation is flexible and enables multiple users to share common computing platform dynamically available resources. This sharing invariably means that a PM can potentially share its resources among a set of distinct users or VMs, known as VM co-location. Therefore, the security threats for these shared computing environments have invariably shifted, leading to significant threats to cloud users, unlike the traditional on-premises data centres where the VMs have dedicated resources, a more secure and control environment. The types of threats that arise when a malicious

VM shares with a target VM range from confidentiality breach to service interruption attacks [52].

Thus, and as it shown in Figure 4.5, VMs co-location, though enabling efficient resource sharing, is creating unwanted side channels, which can be sources of potential SCAs, such as cache-based SCAs. Informally, side channels are (unwanted) communication channels between processes that may leak sensitive outputs from a process [97]. As such, SCAs will have impact that can extend from applications level to the hardware level [14, 82] and will become more prevalent due to the range of side channels that exists. In Section 2.3, we presented background information about SCAs generally and in cloud computing systems.

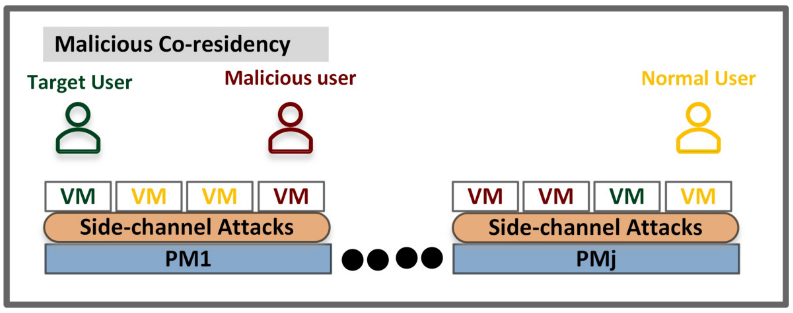


Figure 4.5: Threat Model

Accordingly, when VMs are co-resident (or co-located) on the same PM, one (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. Specifically, the attacks can occur through a cache-based channel by utilising the sharing capabilities of the cache levels [77]. 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 analyse them to formulate an attacks model using a machine learning-based approach. Afterwards, from the outcome of the analysis, the malicious VMs can conduct another form of attack, such as a DoS, or compromise the target VM based on the collected data. In the following sections, we will explain more about the steps of the attack.

4.2.1 Achieving Malicious Co-residency

From the malicious user perspective, the first step of conducting an SCA is to achieve a co-residency with the target VMs, leading to a malicious co-residency. The malicious co-residency means that the malicious VMs and the target VMs sharing the same PM. 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 attacker needs to understand how the CSPs allocates the VMs to formulate the attacks based on this knowledge. For instance, in [85], 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.

4.2.2 Capturing Execution Time Data: Cache-based Attack

After the malicious co-residency occurs between the malicious VMs and the target VMs, the SCA will be initiated by the malicious user. In our 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. Specifically, the cache-based SCA occurs on shared cache levels, level 2, denoted as L2 and level 3, denoted as L3, shared among hosted VMs, or threads. 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 consider a current heavily operation executing on the physical machine from a co-resided VM.

For example, we will briefly explain a well-known form of this attack called Prime-Probe attack [14]. It performs in two stages; the first stage is prime, which means that the malicious VM will perform a read-memory operation on all memory blocks that lead to placing all the malicious VM data in the cache. At a specific time, the target VM will perform a heavily encrypted operation, requiring utilising the full cache. In this case, the malicious VM data will be evicted from the cache and replaced with the target VM data. The malicious will repeat the read-memory operation and calculate the time if it takes longer; it indicates a sensitive operation executing at this time. If the execution time is significantly high at a precise moment, it will indicate that the VM performs a sensitive or essential operation, and the attacker can interrupt or extract this information.

Overall, the SCAs depend on collecting information from normal operations output, such as execution time on cache levels. Furthermore, the collected information has no major impacts when they are 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.

4.2.3 Formulate attacks model

As stated, the extracted information helps the malicious user profile the target VMs with the help of machine learning techniques. In other words, formulating an attacks model to systematically conducting different forms of attacks to interrupt and compromise the services of the target VMs. For example, in [39], they utilise a machine learning-based approach to conduct a cache-based attack by profiling the activities of cloud users. They were able to capture data features resulted from cache-based access that represents different types of applications. Their approach showed that the captured information could be collected regardless of the need to synchronising the cache access between the malicious and target VMs.

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. We assume that the malicious users are sophisticated attackers who can utilise the leakage channels through SCA by merely co-residing with the target VMs. In this thesis, we present two significant steps involved to overcome this problem. Firstly, identifying the malicious and target VMs, as we describe using the learning module, in Section 4.1.6. Secondly, develop an allocation algorithm that aims to allocate the malicious and target VMs apart from each other.

4.3 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) [34]. Generally speaking, BPP aims to pack objects into bins with a fixed number of available resources while using fewer bins. Altogether, this section introduces the system model formulation of the thesis by explaining the objective functions of secure VMs allocation. Moreover, it describes each different aspect of the system constraints implemented to achieve the optimal solution and address the problem of SCAs.

4.3.1 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. $A_u : V \to P$: ***VM allocation function to a PM.***

9. $\mathcal{A}$: ***Set of all possible allocations.***

10. $\mathcal{M}$: ***Set of transitions between allocations.***

11. $Move(A_u, A_{u+1})$: ***Set of VMs that are migrated during transition from (A_u) to (A_{u+1}).***

12. $CoRe(A_u)$: ***Set of PMs at which malicious co-residency occurs.***

4.3.2 System Model

The system consists of a set P of k physical machines, labelled $PM_1, \ldots$ which remains unchanged in the lifetime of the allocation process. Each $PM_j, 1 \leq j \leq k$ has the same set of configuration, i.e., same hardware configurations, but in varying quantities. For instance, one PM may have more storage, CPU cores or RAM allocated than another, i.e., we assume that the PMs resources are heterogeneous, thus, the system to be heterogeneous in terms of resource availability. The reason for the assumption about having the same hardware configuration is because it's a requirement for the live migration to work, as explained in Section 4.1.4. Moreover, we denote by $R(PM_j)$, the amount of physical resources available on PM_j during the VM allocation.

Moreover, the system consists of a set V of virtual machines, labelled $VM^1, \ldots$. We assume that the set of VMs is both heterogeneous and non-heterogeneous for our model. In other words, we will examine both situations when the requested set of VMs are either have the same set of resources or different ones. Thus, each $VM^i, 1 \leq i \leq n$ has either the same or different set of resource type requirements in our system model. Furthermore, we assume that all the resources needed by a VM can be met by the available PMs, and we denote by $N(VM^i)$, the amount of resources needed by VM^i. As explained in Section 4.1.6, the set V is partitioned into three sets: set T of target VMs, set M of malicious VMs and set N of normal VMs.

4.3.3 Bin-Packing Problem

The purpose of BPP is to pack all the requested objects into the available bins while utilising the fewest available bins possible. In other words, stacking the requested objects into the available bins reduces the number of used bins. The objects here are the VMs, and the bins are the PMs.

Thus, in our work, we will utilise the BPP to obtain secure VMs allocation while reducing the number of used PMs. To define it formally, a group of VMs, denoted as $[VM^1, \ldots, VM^n]$, with different requested sizes; hence, the set of VMs is both heterogeneous and non-heterogeneous for our model. The goal is to pack the requested VMs into a group of available PMs, denoted as $[PM_1, \ldots, PM_k]$, with different available capacities; hence the PMs resources are heterogeneous. However, BPP not classified as a secure-aware packing problem. For example, the first fit is a heuristic of BPP that require modification to be a secure-aware allocation algorithm [73]. Thus, our proposed model is an approximation of BPP that aims to obtain a secure VMs allocation.

4.3.4 Secure VMs Allocation with Minimum VMs Migration

In this section, we will explain the proposed secure VMs allocation model with minimum VMs migration. Moreover, we will explain the representation of the multidimensional resource assumed in our model.

VMs Allocation Model

We will start by defining the VMs allocation model to a selected PM, i.e., assigning a VM to PM that satisfies the required resources of the requested VM. The model of the VM allocation denoted as a function $A_u : V \rightarrow P$, i.e., an allocation is an assignment of VMs to PMs. A VM 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, then PM_j can satisfy the resource requirements of VM^i, i.e., $A_u(VM^i) = PM_j \Rightarrow R(PM_j) \geq N(VM^i)$. The system will keep tries to allocate any unallocated VM, VM_v, to some PM_j until all the VMs in V are allocated.

VMs Allocation Space

Following, we will define VMs allocation space; an allocation space $\mathcal{A}$ is the set of all possible allocations. We can then view the allocation system as a transition system $(\mathcal{A}, \mathcal{M})$, with $\mathcal{M}$ being the set of transitions between allocations. The transition from an allocation A_u to an allocation A_{u+1} is called a *migration*. A system execution is an infinite sequence of allocations $A_1 \ldots$, where $(A_u, A_{u+1}) \in \mathcal{M}$. If the sequence is finite, it can be made infinite by infinitely repeating the final allocation. Alternatively stated, depending on the system and VMs allocation algorithm configurations, the system execution sequence can be terminated at some point during the execution or continues as long as its objective is not reached. The set of VMs that are migrated during a migration (A_u, A_{u+1}) in execution is given by:

$$Move(A_u, A_{u+1}) = \{v | A_u(v) \neq A_{u+1}(v)\} \tag{4.3}$$

Where the $Move$, in Eq. (4.3), is the set that define the migration of VMs allocation in an allocation space. In order to consider an allocation as VMs migration, the allocation A_u of VMs, v, should not be equal to the allocation A_{u+1} of the VMs v. The v captures the current set of VMs allocated during the VMs triggering process.

Secure VMs Allocation

Subsequently, we will define the secure VMs allocation, which aims to find a VM allocation for a target VM separately from a malicious VM.

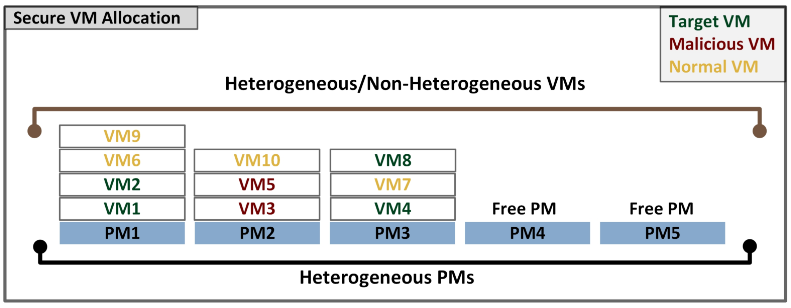


Figure 4.6: Secure VMs Allocation

As illustrated in Figure 4.6, We say an allocation A_u is *secure* if $\forall m \in M, \forall t \in T : A_u(m) \neq A_u(t)$, i.e., an allocation is secure if no malicious VM is co-located with a target VM.

In addition to obtaining a secure VMs allocation, our model aims to minimise the number of used PMs by utilising the BPP as described in Section 4.3.3. The motivation to use fewer available PMs is to allow more spaces available for the upcoming VMs, therefore increasing the chance of finding secure PMs available. In other words, for each VMs request, and by allowing more spaces available while allocating the requested VMs, the chance of a malicious co-residency to occur is potentially lower because the allocation algorithm will have more options to obtain a secure allocation in the available free PMs. After all, the behaviour of allocating the VMs, as described in BPP, will follow the stacking-based approach.

On the other hand, the stacking approach might leads to a significant potential leakage through SCA. To recall the SCA definition, defined in Section 4.2, SCAs are the attacks that occur on communication channels between processes that may leak sensitive outputs from a process on the shared PM. SCAs depends on the time spends on shared PM and the amount of information gathered through the compromise side channel to profile the target VMs and consequently perform the attack. Therefore, As a consequence stacking approach, and if the malicious VMs manage to be stacked, shared PM, with many target VMs, the amount of sensitive data that can be captured is substantially beneficial for the attacker.

Thus, it is vital to reduce the chance of malicious co-residency, and most importantly, reduces the number of malicious VMs and target VMs sharing the same PM when the malicious co-residency inevitable. Alternatively stated, if a malicious co-residency was inevitably occurring, then another solution should be proposed to reduce this effect, which will be described and discussed in the VMs migration model.

Malicious VMs allocation

Moreover, we will define the malicious VMs allocation where the target and malicious VMs co-located on the same PM. An allocation that is not secure is termed as a *malicious co-resident* allocation.

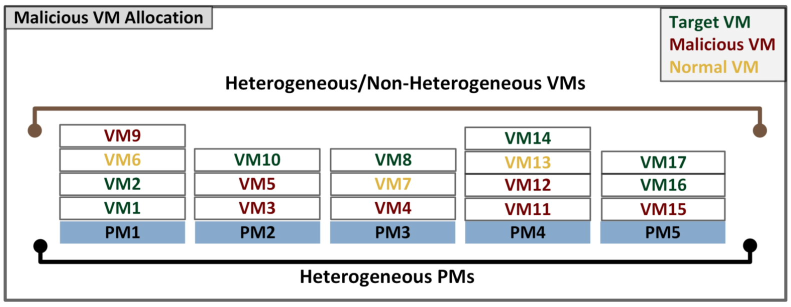


Figure 4.7: Malicious VMs Allocation

The set of PMs at which malicious co-residency occurs is denoted by:

$$CoRe(A_u) = \{p \in P | A_u(t) = A_u(m) = p, t \in T, m \in M\} \tag{4.4}$$

As illustrated in Figure 4.7, the defined $CoRe(A_u)$ is the allocation A that produce a set of VMs where the target VM, denoted as t and malicious VM, denoted as m are sharing the same PM, denoted as p. This malicious co-residency will lead to a potential SCA which eventually leads to compromise the security target VMs. Thus, the goal of our model is to obtain VMs allocation where the $CoRe(A_u)$ is minimised. Moreover, the set of available PMs, P, is utilised to the minimum as possible.

Secure VMs Allocation with Minimum VMs Migration

In our model, and as depicted in Figure 4.8, we aim to produce a secure VMs allocation and consequently a secure VMs migration. As we described in Section 4.1.5, we utilise the VMs migration to preserve the protected status of the current VMs allocation, i.e., keeping the allocation secure before and after the migration. However, we aim to avoid triggering many VMs migration as it will lead to interruption to the VMs running state and possible SLAs violation.

Definition 4.3.1 (VMs 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_u, obtain a secure allocation A_{u+1} such that (i) all VMs $v \in U$ are allocated and (ii) $Move(A_u, A_{u+1})$ 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.

Definition 4.3.2 (VMs Migration with minimal co-residency)**.** Given a set P of PMs, a set U of (unallocated) VMs, a set L of allocated VMs, an allocation A_u, obtain an allocation A_{u+1} such that (i) all VMs $v \in U$ are allocated and (ii) $Move(A_u, A_{u+1})$ is minimised and (iii) the number of PMs at which co-residency occurs is minimized, i.e., minimise $CoRe(A_{u+1})$.

Moreover, in the situations where there are no options, lack of available PMs, to obtain a secure VMs allocation, the VMs migration aims to reduce the number of targets and malicious VMs co-residing on the same PM, thus reducing the amount of data that captured by the malicious VM during the SCA.

Therefore, in the VMs migration, we aim to trigger the fewest possible VMs migration that preserves the protected status of VMs allocation. Furthermore, in the occurrence of an inevitable malicious co-residency, we aim to reduce the number of target VMs and malicious on the same PM as much as possible to reduce the amount of leaked information through SCA.

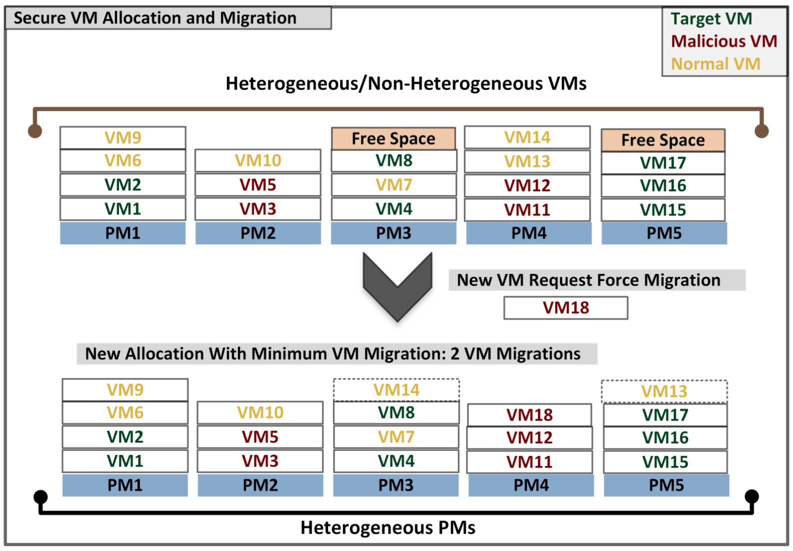


Figure 4.8: Secure VMs Allocation with Minimum VMs Migration

We say that a migration is secure if both the start and the end allocations are secure. Whenever there are malicious allocations 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., $Move(A_u, A_{u+1})$ needs to be minimized.

Resources Representation

In the cloud system, the required resources for the VMs, and the available resources for the PMs are considered multidimensional resources. This multidimensional of resources means that each VM has a set of resources representing the overall resource requirements, which we denoted in our model as $N(VM^i)$. These resources include the size of the RAM, the number of CPUs and their cores, the storage, the network switches, and the OS requirements. Moreover, each PM has a set of resources representing the overall resource availability, which we denoted in our model as $R(PM_j)$. These resources include the size of the available RAM, the number of available CPU processors, the available storage on the PMs, and the network switches availability and reachability.

Therefore, we specify the resources of the VMs and PMs as a unified unit that represents the multidimensional resources as one. In other words, we define the demanded resources of the VMs as $N(VM^i)$, while the available resources of the PM as $R(PM_j)$ to avoid the complication of the structure of the resources. The definition does not ignore the existence of the specific resources or their consideration during the allocation; however, it is simpler to represent the resources in this form.

Furthermore, we assume that part of the resources for VMs and PMs are always available during the allocation. These resources include the network switches availability and reachability, storage capacity, Power supply availability and other resources. However, the essential resources considered to be validated during the VMs allocation are the RAM size and the CPU cores with their sizes. Hence, when we refer to the resources in the thesis, either requires resources from the VMs or available resources of the PMs, we consider this assumption.

4.3.5 Objective Function Formulation

In summation, we will define the objective function and its constraints of the proposed model as an approximation of BPP. As we described in Section 4.3.3, the BPP aims to allocate the VMs into a selected PMs while minimising the used PMs as an objective. However, 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 described as follow:

The objective is ***Minimize***

$$\sum_{i=1}^{n} CoRe(A_u) * x_{ij} \qquad \forall_{i \in V,\ j \in P,\ \textbf{for}(j=1...k)} \tag{4.5}$$

Subject to:

$$\sum_{j=1}^{k} y_j \leq |P| \qquad \forall_{j \in P} \tag{4.6}$$

$$\sum_{i=1}^{n} N(VM^i) * x_{ij} \leq R(PM_j) * y_j \qquad \forall_{i \in V,\ j \in P,\ \textbf{for}(j=1...k)} \tag{4.7}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad \forall_{i \in V,\ j \in P,\ \textbf{for}(j=1...k)} \tag{4.8}$$

$$x_{ij} \in \{0,1\}, \qquad \forall_{\textbf{for}(i=1...n),\ \textbf{for}(j=1...k)} \tag{4.9}$$

$$y_j \in \{0,1\}, \qquad \forall_{\textbf{for}(j=1...k)} \tag{4.10}$$

$$x_{ij} = \begin{cases} 1 & \text{If } VM^i \text{ allocated to } PM_j; \\ 0 & \text{Otherwise} \end{cases} \tag{4.11}$$

$$y_j = \begin{cases} 1 & \text{If } PM_j \text{ selected;} \\ 0 & \text{Otherwise} \end{cases} \tag{4.12}$$

Objective Function

Starting from Eq. (4.5), which aims to minimise the malicious co-residency of a selected possible allocation, i.e., for each possible VMs allocation of a requested VMs and an available PMs, the objective is to select a possible allocation that yields to produce an allocation with minimum malicious co-residency. The term possible allocation refers to all possible combinations of allocations between VMs, as shown in Figure 4.9. Alternatively stated, generate a list of all possible allocations of the requested VMs with each other, regardless of PMs availability. Meaning, at this stage, generating all possible allocations, the relation that determines the possible allocations is between the VMs with each other, not the relation between the VMs with the PMs.

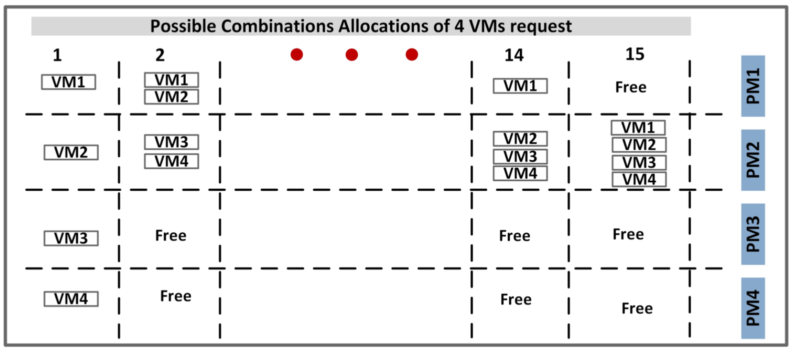


Figure 4.9: Possible allocations for a given four VMs requests.

After generating all the possible allocations, the decision variable (x_{ij}) will select the best allocation from the possibilities list concerning the requested resources for VMs, PMs available resources, and minimising malicious co-residency objectives. The following equation describes the calculation needed for generating all the possible allocations:

$$\sum_{c_v=1}^{t_v} \frac{t_v!}{c_v!(t_v - c_v)!} \tag{4.13}$$

The t_v represents the total number of the VMs, and c_v represents the number of VMs chosen per iteration. The sum of the iterations provides the total number of possible allocations regarding that there are no repetitions for each iteration. To break it down, in Figure 4.9, four VMs requests will produce a total of 15 possible allocations. By applying the Eq.(4.13), the first selection, $c_v = 1$ will produce four possible allocations, in which all the allocations generated are allocated alone. The second selection, $c_v = 2$ will produce six possible allocations, in which all the allocations generated are allocated in a couple, two VMs in each allocation. The third selection, $c_v = 3$ will produce four possible allocations, in which all the allocations generated are allocated in a trio, three VMs in each allocation. The final selection, $c_v = 4$ will produce one possible allocation, in which the allocation generated is allocating all the VMs together. Adding all the possible allocations generated from the four selections trial will produce 15 possible allocations. There are different ways to perform this activity and indeed much simpler, however, our aim is to feed the model with many data as possible to produce a better decision for the allocation. As we stated at the beginning of this chapter, our goal is evaluating the proposed model using LP tool called PuLP [70]. This tool performs better when the amount of data formulated are detailed in order to produce an optimal solution. As such, we perform this step while considering the evaluation of tool and its requirements, which will be explain in Section 4.4.

This step aims to generate possible allocations with different structures, which contribute to obtaining more efficient results and selection by x_{ij}. The Eq.(4.13) is inferred from the formula of combination without repetition [74]. We will explain more about creating and selecting the combinations of possible allocation in the following section, Section 4.4.

Objective Constraints

The first constraint in Eq.(4.6) is to make sure that the selected number of PMs is less than the total number of the available PMs, which means that this constraint will control the use of the number of selected PMs to the limit of the available ones. Moreover, as we stated earlier in this section, the BPP aims to allocate the VMs into selected PMs while minimising the used PMs as an objective, in our model, this objective is achieved though the Eq.(4.5) when the malicious allocations are minimised and therefore, their PMs. The possible allocation function will produce a set of possible allocations of VMs with their malicious co-residency score and combined them with their selected PMs. As such, these two aspects, which are represented as i and j, are minimised during the selection of an allocation. More details about the steps of the selection possible VMs allocations and selected PMs are explained in Section 4.4.1.

The second constraint in Eq.(4.7) 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.(4.8), verifies that each VM is allocated once on a single PM to prevent duplicated allocations. The equations, Eq.(4.9 to 4.12) 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, and it's an integer value.

VMs Migration

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

$$|Move(A_u, A_{u+1})| \leq \beta \tag{4.14}$$

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 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.

4.4 Evaluating Secure VM Allocation Model

In this section we will present an evaluation of the proposed model, in Section 4.3.5, using a tool called PuLP [69, 70]. PuLP is an open-source Linear Programming (LP) solver package that utilises python programming to solve an optimisation problem, refer to **Appendix** A for more details about PuLP process.

Generally speaking, LP is an optimisation method that aims to find the optimal solution of a particular situation, given that this situation has variables and constraints related to each other linearly. Alternatively stated, LP is the method of obtaining the best possible solution of a given problem that is modelled mathematically to find its maximum best or minimum best solution depending on the goal or objective function [80]. For instance, in our model in Eq. (4.5), the objective is to find an allocation that aims to minimise the malicious co-residency in an allocation, subject to a given set of constraints. Thus, we utilise the LP solver to obtain an optimal allocation for a given set of VMs and PMs configuration to understand the behaviour of the optimal allocation for this given situation.

Each objective function has a set of independent variables called decision variables, e.g., x_{ij} in Eq. (4.5) or y_j in Eq. (4.6). These decision variables, in our model, decide and select the best allocation of the set of all the possible allocations given the defined constraints, and they represented as a binary variable. Meaning that the variables either will be 0 or 1, where one means the allocation is selected and zero otherwise. In our evaluation, we utilise an extension to the LP called Mixed Integer Programming (MIP) when some variables need to be defined as integer variables. Additionally, the constraints, which are in Eq. (4.6) to Eq. (4.10), are the variables that define the limitations or the boundaries of the objective function.

In short, the LP structured with an objective function that represents the profit of situations that need to be minimised or maximised. Moreover, decision variables responsible for deciding which solution, of all the available possible solutions, need to be selected as the best one. Furthermore, a set of constraints represent the limitations of the available resources.

Using an LP solver helps to make a better decision for a specific situation under a set of constraints. In our model, the motivation behind using LP for our model evaluation is to capture the behaviour of the optimal solution for obtaining secure VMs allocation. Knowing how to perform such activity, and by capturing the allocation behaviour of the optimal situation helps to develop the algorithm intended for obtaining a secure allocation. In other words, LP helps to understand how to allocate the VMs efficiently under a given set of resources, which leads to minimising the malicious co-location.

In the following section, we present the steps that were made to evaluate the proposed model using PuLP solver:

- We started by introducing an optimisation-based algorithm to produce optimal secure VMs allocation using PuLP solver, called Optimisation-based Secure Allocation (OSA).
- We present the experiment configurations that examines different VMs number and structure scenarios and the PMs number structure. These scenarios study the effect of obtaining the secure VMs allocation on different VMs and PMs configurations.
- Then, we present the results that we obtain from performing the experiments, hence capturing optimal secure VM allocations under different configurations.

4.4.1 Optimisation-based algorithm

This section describes how we utilise PuLP solver to produce an optimal secure VMs allocation under several constraints and scenarios. Hence, evaluating the behaviour of secure VMs allocations for a different set of configurations of VMs and PMs. We introduce an Optimisation-based algorithm, called OSA, as shown in Algorithm 4.1.

The OSA algorithm has two inputs: a set of unallocated VMs (V) and a set of available PMs (P). The set of VMs is structured and classified into three types: target, malicious, and normal VMs. Moreover, each VM in the set of VMs has different resources requirement, which means that the resources requirements of the VMs are heterogeneous. In addition, in the set of PMs, each PM has different available resources, which means that the PMs resources are heterogeneous as well. The details of the structure for the VMs and PMs will be described in the following section, Section 4.4.2.

Algorithm 4.1: Optimisation-based Secure Allocation (OSA)

```
Input: $V$, $P$
Output: $A$: Optimal Secure Allocation
1 Combinations ← Generate All Possible Combinations ($V$, $P$);
2 for $i, j$ in Combinations do
3     Send ($i$) to PuLP to Select ($i$) With Minimum Co-residency while
        Reducing used ($j$).
4     if $N(i) <= R(j)$ then
5         $A$.Add($i, j$);
6     end
7 end
8 return $A$
```

The OSA starts, at line 1, by producing all the possible combinations allocations between VMs. Then, from the produced VMs combinations allocations, generate another combination between each VMs combination with each PM. Thus, generating a possible VM allocation for each VM. Afterwards, assign a co-residency score to each selected combination. To explain, the function, *Generate All Possible Combinations (V, P)*, has three main steps:

1. Generate all the possible VMs combinations allocations of when they possibly allocated together, as described in Figure 4.9. This combination will be only between the VMs set. For example, if the set of VMs contains two VMs, (VM1, VM2), then this step will produce a combination subset of the VMs set to be ((VM1), (VM2), (VM1, VM2)). Meaning, each VM, VM1 and VM2, will allocate separately or be allocated together. We called the produced VM combinations allocations (*VMsComb*). The number of possible allocations is calculated in Eq.(4.13).

   This step is inspired from the set partitioning problem, which is having a set of elements divided into subsets of partitions, which belong to the main set. The selection of the partitions depending on the required situation of either maximizing or minimizing the objective function [35].

2. After generating VMs combinations allocations, (*VMsComb*), we repeat the first step with each combination in (*VMsComb*) with each available PM. For example, if we have two PMs, PM1 and PM2, and using the same (*VMsComb*), this step will produce a combination as follow:

((PM1, VM1), (PM1, VM2), (PM1, (VM1, VM2)), (PM2, VM1), (PM2, VM2), (PM2, (VM1, VM2))). We called this produce combination as (*PMsToVMsComb*).

3. Last step, the algorithm will assign a co-residency score to each selected combination of the set(*PMsToVMsComb*). The score is calculated based on the malicious co-residency between target and malicious VMs. In other words, for each (*PMsToVMsComb*) combination, if a combination has a malicious co-residency between target and malicious VMs, then the score will be assigned based on the number of targets and malicious VMs. For example, if we allocate one target VM with one malicious VM, then the score will equal 1. If the malicious VMs are two, then the score will equal 2, and so on. Therefore, the score of co-residency is calculated as follow: The number of target VMs multiplies in the number of malicious VMs for each (*PMsToVMsComb*). The normal VMs does not affect the co-residency score; thus, a normal VM can be allocated with either the target or the malicious VM without affecting the score of co-residency.

Afterwards, from line 3 to line 5, and after generating all the possible combinations, using *Generate All Possible Combinations (V, P)*, the algorithm starts selecting the optimal allocation. The previous step helps the PuLP solver select the optimal allocations, based on the defined set of constraints, among all possible allocations. As we described in the model objective, in Section 4.3.5, we want to produce a secure VMs allocation while utilising the minimum possible number of PMs. Also, the total selected VMs required resources should not exceed the available PMs resources. Moreover, and as described in Eq.(4.8), each VMs allocation should only be selected once to avoid any duplication and wrong produced allocation by the PuLP solver.

Therefore, at this step, from line 3 to line 5, the PuLP solver will select a (*PMsToVMsComb*), where the selected allocation, i, j, aims to minimise the malicious co-residency, select a fewer number of PMs as possible, and make sure the resources required for both VMs and PMs are met. The i denote the selection of the VMs combination, and j denotes the PM selected for the selected VMs combination. The selected allocation, i, j, will be added to the allocation list, A, and consider a final allocation.

4.4.2 Experimental setup

This section will describe the experimental setup for evaluating the proposed (OSA) algorithm, hence evaluating the behaviour of secure VMs allocations. In general, we consider the number of VMs ranges from 12-60, increasing by 12 VMs in each experiment. The structure of the VMs is heterogeneous, meaning the VMs requirement of resources is different from each other.

Furthermore, we consider the number of each VMs type in each experiment, i.e., each experiment has a different setup of how many target, malicious or normal VMs. For the PMs, we consider the PMs ranges from 5-40 in each experiment. The PMs structure, or level of PMs heterogeneity, is high, medium and low heterogeneous PMs. Meaning the resources of the PMs are structured based on the classification of PMs heterogeneity.

The motivation behind designing the experiment in this form is to study the VMs allocation behaviour under different scenarios and configurations. In summation, which will be explained as well in the following subsections, we consider the following in the setup of the experiment:

- The structure of each VMs type: we consider seven VMs type structures of each VMs type in each experiment.
- The structure of the VMs size (VMs Heterogeneity): we consider heterogeneous VMs resources for the VMs, and the assignment of the resources will be according to the available resources of the PMs. Meaning the total number of resources of all the VMs should be less or equal to the available resources of the PMs.
- The PMs size (PMs Heterogeneity) structure: we consider three types of PMs structure: high, medium and low heterogeneous PMs.

VMs Structure

The number of VMs ranges from 12-60 VMs, and for each VMs number, for example, 12 VMs, we conducted several experiments under different scenarios. The VMs number are increases by 12 for each possible experiment, yielding to

12, 24, ..., 60. The VMs, for each VM number, is divided into structures of types: number of malicious, target and normal VMs. The number of each VMs type is not configured randomly for each experiment. Instead, it is configured as described in Tables 4.1 and 4.2.

Table 4.1: VMs structure with the existence of Normal VMs

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

Table 4.1 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 12 VMs, then this VMs type number will be structured 7 times, as described in Table 4.1, 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.

Table 4.2: VMs structure without the existence of Normal VMs

Tries No.	% Malicious VMs	%Target VMs
1	20	80
2	80	20
3	35	65
4	65	35
5	50	50

On the other hand, and to make it more challenging for PuLP solver, we ignore the possibility of having normal VMs during VM allocation of a set of VMs. As described in Table 4.2, we present five VMs structures for the malicious and target VMs only, and each VMs type number will be examined for its secure VMs allocation level. The number of possible situations should only be three situations, $2^2 = 4$ minus the zeroes situation, as we only have two VMs types considered. However, we added two more situations to see the effect of the malicious co-residency when gradually increasing the percentage of the two VMs types.

Table 4.3: VMs number and size structure

Tries No.	VMs Number	VMs shape
1	12	2*(1r-2r-2r-2r-4r-4r)
2	24	4*(1r-2r-2r-2r-4r-4r)
3	36	6*(1r-2r-2r-2r-4r-4r)
4	48	8*(1r-2r-2r-2r-4r-4r)
5	60	10*(1r-2r-2r-2r-4r-4r)

In Table 4.3, we described the VMs number and size structure for each experiment, and, as mentioned earlier, the VMs number ranges from 12 to 60 VMs. The resources required for each VM is different from each other; for example, in try 1, we have 12 VMs number, and the shape of resources is as follow: the first VM require one resource (1r) from the PM, the second VM require two resources (2r) from the PM, and until the sixth VM that require four resources (4r) from the PM. Then assigning the resources will loop through the same set of resources to the rest of the VMs; for instance, the seventh VM require one resource (1r) from the PM, and so on. For each VMs number, assigning the resources will follow the same approach regardless of the VMs type. Therefore, in some experiments, the malicious VMs may have a more significant number of VMs resources, potentially affecting the overall secure VMs allocation.

PMs Structure

As described in Table 4.4, the number of PMs ranges from 5-40 PMs, and for each PMs number, we conducted several experiments under different VMs and PMs structure scenarios. We divided the PMs structure into three categories or PMs heterogeneities; high, medium and low PMs heterogeneous. The PMs heterogeneity describes how much the PMs are different from each other regarding available resources. Hence, our objective is to examine the effect of this difference on obtaining a secure VMs allocation for each possible scenario.

Additionally, the low PMs heterogeneous structure, try 1 to 5, considers having one PMs resource structure, six resources (6r) for all the five PMs. Therefore, this structure makes the PMs available resources non-heterogeneous. The medium heterogeneous, try 6 to 10, consider having two structures of PMs resources, (4r) and (6r), which makes the structure of the PMs more heterogeneous than the first one. The high heterogeneous, try 11 to 15, consider having three structures of PMs resources, (2r), (4r) and (6r), which makes the structure of the PMs the most heterogeneous structure, as we assumed defined.

Table 4.4: PMs number and size structure

Tries No.	PMs Number	PMs Heterogeneity	PMs shape
1	5	Low	5(6r)
2	10	Low	10(6r)
3	15	Low	15(6r)
4	20	Low	20(6r)
5	25	Low	25(6r)
6	6	Med	3(4r)-3(6r)
7	12	Med	6(4r)-6(6r)
8	18	Med	9(4r)-9(6r)
9	24	Med	12(4r)-12(6r)
10	30	Med	15(4r)-15(6r)
11	8	High	3(2r)-3(4r)-2(6r)
12	16	High	6(2r)-6(4r)-4(6r)
13	24	High	9(2r)-9(4r)-6(6r)
14	32	High	12(2r)-12(4r)-8(6r)
15	40	High	15(2r)-15(4r)-10(6r)

Furthermore, the number of PMs are different for each structure; for instance, the low PMs heterogeneous starts from five PMs while the high PMs heterogeneous starts from eight PMs. There are two reasons for this difference; firstly, we wanted to have the same amount of available resources for the PMs on the same try. For example, the first try of low structure PMs has five PMs that can accommodate 30 VMs in total, according to the defined VMs size structure. The same for the high and medium PMs structure, which has eight PMs and six PMs, respectively, can also accommodate the 30 VMs. The second reason, we wanted to examine the effect of the PMs number on the secure VMs allocation. In other words, is having more PMs number will yield to more secure VMs allocation or the opposite. Thus, we design the PMs structure to examine more possible situations of the VMs allocation and capture the behaviour of the optimal allocation of these situations.

Experiments Methodology

Overall, and considering all the possible VMs and PMs configurations described earlier, we conducted 540 experiments to study and examine these configuration's optimal secure VMs allocations behaviour. The experiment method examines each possible combination of each defined structure of the VMs and PMs. It means each possible structure of the VMs number, VMs type, PMs number, and PMs heterogeneity will be examined. For example, VMs number 12 in Table 4.3 will be examined against all possible PMs structures and PMs available resources and heterogeneity, described in Table 4.4. Furthermore, each PMs structure will be examined against each VMs type structure described in Tables 4.1 and 4.2. Thus, we aim in this defined configuration to explore the behaviour of the optimal secure VMs allocations of each possible configuration.

It is worth mentioning that the increasing of VMs number leads to fewer experiments possibility we perform. For example, for 12 VMs, we compare it against PMs number from 5 until 40 PMs because the availability of the resources of the PMs allows it. However, for the 60 VMs number, we only can compare it against the 25, 30 and 40 PMs in low, medium and high PMs heterogeneous situations, respectively. Because the availability of the resources of the PMs only allows these three situations.

4.4.3 Model Evaluation Results and Discussions

This section presents the results obtained to show the effect of the secure VM allocation and PMs usage of an allocation. It examines the effect of the existence and non-existence of normal VMs, and the VMs type number structure on the overall secure VMs allocation under each type of PMs and VMs configuration. It also presents the optimal allocation behaviour to study how the VMs allocated securely in optimal situations. Moreover, it will discuss some of the limitations of conducting the experiments using the presented PuLP solver in the matter of performance of the results.

Malicious co-residency

We calculate the percentage of PMs with malicious co-residency, denoted as (M_{pms}), as follow:

$$M_{pms} = \frac{I_{pms}}{U_{pms}} \tag{4.15}$$

Where the (I_{pms}) specify the infected used PMs, and the (U_{pms}) specify the total used PMs for an allocation. The U_{pms} is not the same as the total available PMs; in most cases, it is less, in our model, because the model depending on reducing the number of PMs by utilising BPP approach.

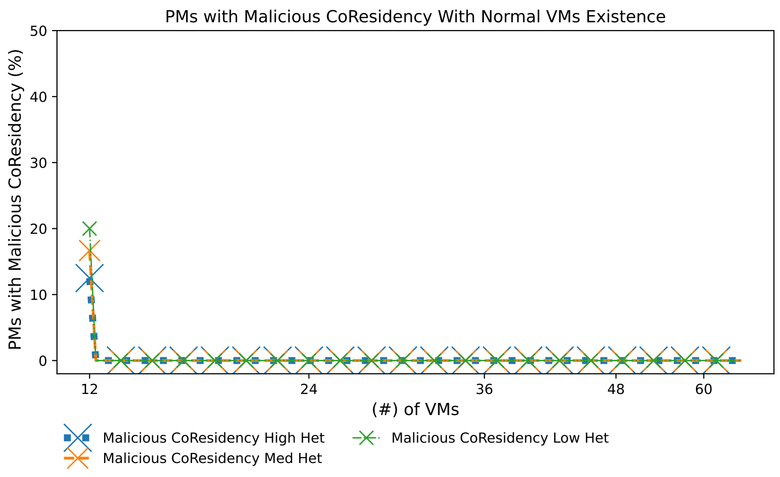


Figure 4.10: Percentage of Malicious Co-residency with normal VMs.

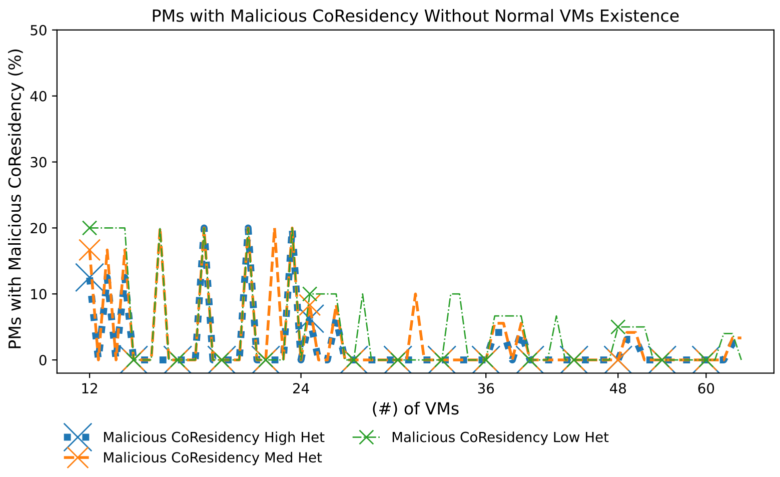


Figure 4.11: Percentage of Malicious Co-residency without normal VMs.

Overall, in Figures 4.10 and 4.11, the M_{pms} of the experiments where the normal VMs existed is performing better than when the normal VMs are not. In Figure 4.10, only three experiments, out of 315 experiments, showed malicious co-residency spikes, and these spikes only occur when the available PMs resources are limited. The 12 VMs situation shown in Figure 4.10 starts from when the PMs number is limited, meaning the required resources of VMs equal the available resources of the PMs. Then, for the same 12 VMs number situation, the number of PMs gradually increases until it reaches 40 PMs.

On the other hand, in Figure 4.11, 60 experiments showed a malicious co-residency out of 225 experiments. In this situation, some of the spikes of M_{pms} are related to the limited PMs spaces; however, some are merely related to VMs type structure being limited to target and malicious VMs, which leads to fewer possible solutions for the algorithm. Thus, for an allocation where the VMs type structure has a more diverse structure, it potentially leads to more secure VM allocation.

Moreover, the increasing number of VMs has a positive impact on the overall malicious percentage. It is expected that the increased number of PMs available resources leads to more secure VMs allocation because of the increased number of chances for the possible allocations. It was not expected to obtain this positive impact when the number of VMs increased, even with the limited PMs resources. In Figure 4.11, for example, we can see that the number of M_{pms} spikes are less when the number of VMs increases, which indicates that the more VMs are allocated, the more possible allocations. The more possible allocations lead to a better result, and therefore, secure VMs allocation.

Furthermore, we investigate the M_{pms} spikes in Figure 4.10 by examining the allocation behaviour of one of the situations where the malicious co-residency occurs. In Figure 4.12, five PMs with the same available resources, therefore, low heterogeneous PMs structure, and 12 VMs where the majority of them are target VMs. The N, T and A stands for normal, target and malicious VMs, respectively. The normal VMs require higher resources than the malicious VMs, and the target VMs are mixed.

Arguably, we can say that there is a negative impact of normal VMs existence on the M_{pms} for this single situation. However, it is only one situation that also occurs on high and medium PMs structures. A possible reason for this result is that the PMs available resources configuration is not high enough to accommodate the required VMs resources. It can not accommodate more than four VMs with small resources or more than two VMs with medium to high resources. Therefore, when the PMs resources are limited, it will be more secure to avoid allocating VMs where the demanding of their resources prevents the scheduler from having more possible allocation choices to allocate more VMs on the same PM, as it happens in this situation.

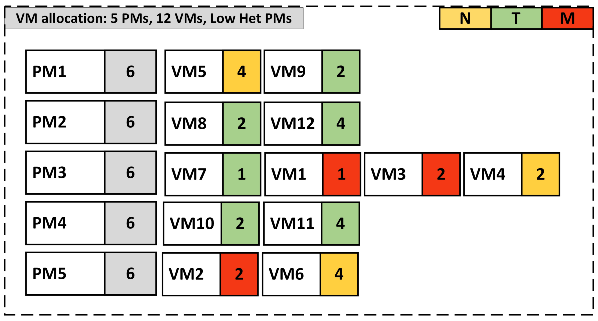


Figure 4.12: A Malicious Co-residency Allocation With Normal VMs.

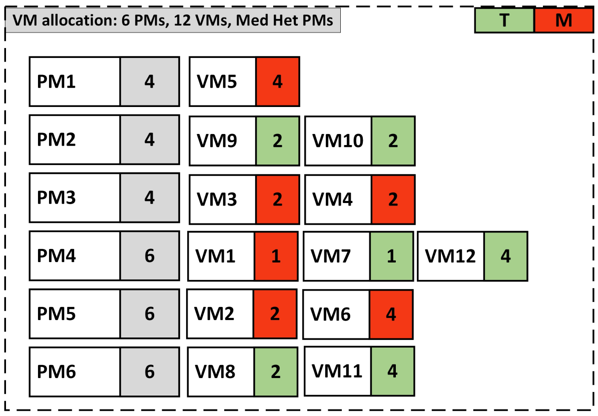


Figure 4.13: A Malicious Co-residency Allocation Without Normal VMs.

Similarly, in Figure 4.13, the possible allocation choices are limited due to the PMs structure and the VMs type structure. In this scenario, we have the only target and malicious VMs for an allocation and the PMs structure is medium heterogeneous. The number of malicious and target VMs are the same as well as their demanding resources. For this situation, a possible solution is to avoid allocating the VMs with the same structure of VMs resources, when PMs resources are limited, at the same time, or on the same structure of PMs. The VMs scheduler could be configured to allocate the VMs with minor requires resources on the same PMs cluster if it offers high possible allocations. The idea here is not to have extra resources from the available PMs. Instead, it has more possible allocations or more choices. Moreover, a potential solution is to delay the allocation of some VMs, meaning change their arrival times in a way that leads to gain a secure allocation.

PMs usage

In our model, we aim to obtain a secure VM allocation while reducing the number of PMs used for an allocation, thus, utilising BPP for this part. Therefore, We calculate the percentage of used PMs compared to the total available PMs, denoted as ($Usage_{pms}$), as follow:

$$Usage_{pms} = \frac{U_{pms}}{T_{pms}} \tag{4.16}$$

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

In Figures 4.14 and 4.14, and as described previously, the experiments start from the 12 VMs case where they require resources of the VMs are equal to the available resources of the PMs, thus, a limited number of PMs. Then we increase the PMs available resources gradually, for the same 12 VMs cases, until we cover all the defined configuration of VMs types and PMs heterogeneity. Therefore, it is normal and expected to have a high $Usage_{pms}$ at the beginning of each VMs case. The goal is to capture the effect of $Usage_{pms}$ when there are limited and less limited resources.

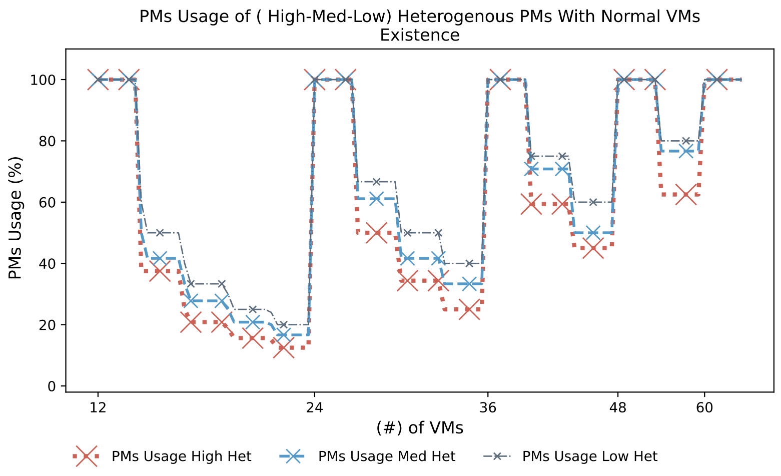


Figure 4.14: Percentage of PMs Usage with normal VMs.

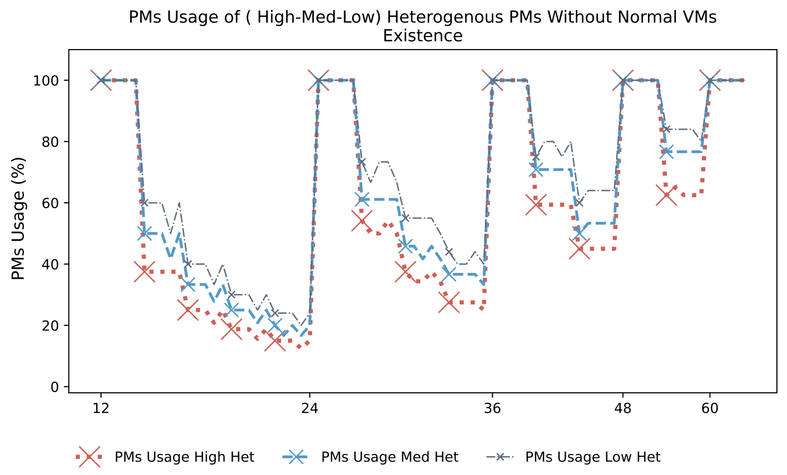


Figure 4.15: Percentage of PMs Usage without normal VMs.

Overall, in Figures 4.14 and 4.15, the $Usage_{pms}$ is almost the same with slight spikes in Figure 4.15, where the VMs allocation algorithm only has targets and malicious VMs to allocate. It means that, for the identical VMs and PMs number, and when there is no normal VMs available, the scheduler tends to utilise more PMs for an allocation. This higher utilisation of PMs, in some cases, is because the possible choices of finding a secure allocation are limited and, therefore, require more PMs.

It is expected that the PMs in high heterogeneous, in the red line, is having lower $Usage_{pms}$ because the number of PMs is higher than other configurations, as explained earlier. Furthermore, for comparing the three PMs heterogeneous structures on both Figures, the number of spikes in low heterogeneous PMs larger than the number of spikes in medium and high heterogeneous structures is noticeable. These spikes situations could potentially lead to a malicious co-residency even if there are more spaces in the PMs. In another word, the number of cases where the VMs allocation utilises more PMs is becoming larger when the available resources structure of the PMs becoming similar. Meaning, for identical VMs, require resources and type structure, but with different PMs structures, such as high heterogeneous PMs, the possible situations of secure VMs allocation occurring are higher. Because when utilising fewer PMs leads to free spaces for the oncoming VMs, a more significant possibility of secure VM allocation is possible.

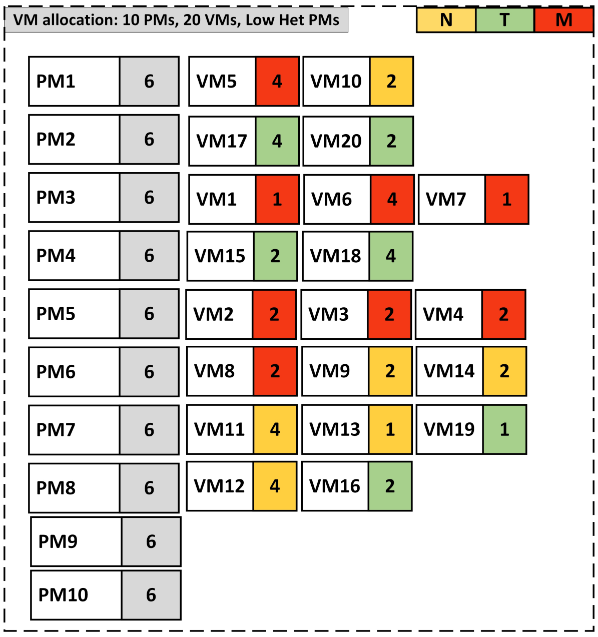


Figure 4.16: The effect of VMs type structure on PMs utilisation with normal VMs.

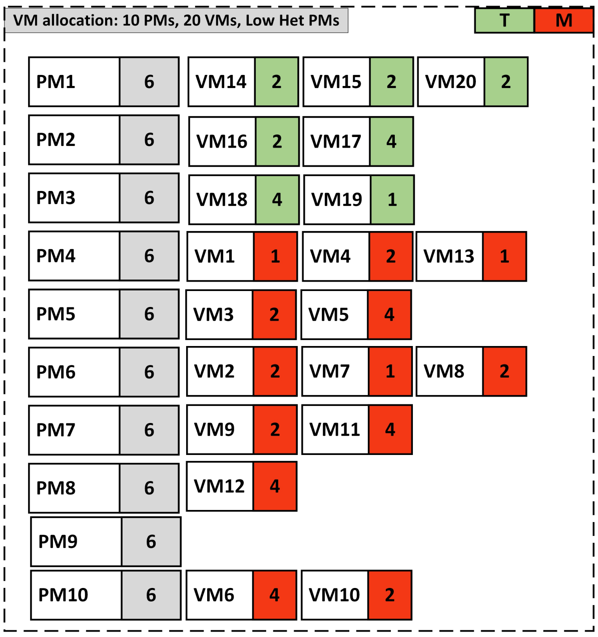


Figure 4.17: The effect of VMs type structure on PMs utilisation without normal VMs.

For example, in Figures 4.16 and 4.17, we demonstrate similar situations of VMs allocation to show the effect of the VMs type structure on utilising more PMs during the allocation. The VMs number, PMs number, and PMs heterogeneous structure are similar in the two situations, but with different

VMs type structures. It is clear that when trying to separate two competeing VMs types, target and malicious, the VMs allocation will utilise more PMs than when there is a third neutral VM type between them. It creates flexibility for the VMs allocator even though the normal VMs utilised the PMs resources and allocating more resources. Otherwise stated, utilising more VMs types, with more potential VMs resources, during the VMs allocation is not necessarily leading to utilising more PMs. Therefore, leading to more accessible spaces for the upcoming VMs and more secure VMs allocation.

Effect of VMs type number and PMs structure on percentage of malicious co-residency

In this part, we will examine, in more detail, the effect of VMs type number and PMs structure on the M_{pms}. Specifically, in connection to the situations in Figure 4.11, which shows the effect of M_{pms} when the VMs type is structured without normal VMs.

In most cases of Figures 4.18, 4.19 and 4.20, the M_{pms} is higher when the number of VMs is small compared to a lower M_{pms} when the number of VMs is large. In other words, the more significant number of VMs are allocating simultaneously is potentially more secure than dividing the VMs into small numbers then allocating them. Thus, the number of VMs contributes significantly to obtaining a secure VMs allocation when it is carefully selected and managed. This outcome is because the VMs are allocated using PuLP solver, and the small number of VMs led to fewer possible solutions. As we explained in the Algorithm 4.1, the allocator will select the optimal secure allocation of the generated possible allocations, and when the number of VMs is high, then we can obtain more possibilities to select from, hence more allocations of a secure allocation. In the case of PuLP allocation methodology, it potentially yields a more secure allocation if the allocation algorithm is modified to allocate the VMs differently.

Furthermore, it is conspicuous that the high heterogeneous PMs in Figure 4.18 have fewer cases of the malicious co-residency than medium and low, in Figures 4.19 and 4.20, respectively. This outcome is due to the high heterogeneous PMs offers more PMs possibilities than medium and low heterogeneous cases. Therefore, allocating PMs with a more diverse structure of available resources has less M_{pms} than the less diverse PMs structure.

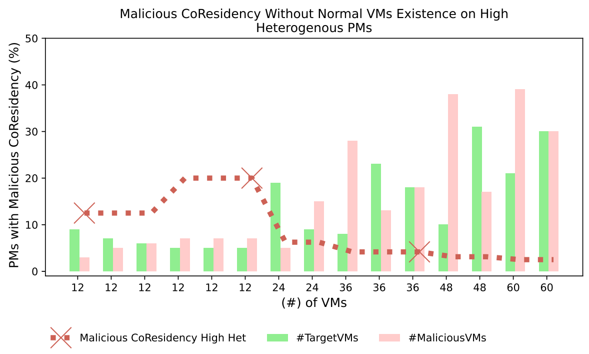


Figure 4.18: High Heterogeneous Malicious Co-residency without Normal VMs.

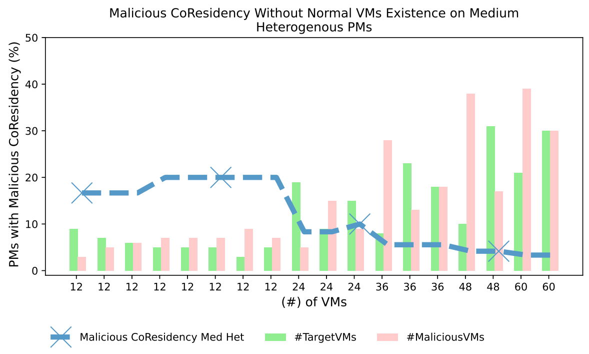


Figure 4.19: Medium Heterogeneous Malicious Co-residency without Normal VMs.

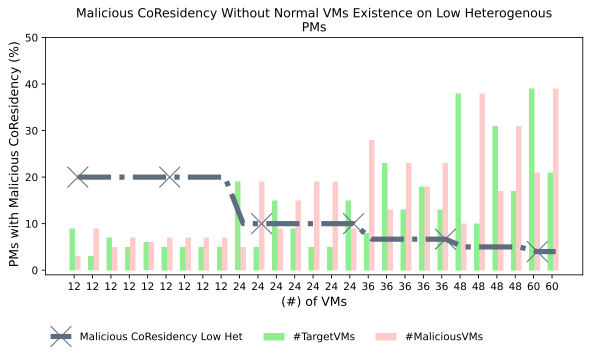


Figure 4.20: Low Heterogeneous Malicious Co-residency without Normal VMs.

Moreover, concerning the VMs type number, there is no clear indication that changing the number of either target and malicious VMs yields secure allocation. It is more than just simply making the allocation decision based on the number of the VMs type. Because there are other important factors to consider, such as the amount of the require resource for each VMs type and the PMs available resources to consider. As we saw in the allocation example in Figure 4.13, each of the requires resources of the VMs contributed to the outcome of the malicious allocation. In other words, if the resources requirements for VMs and PMs are managed to produce the right balance between VMs and PMs resources, it will lead to a secure allocation.

Therefore, in our model, we focus on producing a VM allocation that carefully manages the resources to obtain a secure allocation. Alternatively, we aim to find the correct balance between the required VMs resources and the PMs available resources to produce secure VMs allocation.

Optimal allocation behaviour

In this part, we will examine the behaviour of the optimal VMs allocation conducted by the PuLP solver to learn how the secure VMs allocation is performed optimally. This study's outcome contributes to the development of the allocation algorithm, which we will present in the next chapter of the thesis. The optimal allocation behaviours are produced by PuLP under the defined constraints, the set of VMs, PMs and allocation scenarios. The term optimal is used by PuLP to indicate that the produced solution complies with the defined objective and constraints, and it will be not optimal if it is not. For example, if some of the VMs in the allocation exceeded the PM's available resources due to failure in the selection, then the final solution will be not optimal. When we refer to the allocation as optimal, it does not necessarily mean that this allocation is the perfect one; however, it will be one to consider as an allocation that complies with the defined objective. However, the PuLP tool is an optimisation tool, and its goal is to generate the best solution of a defined objective and constraints; hence we call the outcome optimal.

As mentioned earlier, we conducted 540 experiments to examine different VMs and PMs configurations to learn the optimal secure VMs allocations behaviour.

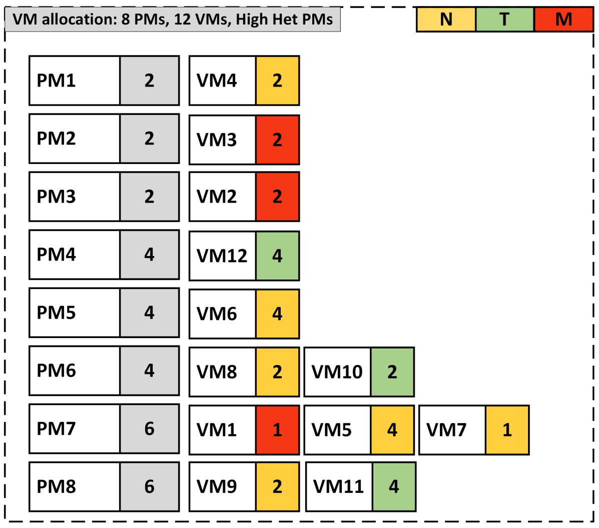


Figure 4.21: An Example of Optimal Allocation Behaviour.

In most cases, and as shown in Figure 4.21 as an example, the allocator tends to allocate VMs on PMs with the exact resources configuration. In other words, it selects the PMs with resources that match the required resources of the VMs. For instance, a VM requires two resources; it will be allocated to a PM with the exact available resources. This behaviour creates a perfect match between the VMs needs and the available resources, and it is expected. Because, in our model, we aim to reduce the number of used PMs while producing a secure allocation; therefore, by utilising the BPP, the allocator will stack the VMs into the available PMs. This stacking behaviour allows to free more PMs for the oncoming VMs and, accordingly, allows for more secure allocation possibilities.

Moreover, the stacking behaviour is also applied to a group of VMs allocating at the same time. For example, if two VMs request yields to a secure allocation and perfect match of recourse between the VMs and the selected PM, this allocation will be confirmed.

Furthermore, the allocator tends to select the active PMs rather than the idle ones when allocating the VMs. For instance, when a VM requires two resources and two PMs are available, the first PM has already three VMs allocated with available resources that are sufficient for the requested VM. Moreover, a second PM is free of VMs and have sufficient resources as well. Then, in this case, the allocator will select the first PM as its already host group of VMs and most importantly, this allocation will lead to filling the first PM to its maximum potential resources.

Additionally, in some cases, the allocator aims to produce an allocation where the result of the VMs to PMs allocation yields less space in the PMs. In other words, a VM allocation that yields free space on the selected PM will not be prioritised over another VM allocation that yields to consume all the available space on the PM.

In summation, the allocator tends to follow a stacking-based behaviour when allocating the VMs. This stacking approach yields fewer PMs than the available and produces secure allocations for the oncoming VMs.

Performance Limitation

In this part, we highlighted some of the performance limitations of utilising the proposed OSA algorithm, Algorithm 4.1, under the environment of PuLP.

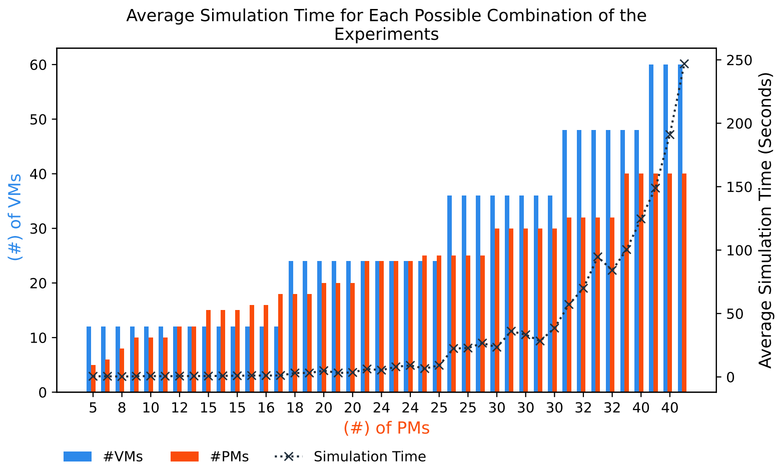


Figure 4.22: Average Simulation Time for Each Possible Combination of the Experiments.

We were only able to examine a limited number of VMs and PMs during the proposed experiments, as a result of the performance limitation. As illustrated in Figure 4.22, the average simulation time is exponentially increasing when VMs and PMs are increasing. The reason for the accelerated growth in simulation time is because the VMs allocation combinations are increasing. These combinations are generated as part of the OSA algorithm to allow the PuLP solver to select the optimal allocation. However, the more VMs and

PMs number, the more VMs allocation combinations generated, thus, a slower performance of the simulation process. For instance, if the number of VMs is 12 and the number of PMs is 5, the average simulation time will be around 0.5 seconds. When the number of VMs is 24, and the number of PMs is 10, the average simulation time will be around 3 seconds. It means that the average time increased six times when doubling the number of VMs and PMs. As such, when having 100 thousand PMs and the number of VMs is 10 times the number of PMs, the expected average simulation time will be exponentially extended to days, which is not practical. It is worth mentioning that the goal of showing this part of the result is to demonstrate why we examine a small number of VMs and PMs. Moreover, the ultimate goal of this section is to study the behaviour of optimal outcomes; hence, examining a small number of items will be accepted for this purpose.

Summary

In this chapter, we present a cloud data centre model including its components that affect the development of the cloud system. Include the PMs and the VMs and the virtualisation layer that is responsible for allocating the shared recourse in the CCEs. We also present model definitions of the method of VMs allocation, VMs migration and how the resources sharing of CCEs computing resources is conducted. Besides, we introduce the model of the VMs secure allocation in CCEs, including the model's objective function and its constraints. The defined objective function aims to obtain a secure VMs allocation to defend against SCAs by minimising the malicious co-residency between the target and malicious VMs. Moreover, we present a framework definition of the cloud user classification based on their behaviour analysis, classifying them into specific types. Also, we presented an extensive evaluation of the proposed secure VM allocation model using PuLP solver, which yields to study and examine the behaviour of the secure optimal allocation under different situations and scenarios. The outcome of this study contributed to formulating the secure VM allocation algorithms, which will be discussed in the next chapter.

Chapter 5

Developing and Evaluating Security-aware VMs Allocation Algorithms

Preface

The previous chapter introduces the secure VMs allocation model that aims to minimise the malicious co-residency in the CCEs. In other words, the model's goal is to minimise the co-residency between the malicious and the target VMs in cloud CCEs while minimising the utilised number of PMs. Moreover, we utilise the BPP to develop a secure VMs allocation to reduce the chance of malicious co-residency while using fewer available PMs in the cloud system. Furthermore, we examine the VMs allocation behaviour of our model to understand how the optimal secure VMs allocation produces its allocation and develop the algorithms from the study outcomes.

In this chapter, we develop two algorithms called Secure Stacking (SS) Algorithm and Secure Random Stacking (SRS) Algorithm that aim to obtain secure VMs allocation in CCEs. The two algorithms follow a stacking-based behaviour in allocating the VMs, thus an extension of the BPP. Moreover, we develop a VMs migration algorithm that aims to enhance the secure VMs allocation process and keep the allocation as secure as possible.

Furthermore, we presented 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 allocation. 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 Bin Packing Based Allocation

In the previous chapter, specifically in Sections 4.3.5 and 4.4.3, we introduce and evaluate the model of our work which is based on the Bin Packing Problem (BPP). The model extended the BPP to develop a secure VMs allocation to reduce the chance of malicious co-residency while using fewer available PMs in the cloud system.

Subsequently, we examine the model by studying the behaviour of the optimal secure VMs allocation using an LP solver called PuLP. As such, we utilised the LP solver to understand the optimal behaviour of VMs allocations of our proposed model, which is based on the BPP, under different situations and scenarios to develop an algorithm utilised in suboptimal situations. To summarise our results, we found out that the allocator tends to follow a stacking-based behaviour when allocating the VMs, meaning allocating the VMs that on PMs that have the available resources match the required resources of the VMs. Moreover, the allocator will stack the VMs into the available PMs, and this stacking approach yields fewer PMs than the available and produces secure allocations for the upcoming VMs.

In brief, as shown in Figure 5.1, the BPP aims to stack the VMs into the available PMs to minimise the number of utilised PMs [58]. Therefore, the selection objective is based on using fewer PMs as possible concerning the other constraints, such as resources constraints. However, this stacking behaviour can be utilised for other allocation objectives, such as minimising the PMs to reduce the power consumption of the PMs. Alternatively, to optimise the available resources as much as possible, in other words, utilising the available PMs to its full potential to avoid resource wastage. Generally speaking, the stacking-based behaviour of the VMs yields different benefits for the VMs allocation outcome based on the objectives defined.

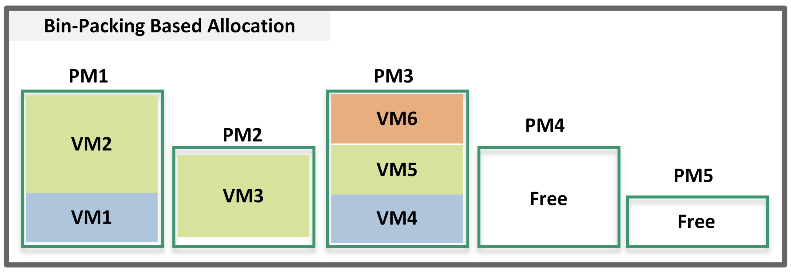


Figure 5.1: An Example of Bin Packing Based Allocation.

Considering the outcome of our model analysis, we develop two VMs allocation algorithms called Secure Stacking (SS) Algorithm and Secure Random Stacking (SRS) Algorithm. The two algorithms, explained in Sections 5.2, and 5.3, are stacking-based behaviour; thus, they extend the BPP. The goal of the proposed algorithms is to develop a secure VMs allocation based on stacking behaviour while minimising the utilised PMs. Generally, in CCEs, the VMs migrations are unwanted events as they lead to service interruption to the VMs while migrating, as explained in Section 4.1.4. Thus, in our proposed algorithms, we aim to develop secure VM allocation that minimises the utilised PMs and minimise the VMs migrations. As a result, the minimisation of the utilised PMs potentially reduces the VMs migration from PMs to another.

5.2 Secure Stacking (SS) Algorithm

The main objective of our work is to address the secure VMs allocations, i.e., develop secure allocation algorithms while minimising the VMs migrations and the number of PMs utilised where co-residency occurs (i.e., Definition 4.3.2). As such, we will endeavour to keep the number of PMs used as small as possible. Thus, using a stacking-based algorithm (e.g., bin-packing) will allow the use of a small number of PMs.

As such, we propose our first security-aware heuristic, a variant of bin-packing, called Secure Stacking (SS), which is shown in Algorithm 5.1. 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.

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.

The process of the SS algorithm starts, at line 4, 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 7 to line 13, 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 14 to line 19, 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 20 to line 28, 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.

Specifically, we will start with the first try and explain its functions, from lines 7 to 13. In line 4, and before starting the first try, the SS will start allocating the VMs one by one from the set of unallocated VMs. Afterwards, in line 5, 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, i.e., computing each PM FR, 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 details of this step will be explained in the following section, Section 5.2.1.

After this step, in line 7, 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.

Algorithm 5.1: Secure Stacking (SS) VMs Allocation

```
   Input: V: Set of unallocated VMs, P: Set of PMs
   Output: A: Secure Allocation
 1 sortedPMsList ← $\emptyset$
 2 COR ← $false$
 3 A ← $\emptyset$
 4 for $v$ in $V$ do
 5     sortedPMsList ← getSortedFRPMs(v,P)
 6     for $p$ in sortedPMs // first try to allocate $v$
 7      do
 8         COR ← getCORvmCheck($v$,$p.getVMslist()$)
 9         if $COR \neq true$ then
10             A ← Assign(v,p)
11             break
12         end
13     end
14     if $v.getPM() = null$ // second try, migrate VMs then retry allocate $v$
15      then
16         vmMigration(sortedPMsList,P)
17         rerun() // repeat steps from 5 - 13
18         break
19     end
20     if $v.getPM() = null$ // third try, allocate $v$ in any available $P$
21      then
22         for $p$ in $P$ do
23             if $p.suitablePM(v) = true$ then
24                 A ← Assign($v$,$p$)
25                 break
26             end
27         end
28     end
29 end
30 return A
```

Then, in line 8, the SS will check if allocating the VM, v, yields a malicious co-residency. According to the presented learning model, in Section 4.1.6, each VM in the CCE 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 details of the **getCORvmCheck(v,p.getVMslist())** will be explained in Section 5.2.2.

The last step of the first try, in line 10, is to assign the v into the selected p if previous conditions are met. Otherwise stated, if there is no malicious co-residency, the assignment of v to the selected p is performed by the function **Assign(v,p)** and added to the secure allocation list, denoted as **A**. The **Assign(v,p)** function will override any previous allocation commitment for the same VM. Meaning if the same VM re-accesses the function and a new PM is selected, it will select this new assignment as final. The process of adding to the allocation list will ensure a unique VM allocation resulted from the **Assign(v,p)** function.

In lines 14 to 19, the second try starts if the first try failed to obtain a secure allocation for the unallocated VM. As indicated in line 14, the second try is triggered if the v is not allocated to a PM yet. The primary step of this try, in line 16, 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 details of the VM migration function will be explained in Section 5.4. After migrating the VMs, the SS will allocate the v by repeating the same steps performed on the first try.

The last step of the SS algorithm is started, in lines 20 to 28, 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 30, all the assignments will be registered in **A** and returned as a final allocation.

5.2.1 Fullness Ratio (FR)

The key factor of the stacking-based behaviour is the fullness ratio (FR) function, which was mentioned earlier in SS algorithm 5.1 at line 5 denoted as **getSortedFRPMs(v,P)**.

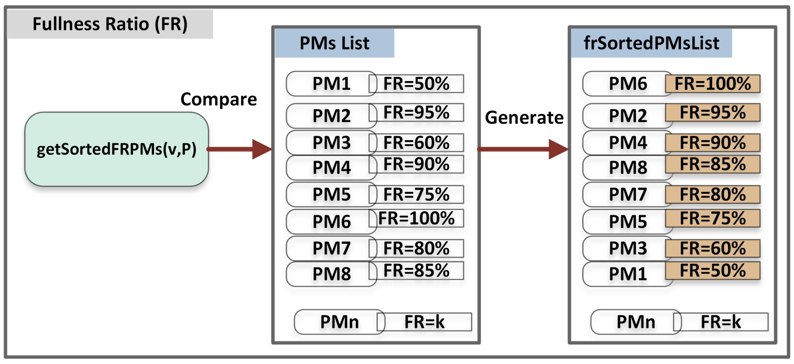


Figure 5.2: Computing the Fullness Ratio (FR) of an Arriving VM.

As illustrated in Figure 5.2, this function sort the PMs that have resources with a high FR compared to the upcoming VM demanded resources, to be selected firstly for the allocation. In other words, the sorted PMs are prioritised to make the highest FR PMs, which will be utilised to its highest available resources, as a first candidate. For example, in figure 5.2, The PM6 is the highest FR PM out of the available PMs for it has high FR among the available PMs, if the upcoming VM is allocated on it. Thus, PM6 will be the first selection for the SS algorithm to find a PM for the unallocated VMs.

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 allows more space for the upcoming VMs to be allocated securely. The score of the FR depends on the current situation of the available PMs resources, the VM required resources and the time that VM arrives.

To explain more about the FR function, we introduce the Algorithm 5.2, to illustrate the computation of the FR for each VM. The FR function, denoted as **getSortedFRPMs(v,pmsList)**, has two inputs: the unallocated VM, denoted as v, and the list of the available PMs selected for an allocation, denoted as *pmsList*. The main objective of the FR function is to sort the available PMs according to their fullness ratio compares to the VM that needs to be allocated.

Algorithm 5.2: Fullness Ratio (FR) Computation

```
1  Function getSortedFRPMs(v,pmsList)
2      frSortedPMsList ← $\emptyset$
3      frMap ← null
4      frScore ← 0
5      for p in pmsList do
6          if p.getAvailableResources() > 0 then
7              frScore ← v.getResources() ÷ p.getAvailableResources()
8              frMap.put(p,frScore)
9          end
10     end
11     frMap ← sortByFrValue(frMap)
12     for pms in frMap.key() do
13         frSortedPMsList.add(pms)
14     end
15     return frSortedPMsList
```

Thus, this function compares the VM required resources with the remaining resources of the PM. The remaining resources are different from the allocated resources or the utilised resources of the PMs. The remaining resources of the PMs are the resources that are not utilised at the time of the VMs arrival. In contrast, the allocated resources are the resources that the PM set up with initially. Moreover, the utilised resources are the resources that other VMs currently uses. Therefore, the FR 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 will refer to the *remaining resources* of the PMs as the *available resources* in the Algorithm 5.2.

In lines 5 and 6, the FR function compares the first PM in the *pmsList*, denoted as p, and checks if this PM has reaming resources. If the p has available resources, or remaining resources, then the comparison with resources of the unallocated VM will start as shown in line 7. At this step, we calculated the fullness ratio score, denoted as *frScore*, as follow; dividing the demanded resources of the v by the available resources of p. The results of *frScore* will determine the priority of the PM in the final produced sorted list. For example, if we have the demanded resources of the VM is 2, the allocated space of the PM is 8 with 4 available resources, then the *frScore* will equal (0.5). In contrast, if the allocated space of the PM is 8 with 6 available resources, then the *frScore* will equal (0.33). Therefore, the highest score will be considered first as it leads to allocating the VM into the PM with the highest FR, thus utilising the PM to its highest potential resources.

Afterwards, the p with its *frScore* will be stored in a map data structure for later processing. The map is linking each PM with its *frScore* compares to the single unallocated VM. Then, from lines 11 to 13, the FR will sort the values of the map based on the highest value of *frScore* with its PM. At last, at line 15, the produced sorted list will be selected as a final sorted PMs list based on its high FR value.

The calculation of FR for multidimensional resources

In the previous chapter, specifically in Section 4.3.4, we describe the resources of the VMs and PMs as multidimensional resources. These resources include the RAM, CPUs with their cores, storage, network switches and other resources. Moreover, we assume that some of these resources are always available during the allocation. For instance, the network switches availability and reachability, storage capacity, Power supply availability and other resources. Furthermore, we consider the RAM size and the CPU cores with their sizes are the essential resources to be validated during the VMs allocation. Thus, we will explain how the calculation of the FR for the multidimensional resources is performed in the FR function, specifically in line 7. The *frScore* computation described earlier is a generalisation of the detailed FR calculation for simplicity.

The detailed calculation of *frScore* described as follow:

$$frScore = (VM_{ram} \div PM_{ram}) + (VM_{cpu} \div PM_{cpu}) \tag{5.1}$$

The idea is to compare the VM with the PM, considering the RAM and CPU specifications. The outcome of the comparison is the *frScore*, which represents the priority of the PM for the VM allocation. If the *frScore* is high for a PM compared to others, this PM is considered a better match for the unallocated VM, which we compare the PMs with.

In Eq(5.1), 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.

5.2.2 Malicious Co-residency Detection

This section will explain the process of the malicious co-residency detection function mentioned in the SS algorithm in line 8. This function, denoted as **getCORvmCheck(v,p.getVMslist())**, aims to check the existence of malicious co-residency between the VMs hosted in cloud systems. Alternatively stated, this algorithm checks if the VMs co-resident in a PM is classified as the target and malicious hosted together on the same PM. The classification of the VMs discussed and described in the learning model in Section 4.1.6.

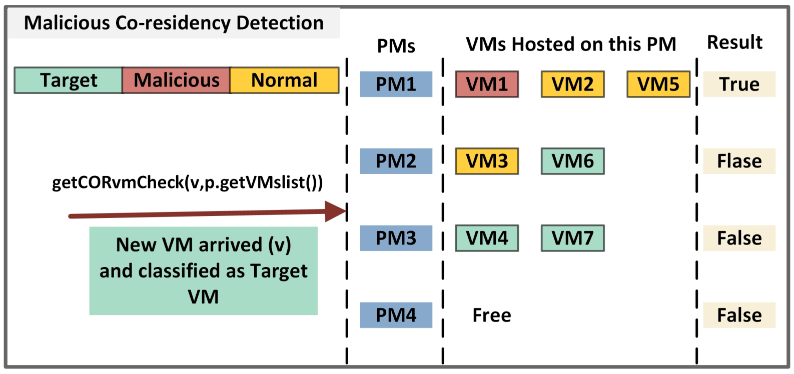

Malicious Co-residency Detection
Target
Malicious
Normal
getCORvmCheck(v,p.getVMslist())
New VM arrived (v) and classified as Target VM
PMs
VMs Hosted on this PM
Result
PM1
VM1
VM2
VM5
True
PM2
VM3
VM6
Flase
PM3
VM4
VM7
False
PM4
Free
False

Figure 5.3: Detecting the Malicious Co-residency of a PM.

As shown in Figure 5.3, the **getCORvmCheck(v,p.getVMslist())** function will compare the upcoming VMs, unallocated VM, with the VMs hosted in the list of selected PMs. The selected PMs are the PMs that are selected for a VM allocation. It depends on the sorting FR function described earlier and the available resources of the PMs. The outcome of this comparison is a Boolean variable denoted as COR to identifies whether the PM is having or will have a malicious co-residency after the allocation. This function tries to avoid allocating the VMs classified as a target with malicious, while the normal VMs can be allocated with any of the two types.

This checking is essential to the SS algorithm as it will be the main responsible for triggering the VMs migration. The VMs migration, will be described in Section 5.4, depends on the event that the upcoming VM could not find an allocation on the available PMs due to malicious co-residency. Therefore, initiating the VMs migration to change the form of the current VMs allocation in order to avoid malicious co-residency, hence avoiding SCA.

5.2.3 Time Complexity Analysis for SS Algorithm

We will focus on our analysis on the time complexity, focusing on the time our algorithms take to complete a function compared to its input length. In this part, we will analyse the SS algorithm's time complexity by calculating the algorithm's worst-case running time. The worst-case scenario time or upper bound limit is the longest time an algorithm can take to complete all its steps. Moreover, we assume that certain parts of an algorithm can take a constant unit of time, such as the arithmetic operations, assignments, comparisons and returns statements. As such, our focus is on the worst cases, called the big-O notation, that the algorithm can take. This approach will be applied to all the algorithms time analysis in the thesis [31].

Two main inputs affect the complexity: the VMs, denoted as N, and the PMs, denoted as M. Moreover, we will calculate the big-O for each try, then add their complexity at the end. In the first try, this block will iterate over all the request VMs to find an allocation for each VM; as such, it will take $O(N)$ times to compete the loop based on the number of VMs. The nested function in this try, $getSortedFRPMs$, explained in section 5.2.1, will take $O(M + M * logM)$ to first calculate the FR score for the available PMs, then sort them compared to the requested VMs resources. This step is nested inside the VMs allocation loop. As such, the big-O of the first try of the algorithm will take $O(N * (M + M * logM))$ to complete.

Then, the second try of the algorithm will consider the summation of time complexity of VMs migration and the rerun functions. As such, the VMs migration function explained in section 5.4, will take $O(M + N * M)$ to the selected VMs for migration from the list of PMs, the VMs that classified either malicious or normal VMs. Then iterate again to each available PM to select the destination PM as a nested loop. Moreover, as explained earlier, the rerun function will take the amount $O(M + M * logM)$. Therefore, and considering this try is nested, it will take $O(N * (2M + N * M + M * logM))$.

The last part, the third try, will take $O(M)$ times to complete the iteration over the available PMs, and since it nested inside of the original loop of the VMs, it will take $O(N * M)$. Therefore, the big-O for the SS algorithm will be $O(4N * M + 2N * M * logM + N^2 * M)$ or $O(M * (4N + 2NlogM + N^2))$.

5.3 Secure Random Stacking (SRS) Algorithm

As we explain in the threat model, in Section 4.2, the SCAs depends on collecting information from normal operations leakage in a shared PM. In essence, the malicious VMs will try to co-reside with the target VMs to extract information from the vulnerable side-channel, such as cache-based attacks. Thus, achieving this goal depends on the VMs allocation algorithm that the CSP utilises to allocate the VMs, and the ability of the malicious user to predict the allocation scheme. Alternatively stated, the malicious VM relies on the information collected from the SCA to understand how the VMs allocated to profile their allocation and migration behaviour. In addition, as we mentioned, profiling aspects related to target VMs usage such as the type of applications and the frequency of running certain operations.

Moreover, the proposed SS algorithm is classified as a deterministic algorithm, meaning that allocating a set of VMs under defined requirements and resources available will leads to the same result over and over. Hence, predicting the outcome of its allocation behaviour under a specific set of allocation configurations is relatively achievable by a sophisticated attacker.

Algorithm 5.3: Secure Random Stacking (SRS).

```
Input: V: Set of unallocated VMs, P: Set of PMs
Output: A: Most Secure Allocation
Function oneAllocation($V$,$P$):
    oneAllocationList ← ∅
    ElectedPMsList ← ∅
    $p$ ← $null$
    for $v$ in $V$ // first try
     do
        ElectedPMsList ← getHighestFRPMs($v$,P)
        $p$ ← getRandomPM(ElectedPMsList)
        if ($getCORvmCheck(v,p.getVMslist()) \neq true$) then
            oneAllocationList.add(Assign($v$,$p$))
            break
        end
        else
            // second try
            vmMigration(ElectedPMsList,P)
            rerun() // repeat steps from 8 - 10
            break
        end
        // third try
        if $v.getPM() = null$ then
            for $h$ in $P$ do
                if $h.suitablePM(v) = true$ then
                    oneAllocationList.add(Assign($v$,$h$))
                    break
                end
            end
        end
        return oneAllocationList
    end
allAllocationsList ← ∅
oneAllocation ← oneAllocation($V$,$P$)
A ← ∅
$InfectedPMs$ ← getInfectedPMsNumber(oneAllocation)
if $InfectedPMs \neq 0$ then
    do
        allAllocationList.add(oneAllocation)
        oneAllocation ← oneAllocation($V$,$P$)
        $InfectedPMs$= getInfectedPMsNumber(oneAllocation)
    while $InfectedPMs \neq 0 \cup simulationTime \leq \phi\ ms$
    A ← getLowestInfectedAllocation(allAllocationList)
end
else
    A ← oneAllocation
end
return A
```

Therefore, we needed to enhance the proposed SS algorithm to make its allocation behaviour difficult to predict and costly for the malicious VM to achieve. In addition, following the same behaviour of the SS algorithm by following a stacking-based VM allocation behaviour. As such, in this section, we propose a secure random stacking-based algorithm, denoted as SRS, enhancing the proposed SRS algorithm. The SRS aims to allocate the VMs securely in a random stacking-based manner. The random behaviour added to the SRS algorithm to make the allocation behaviour non-deterministic, thus, predicting the allocation behaviour difficult for the malicious user to achieve. Moreover, while randomising the allocation, we design the algorithm to follow a stacking-based behaviour, utilise fewer PMs as possible, and reduce the VMs migration number.

The following section will explain the details of the SRS algorithm, described in Algorithm 5.4.

5.3.1 The Process of SRS Algorithm

As shown in Figure 5.4, 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. 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, another allocation will be generated until a time limit is reached. We define the time limit as a ϕ in the algorithm, as it depends on the algorithm's environment, i.e., the machine hosting the algorithm and the number of VMs, PMs. In our case, the time is defined as 2000ms, and the algorithm will terminate after this time is reached. Then, the allocation with the lowest malicious co-residency is selected as the final allocation. This defined limit is considered adequate for our environment that the algorithm implemented on, resulting in around a thousand allocations, which is considered enough for the algorithm's proposal and producing the desired secure allocation. Eventually, the time threshold and the minimum level of co-residency can be adjusted based on the performance requirements for the allocation.

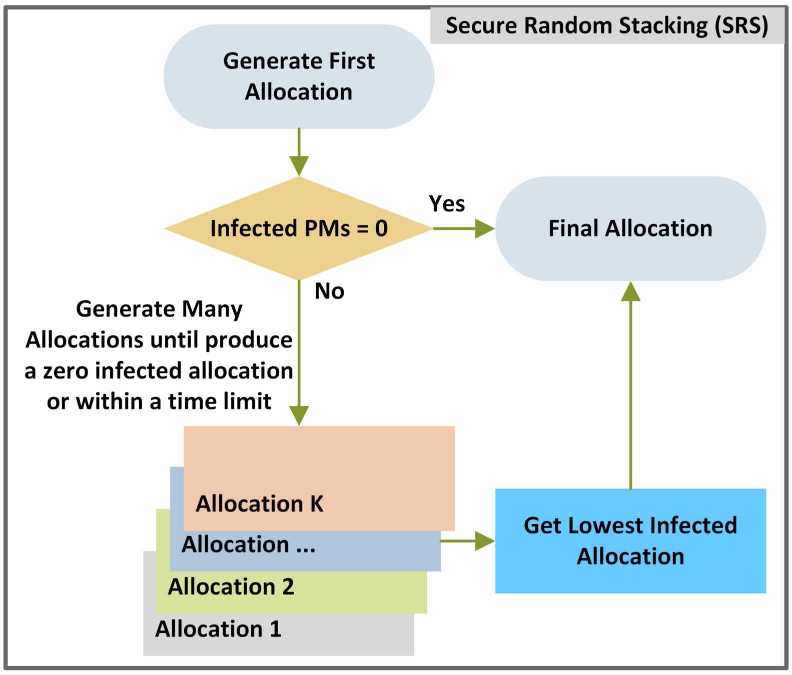

Secure Random Stacking (SRS)
Generate First Allocation
Infected PMs = 0
Yes
Final Allocation
No
Generate Many Allocations until produce a zero infected allocation or within a time limit
Allocation K
Allocation ...
Allocation 2
Allocation 1
Get Lowest Infected Allocation

Figure 5.4: The process of Secure Random Stacking (SRS) Algorithm.

The SRS algorithm has two main inputs: the unallocated set of VMs, denoted as **V**, and the set of the available PMs, denoted as **P**. The output, denoted as **A**, is the most secure allocation found for the available set of resources. The SRS algorithm starts at line 29 when the first allocation is produced using the one allocation function. The goal of the one allocation function is to produce an allocation based on the defined set of resources. The details of the one allocation functions will be described in detail later in Section 5.3.2.

Afterwards, at line 31, the first produced allocation is passed to a function named **getInfectedPMsNumber(oneAllocation)** to calculate the number of infected PMs used in this allocation to compute malicious co-residency. The details of the **getInfectedPMsNumber(oneAllocation)** algorithm will be explained in Section 5.3.4. Then, at line 32, if there is no malicious co-residency, 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, in line 34, another allocation is produced, and the previous steps are repeated until one of two stopping conditions applies: the algorithm can produce an allocation with no co-residency, or the algorithm reaches a timeout limit (ϕ). As we explained, we use a 2000ms 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 the lowest co-residency, at line 38. This step aim to select the most secure allocation produced in a given time limit. The steps of selecting the lowest infected allocations will be explained in Section 5.3.4.

5.3.2 The One Allocation Function

The one allocation function, at line 2, is responsible for many roles: allocate all the unallocated VMs into PMs while maintaining the secure allocation constraints, reducing the number of used PMs and reducing the VM migrations. It is similar to the SS algorithm in the objectives and allocation behaviour; however, it has a different approach in allocating the VMs. The differences are in the method of conducting the FR and selecting the PMs for an allocation. Otherwise stated, the one allocation function allocates the VMs randomly while keeping the stacking behaviour by altering the FR function and selecting the PMs.

The key factor here is the Highest Fullness Ratio (HFR) function, at line 7 denoted as **getHighestFRPMs(v,P)**. It is different from the FR function that we explained in Section 5.2.1 because the FR aims to sort the PMs based on their fullness ratio compare to the unallocated VMs. However, the HFR, in addition to the FR function, aim to select the most two highest fullness ratio PMs only out of the available PMs. The HFR function's outcome is then stored in a list of elected PMs, denoted as **ElectedPMsList**. The details of the HFR step is explained in the following section, Section 5.3.3.

The next step of SRS is to select one of the elected PMs as a candidate for allocating the v, at line 8. It starts by randomly selecting the p among the elected PMs and assigning it as a candidate for the next step. The function

getRandomPM(ElectedPMsList) is responsible for selecting a random PM among the list of the elected PMs and store it as a candidate PM, denoted as *p*. 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 knows that the algorithm is following a stacking behaviour, it will be costly to obtain a malicious co-residency and know the exact PMs with target users as it will need to launch many VMs periodically to be able to achieve SCAs.

Afterwards, at line 9, the algorithm will check if the selected *p* will produce a malicious co-residency. This step is essential to trigger the **vmMigration(ElectedPMsList,P)** function, which will be explained in Section 5.4. The **getCORvmCheck(v,p.getVMslist())** function is 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 *v* to *p* is performed by the function **Assign(v,p)**, at line 10, and added to the one allocation list. The **Assign(v,p)** function will override any previous allocation commitment for the same VM. Meaning if the function is accessed again by the same VM and alternative PM selected, it will select this new assignment as final. The process of adding to the one allocation list will ensure a unique VM allocation resulted from the **Assign(v,p)** function.

In the next step, if there is a malicious co-residency on the selected PM, at line 14, 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, resulting in some downtime, even for the live migration. The process of VMs migration requires moving a VM current state while it is running. Furthermore, copying the VM state and restoring it in the destination PM requires a slight downtime at a particular stage. That results in an unwanted interruption to most cloud users, which SRS tries to avoid by only selecting to migrate the VMs 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. Then the algorithm will repeat the malicious co-residency check after the migration. Finally, at line 18, if performing the initial selection and VM migration leads to a failed assignment, then the algorithm will assign the v to any suitable PM. The one allocation function will continue until all the unallocated VMs find an assignment.

5.3.3 Highest Fullness Ratio (HFR)

This function allows only the PMs with the highest fullness ratio compared to the VMs demanded resources to be selected as elected PMs for the allocation. As such, in HFR, we only select the PMs representing the highest ratio, not all the PMs, specifically those with the two (HFR%) among the PMs. Unlike the FR that aims to sort the PMs according to their FR during the allocation.

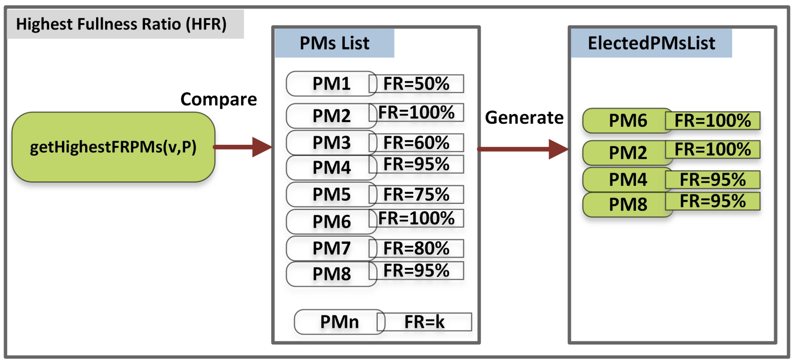


Figure 5.5: Computing the Highest Fullness Ratio (HFR) of an Arriving VM.

For example, in figure 5.5, the PM6, PM2, PM4 and PM8 are the elected PMs for two reasons. Firstly, they have the HFR among the available PMs. Secondly, they are the PMs that represent the two (HFR%) among the other PMs (FR%). The PM7 still has high FR but is not selected as the algorithm will only choose the two (HFR%). The reason for selecting the two (HFR%) PMs is to allow more elected PMs to be available. However, increasing this number higher than two could potentially lead to changing the stacking behaviour of the SRS algorithm.

The motivation for the HFR 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 allows more space for incoming VMs to allocate securely. The score of the HFR depends on the current situation of the available PM resources, the VM required resources and the time that VM arrives. Here we also calculate the HFR score based on Eq (5.1), explained in Section 5.2.1.

5.3.4 Infected PMs and Lowest Infected Allocations

This section will explain the methods of calculating the infected PMs and selecting the lowest infection allocation. The two functions are described in the SRS algorithm, at lines 31 and 38, respectively. Firstly, the infected PMs are denoted as **getInfectedPMsNumber(oneAllocation)**; this algorithm aims to calculate the number of infected PMs used in an allocation to compute malicious co-residency. Alternatively stated, it calculates the PMs with malicious co-residency compared to all the used PMs during the allocation. Secondly, the lowest infected allocations are denoted as **getLowestInfectedAllocation(allAllocationList)**; this algorithm compares the existing produced allocation and selects the one with the lowest co-residency to produce the most secure allocation in a given time limit.

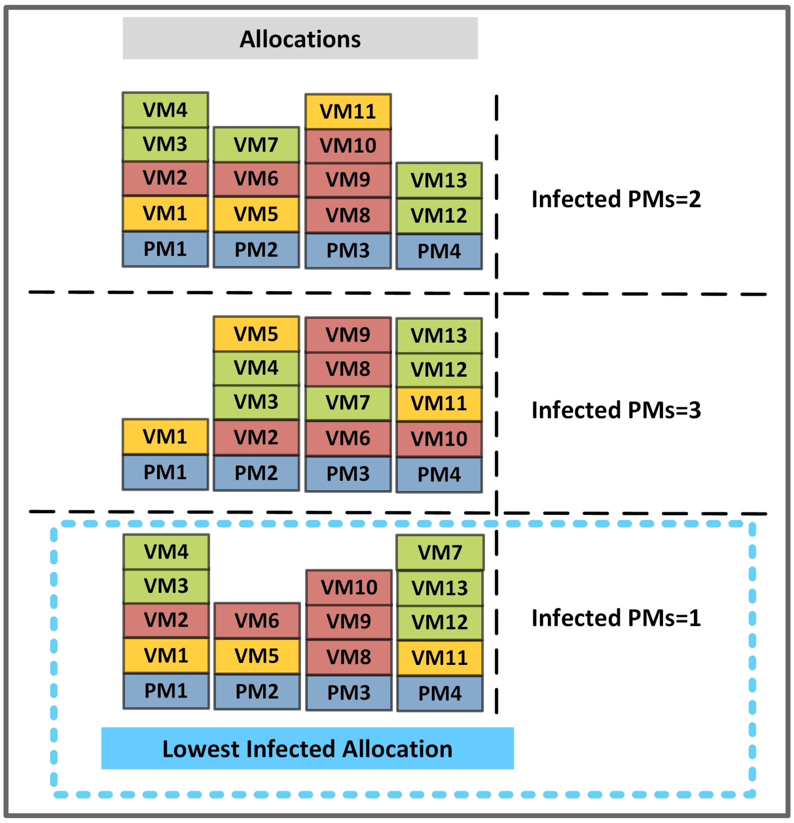


Figure 5.6: Selecting the Lowest Infected Allocation of all Allocations.

As shown in Figure 5.6, the infected PMs function will return an integer value indicating how many PMs, in an allocation, resulted with malicious co-residency. Each allocation produced in the SRS algorithm will go through this function to compare its value with other produced allocations. The value of the infected PMs will be different for each allocation as the SRS follow a random stacking behaviour, which means that a new allocation is produced even for the same set of configurations. In other words, due to the randomness in the SRS algorithm, a different allocation will be produced every time for the same set of resources.

Moreover, retrieving the lowest infected allocation depends on the result obtained from getting the infected PMs. Meaning that each produced allocation has a value of the infected PMs; thus, producing the lowest infected allocation depends on this value to identify the lowest infected allocations out of the produced allocations. As such, the SRS algorithm will select the allocation with the lowest infected allocations produced at the defined time limit or earlier that time if the lowest value reaches zero. Thus, the SRS aims to select the most secure allocation out of the produced allocations at this step.

5.3.5 Time Complexity Analysis for SRS Algorithm

The SRS algorithm depends on producing many allocations within a time limit or until a secure allocation is produced. We will first analyse the function responsible for producing the allocation, the *oneAllocation* function. Three main inputs affect the complexity in SRS: the VMs, denoted as N, the PMs, denoted as M, and the time limit, denoted as K.

The *oneAllocation* function is similar to the SS algorithm in the tries and time complexity analysis steps. However, the only difference is on the FR calculation step, $getHighestFRPMs$ function, where the sorted PMs based on their FR will be again searched for the highest two FR percentages, as explained in Section 5.3.3. Hence the big-O for the first try is $O(N * (2M + M * logM))$.

To avoid repetition, the big-O for the one allocation function is $O(N*(5M+N*M+2M*logM))$ considering each try is a nested loop. As we stated, the SRS producing many allocations within a time limit, we will denote to the time as K. As such, it will contribute to the time taken to complete the algorithm steps, which means the big-O will be $O(K*N*(5M+N*M+2M*logM))$.

5.4 VMs Migration Algorithm

This section describes the VMs migration algorithm utilised in the SS and SRS algorithms at lines 16 and 14, respectively. As stated earlier, one of the objectives of this work is to minimise the number of VMs migrations during the VMs allocation. The reason is that migrating the VMs from one PM to another, resulting in unwanted downtime, which causes the services running on the VMs to be interrupted. In brief, VM migration requires moving the VM current state while the VM is running. Furthermore, copying the VM state and restoring it in the destination PM requires a slight downtime at a particular stage. We described the detailed process of migrating a VM from one PM to another in Section 4.1.4.

Algorithm 5.4: VMs Migration

```
1  Function vmMigration(PMsList,P)
2      vmsToMigrateList ← $\emptyset$
3      migrationAllocationList ← $\emptyset$
4      for pm in PMsList do
5          vmsToMigrateList.add(pm.getMaliciousVms() $\cup$ pm.getNormalVms())
6      end
7      for v in vmsToMigrateList do
8          for p in P do
9              if $p \neq$ v.getPM() then
10                 if (getCORvmCheck(v,p.getVMslist()) $\neq$ true) then
11                     migrationAllocationList.add(AssignBased($v$, P))
12                 end
13             end
14         end
15     end
16     return migrationAllocationList
```

The objective of the proposed VMs migration algorithm, Algorithm 5.4, is to migrate the VMs securely in a way that must leave the system in a secure allocation. If the VMs migration resulted from existing malicious co-residency, the VMs migration will try to migrate the allocated VMs to avoid the malicious co-residency. The allocation algorithms, SS and SRS, aim to produce a secure VMs allocation; thus, the VMs migration must preserve the security state of the secure allocation during the migration or enhance it. Thus, the migration function attempts to maintain that secure status, whenever possible.

The migration function receives the list PMs, denoted as *PMsList*, to select their VMs for migration and the available PMs set, denoted as P. 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 SRS algorithm utilises the VMs migration function by sending the list of the elected PMs, as *PMsList*, to migrate their VMs. This list of PMs has a low number of VMs compared to all the available PMs because the elected PMs only select the PMs with the (HFR%). 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 (HFR%); thus, we have introduced further steps to control the number of VMs migrating per migration event.

The VMs migration function starts at line 4 by selecting all the malicious VMs and normal VMs from *PMsList* for possible migration. At this step, the VMs migration is not guaranteed to occur; there are more steps to consider for the migration to be performed. Thus, this step only selecting the VMs considers candidates for VMs migration. We only select the VMs classified as normal or malicious VMs for migration for several reasons; it will allow more space for the target VMs, by migrating the normal VMs to allow more space, thus reducing the chance of co-residency. Moreover, the normal VMs can be co-located with any VMs as long as the selected PM has enough space. Also, selecting the malicious VMs will increase the difficulty for the malicious VM user to achieve SCA. This step is an extra defence against SCA, as the malicious user will try to predict the allocation behaviour to be placed with the target VM; therefore, selecting the malicious for possible migration will reduce this chance. Hence, reducing the possibility of being allocated with a target VM. Moreover, the target VMs migration is not considered because they are classified as critical VMs before and during the allocation, as described in Section 4.1.6. Thus we wanted to avoid any service interruptions caused by the VMs migration for target VMs.

After this step, at line 11, the selected VMs will be allocated to different

PMs by triggering the **AssignBased(v,P)**. This function aims to allocate the selected VMs to new PMs based on the defined allocation algorithm. For instance, the **AssignBased(v,P)** will allocate the selected VMs based on SS steps if the SS algorithm triggers the migration function. Similarly, for the SRS algorithm and any algorithm triggering this function.

In the following sections, we will describe more algorithms for comparison purposes, and these algorithms are redesigned to utilise the migration function. Thus, the **AssignBased(v,P)** is adapting based on the defined allocation algorithm. As we stated earlier, selecting VMs for migration does not necessarily mean that this VM will be migrated. At this step, and depending on the defined algorithm constraints, the selected VMs might not find a suitable allocation for the migration. Therefore, it will be discarded from the migration and consider the next standing VM. For example, in SS and SRS algorithms, if the selected VM reduces the security state of the allocation to a worse situation, this VM will be discarded from the VMs migration. We will show more about the effect of this selection on the evaluation of the algorithms in Sections 5.6 and 5.7. Finally, if the selected VMs are suitable for migration, they will be allocated to specific PMs according to the same steps of one allocation function.

5.5 Alternative Allocation Behaviours

This section will describe alternative VMs allocation behaviours that is considered in this work for comparison purposes. As we stated, the proposed algorithms, SS and SRS, follow a stacking-based VMs allocation behaviour, which was described in Sections 5.2 and 5.3. In this part, we introduce more VMs allocation behaviours algorithms to compare our stacking-based algorithms 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 [13]. The second one is random behaviour, which aims to allocate the VMs randomly as long as the candidate PM is suitable. In [11], they presented a random-based algorithm called (CLR), aiming to allocate the VMs randomly. In this work, we will denote the Random (Rand) behaviour as Rand for simplicity. The third considered allocation behaviour is a combination of spreading and random behaviour. This behaviour algorithm, described in [46], depends on spreading the VMs of the same user if they exceeded three VMs on the same PM. Moreover, the eligible PMs selection is performed randomly if they have less than three VMs of the same user. The algorithm is called the Previously Selected Servers First (PSSF) algorithm.

The following sections will briefly describe how these algorithms allocate the VMs, i.e., describing their allocation behaviours. Moreover, we will show how we modify them to integrate them with the learning model presented in Section 4.1.6. In other words, how these algorithms have become security-aware of the classified VMs in the assumed cloud system of this thesis.

5.5.1 Spreading Behaviour

For simplicity, we will refer to the spreading behaviour algorithms as round-robin or RR for the rest of the thesis. As stated, this behaviour focuses on allocating the VMs by spreading them across the available PMs. Thus, each unallocated VM will be scheduled for allocation on the next suitable PMs. For example, as it is shown in Figure 5.7, the VMs arrival starts from VM1 until VMk, and VM1 will be allocated to PM1, assuming it has enough resources. The VM2 will be allocated to the next suitable PM, PM2, even if the first PM, PM1, has enough resources. The allocation algorithm will continue with the next VM with the same behaviour by allocating it to the next suitable PM until all the VMs are allocated.

Therefore, we can deduce that such behaviour's objective is to balance the VMs allocation loads over the available PMs. The load-balancing of resources leads to several desired outcomes, such as optimising the power consumption for energy efficiency of the utilised PMs. Alternatively, in some cases, to keep the threshold of utilising the computing resources to a certain level under defined SLAs. For example, the utilisation of a CPU on each PM should not exceed 95%. Moreover, this behaviour has been utilised for security purposes as described in [78].

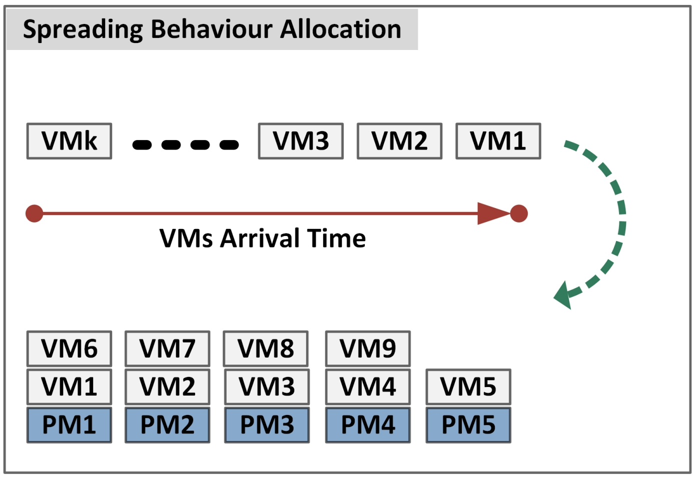


Figure 5.7: The Behaviour of Spreading Algorithms.

5.5.2 Random Behaviour

Random behaviour refers to the methods that randomly allocate the VMs as long as the candidate PMs have enough resources. As stated, we will refer to the random behaviour algorithms as random or Rand for the rest of the thesis.

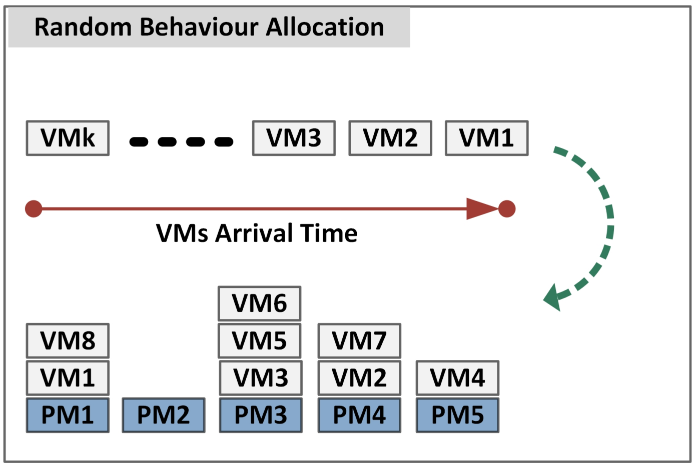


Figure 5.8: The Behaviour of Random Algorithms.

As shown in Figure 5.8, for the arrival VMs, starting from VM1, the random algorithm will attempt to allocate the VMs randomly to the available PMs. The random allocator has no specific constraints other than the selected PMs have enough resources. Moreover, as explained in [11], the random allocator will check if the selected PMs have enough resources and then allocate the VMs. Otherwise, it will randomly reselect the available PMs.

The objective of such an approach is to make it difficult for the malicious user to predict the allocation scheme. However, if the attacker attempts to launch large number of VMs at the same time, it could potentially lead to malicious co-residency. Therefore, this approach makes it is costly to achieve malicious co-residency with the target VMs.

5.5.3 Combining Spreading and Random Behaviour

The last considered behaviour focuses on combining the spreading and the random behaviours, as described in [46]. As stated earlier, the proposed algorithm, called PSSF, aims to spread the VMs belonging to the same user if they exceeded three VMs on the same PM. For instance, if a user has three VMs that already allocated on a PM, the fourth VM belonging to the same user will be allocated on a different PM. Moreover, the algorithm will randomly select the eligible PMs, then allocate the VMs to them if they have less than three VMs of the same user.

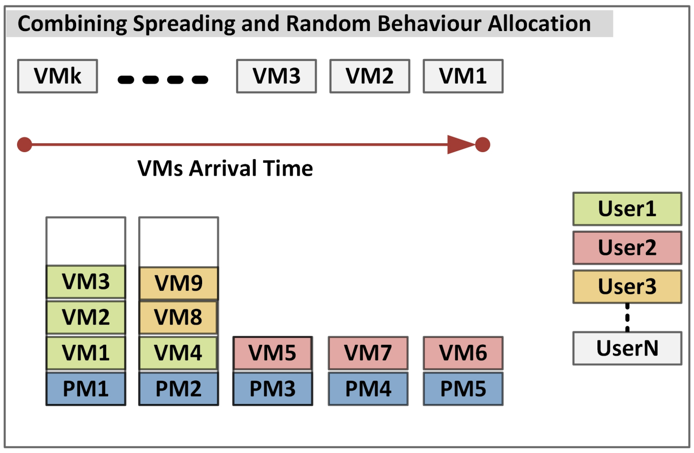


Figure 5.9: The Behaviour of Combining Spreading and Random Algorithm.

As shown in Figure 5.9, user 1 has four VMs, VM1 to VM4, and the first three VMs are allocated on the same PM, PM1. The outcome of this step is because the algorithm aims to allocate the VMs belonging to the same user on the same PM. However, the fourth VM, VM4, is allocated on different PMs, even though the PM1 has enough space. In this case, the algorithm spreads the VMs belonging to the same user if it exceeded three VMs on the same PM. Moreover, user 2 has three VMs, VM5 to VM7; however, these VMs are allocated separately because the first arrival VM, VM5, is allocated randomly among the available PMs. Then the second PMs could not be allocated on the same PM, PM3, as it was already fully utilised. Thus, it will randomly select another available PM to allocate the second VM. The same situation repeated for the third VM, which leads to spread the VMs belong to the same user while randomly allocating them on any available PM.

5.5.4 Security Factor

The three described algorithms Rand, RR and PSSF, do not check the type of the VMs during VMs allocation; thus, they are not aware of the classification of VMs described in the learning model in Section 4.1.6. The proposed algorithms SS and SRS are designed to be aware of the VMs classification based on their type. Specifically, the SS and SRS check the VMs type during the allocation using the co-residency detection function described in Section 5.2.2.

Therefore, we modify the three algorithms 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. For example, in the Random algorithm, if the upcoming VM has two PMs available to select from randomly, one of these PMs will lead to malicious co-residency; then, in this case, the algorithm will select the other PM. This change did not change the overall allocation behaviour of the random algorithm; however, it made it aware of the learning model classification as SS and SRS algorithms do. Moreover, for the RR algorithm, if the upcoming VM is allocated on the next available PM and this allocation will lead to a malicious co-residency, it will try the following available PM. If the following PM leads to secure VM allocation and has enough resources, it will be selected.

Eventually, all three algorithms will allocate the VMs according to their original behaviour if the allocator could not find a secure allocation. However, if the algorithms could not produce an allocation by following the secure approach, it will follow its original approach by allocating the VMs regardless of the malicious co-residency.

5.5.5 VMs Migration Integration

The VMs migration, described in Section 5.4, integrates with all the compared algorithms of this work, including SS, SRS, Rand, RR and PSSF algorithms. The objective of this integration is to migrate the VMs securely in a way that must leave the system in a secure allocation. We described how the VMs migration function integrated with SS and SRS earlier, and it is the same approach for the other algorithms. The VMs migration will be triggered if the algorithm could not obtain a secure allocation for the upcoming VM.

For example, in Rand algorithm starts by allocating the VMs randomly to the available PMs. Then, under a particular situation, the options for obtaining a secure allocation decrease, leading to triggering the VMs migration. Otherwise stated, the VMs migration for Rand algorithm will be triggered if the VMs allocation to available PMs leads to a malicious co-residency. Similarly, the RR and PSSF follow the same approach of VMs migration while keeping the behaviour the same of all the compared algorithms.

5.6 Evaluation of Non-Heterogeneous VMs Allocation

This section presents an evaluation of the proposed VMs allocation algorithms under different scenarios and structures where the resources of the VMs are not heterogeneous. Hence, the VMs required resources are assumed to be the same for this part of the evaluation. However, in Section 5.7, we will present another evaluation where the VMs required resources are heterogeneous.

The objective of this section is to evaluate the behaviours of VMs allocation algorithms, of non-heterogeneous VMs, under different situations and conditions. In other words, examine the occurrence of the malicious co-residency for the proposed algorithms under several VMs structures, PMs structures and VMs arrival times. Thus, this evaluation studies the factors that potentially affect the security of the VMs allocation algorithms and the overall malicious co-residency percentage. These factors include; the VMs arrival time, the effect of the heterogeneity level of the PMs, the effect of VMs structure, and the effect of the VMs migration.

The VMs arrival time referred to when a single VM arrived at a specific time depending on the VM type. For example, study the effect where the target VMs arrived firstly before the arrival of malicious VMs, or vice versa. Further, the heterogeneity level of the PMs refers to the recourse structure of the PMs, how they are different from each other. We consider three PMs structure, high, medium and low, and each structure defines the level of difference between the available PMs. Thus, we aim to examine the effect of the available PMs resources structure on the secure VMs allocation for the proposed algorithms.

Moreover, the evaluation will examine the effect of the VMs structure on obtaining a secure VMs allocation. The VMs structure refers to the number of VMs according to their types. For example, in an allocation, the study of increasing the number of targets VMs over the normal or malicious VMs. Alternatively, if the malicious utilised many VMs, how likely that the malicious co-residency will be achieved. In addition, we aim to examine the effect of VMs migration on the proposed algorithms based on its design. In other words, the VMs migration aims to reduce the effect of malicious co-residency, and therefore, this part will examine this effect for each algorithm.

Additionally, this section will study the effect of the mentioned factors on the proposed algorithms, SS and SRS, and compare them with the described algorithms in Section 5.5. Therefore, we will compare SS and SRS algorithms with three algorithms that follow a different allocation behaviour: Rand, RR, and PSSF. As stated, the SS and SRS follow a stacking-based behaviour while the Rand follows a random-based behaviour, RR follows a spreading-based behaviour and PSSF combines the random and spreading.

Furthermore, we utilise a powerful simulation tool to simulate different aspects related to the CCEs architecture called CloudSim. CloudSim is an open-source cloud simulation environment that builds based on cloud system workloads that aim to simulate the provisioning of cloud computing systems. Therefore, it contributes to providing a valuable estimate of the expected outcome of certain situations and an initial understanding of the behaviour of the allocation process. In general, cloud simulators are helpful to provide a solution and projection of the real-world scenarios; thus, we utilise CloudSim to examine our proposed algorithms [18]. Refer to **Appendix** B for more information about CloudSim, its structure and allocation process. There are alternative cloud simulator tools that can be utilised for simulating cloud resources allocation, for example, CloudSched, GroudSim and GreenCloud. Each tool is suitable for specific functionality, for instance, SLA support, energy-based allocation model, or cost model. Other factors could influence the selection of the simulator, such as open-source tools or graphic user interface support [30]. In this work, we have selected CloudSim as it provides an open-source environment and serves our needed functionality.

Lastly, this work examines the effect of the secure VMs allocation, VMs migration, and PMs usage during the allocation for the presented algorithms. The details of the mentioned factors and simulation structures will be described in detail in the following sections.

5.6.1 Experimental Setup

The section will present detailed information about the simulation environment utilised in this work. Furthermore, it will describe the structure of the PMs resources considered during the allocation process. Moreover, the VMs structure including the VMs arrival times, the structure of VMs resources and the structure of VMs type. Lastly, it will describe the experiment process to illustrate how the experiments were conducted under different situations.

Simulation Environment

As stated, this work utilises CloudSim, a cloud computing simulation environment, to examine the proposed VMs allocation algorithms and compare them. CloudSim provided a valuable estimate of the expected outcome of certain situations and an initial understanding of the behaviour of the VMs allocation algorithms.

Briefly, the process of CloudSim starts by registering the resources information of the data centre. Including the resources of the PMs, VMs and other components such as the data centre architecture. The registered information, including the required and available resources, will be collected to identify the computing resources. In other words, allocate the required recourse to the available recourse under the defined allocation algorithm. Afterwards, the available resources that will host the VMs or task resources depend on different levels and factors. More information about CloudSim process are described in **Appendix** B.

VMs and PMs Number

As shown in Table 5.1, 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 unlimited available resources; then, the resources get limited until it reaches 120 VMs, try number 6. The goal is to examine the effect where the resources is spacious and when it gets limited.

Table 5.1: VMs and PMs number

Tries No.	VMs No.	PMs No.
1	20	24
2	40	24
3	60	24
4	80	24
5	100	24
6	120	24

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.

The CPU is space-shared, meaning that each CPU can only accommodate one vCPU at each time. There are two types of scheduling policy, time-shared and spaced-shared; time-shared means that the VMs sharing the same PM can be processed on the same computing hardware, for example, on the same CPU core, but in different time stamps. These VMs can be scheduled based on the defined requirement for each VMs and the scheduling policy. However, the space-shared allocate a specific space for each VM, and it will be allocated for this VM all the processing time. More information about the scheduling and allocation policy is explained in **Appendix** B.

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, while the. The same definition applies to *T(t)* and *N(t)* for target VM and normal VM, respectively.

Table 5.2: VMs Arrival Time Types

Tries No.	VMs Order	Description
1	GMTN	G(M), G(T), G(N)
2	GNMT	G(N), G(M), G(T)
3	GTNM	G(T), G(N), G(M)
4	SNMT	S(N), S(M), S(T)
5	Mixed MTN	S(MTN), G(M), S(MTN), G(T), S(MTN), G(N), S(MTN)
6	Mixed NMT	S(NMT), G(N), S(NMT), G(M), S(NMT), G(T), S(NMT)
7	Mixed TNM	S(TNM), G(T), S(TNM), G(N), S(TNM), G(M), S(TNM)

As shown in Table 5.2, we study most of the possibilities of VMs arrival time based on each type of VMs. For instance, in try 1 in the table, we examine the effect of when a group of malicious VMs arrives, then a group of targets VMs arrives, then a group of normal VMs arrives last, denoted as GMTN. Moreover, in try 2, 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. The same process applies to try number 3, denoted as GTNM.

Furthermore, in try 4, the VMs will arrive a single instead of a group, meaning one normal VM arrives, followed by malicious, followed by target, denoted as SNMT. In this case, we did not consider the other possibilities of single arrivals, like what we did in the group arrivals, because they showed the same allocation behaviour during the experiments. Thus, to avoid duplication, we ignore the other possibilities.

Lastly, in the tries from 5 to 7, the three arrival times are Mixed MTN, Mixed NMT and Mixed TNM. To demonstrate, in the MTN arrival type, the VMs in any experiment is divided into seven groups; each group will arrive in the same sequence described in the table. For example, in Mixed MTN, out of seven groups, the first group is S(MTM), where the S refer to a Single VM arrival. Here, the malicious, target and normal VM will arrive in a single alternative sequence. Precisely, single (M)- then single (T)- then single(N), then repeats the process until this group of VMs arrived. Then the second group, G(M), means that a group of malicious VMs will arrive second. Then the third group will repeat the same order as the first group. After that, the fourth group G(T), means that a group of target VMs will arrive fourth. Then the rest of the seven groups will arrive until the last VM arrives. The same concept will apply to the other two types of VM orders, Mixed NMT and Mixed TNM. The size of each group, the seven groups of each 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

We used the same classification described in Section 4.4.2, Table 4.1, which considers seven possible situations where each VMs type number might reach for each experiment. To summarize, each VMs type number will be examined for its secure VMs allocation level and its performance under the defined configuration. For example, if we consider 20 VMs, this VMs type number will be structured seven times, as described in Table 4.1, 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. The 20 VMs will be divided into three parts, as per the percentage defined in the table.

PMs Heterogeneity levels

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.

Experiments Methodology

We investigate every combination of VMs with each number of PMs and their different structures of resources. As such, we consider each type of PMs heterogeneity under each VMs arrival time and the number of each VMs type in each experiment.

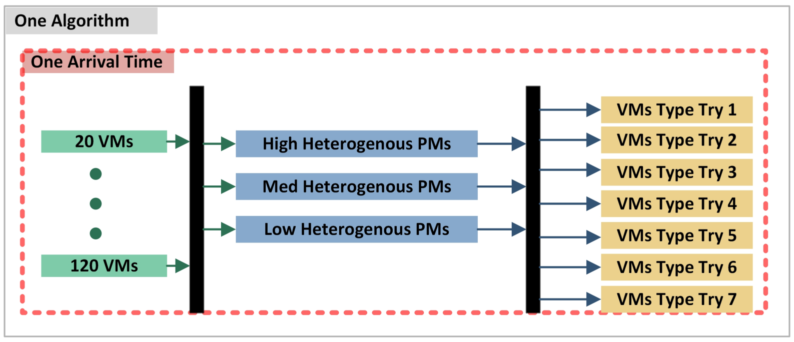


Figure 5.10: Experiments Methodology for each Algorithm under each Arrival Time.

For example, in Figure 5.10, in the situation where we have 20 VMs, we first examine the 20 VMs on high heterogeneous PMs. Moreover, for each high heterogeneous PMs, we examine each possible VMs structure type. The VMs type structure is the seven possible situations that define how many of these VMs are malicious VM, target VMs, or normal VMs, as described in Section 5.6.1. Then, for the same number of VMs, 20 VMs, we repeated the simulations on the low and medium heterogeneous PMs. Then move on to increase the number of VMs to 40 VMs, and repeat the previous steps until we reach the 120 VMs. These experiments were conducted only for one arrival time for one algorithm. For instance, these simulation steps are performed for the SRS algorithm and GMTN arrival time, examining 252 situations for each algorithm under one arrival time. We have defined seven arrivals time; therefore, we examine 3528 situations for each algorithm, including the situations where the migrations are enabled or disabled. Therefore, the total experiments conducted for this part of the evaluation, including all the mentioned algorithms, are 17640.

Moreover, we utilised a real workload while conducting the experiments to mimic real cloud computing scenarios as much as possible. The used workload is published by the Karlsruhe Institute of Technology ForHLR II System [67]. This thesis only examines part of the experiment using this workload to examine the preliminary performance effect shown in Section 5.7.5. For future work, we aim to utilise this workload to specify the type of applications running for each VM and their effect on the performance of the allocation algorithms. Moreover, examine the effect of the dependency of VMs application on the allocation rules, for instance, group of dependent VMs running platform of web application and databases required to be allocated in adjacent PMs.

5.6.2 Results of Malicious Co-residency Respect to VMs Arrival Time

As stated in Section 4.4.3, Eq.(4.15), the M_{pms} is the percentage of the infected PMs out of the used one. The infected PMs means the PM that has a malicious co-residency between the target and malicious VMs and shares the same PM. This section will compare the malicious co-residency concerning VMs arrival time for each algorithm.

Figures Explanation

As an example, in Figure 5.11a, we compare five algorithms representing different VMs allocation behaviours, as explained earlier. These algorithms are SS, SRS which are the ones we proposed in our work following stacking-based behaviour. The others are PSSF, Random (denoted as Rand) and Rand Robin (denoted as RR).

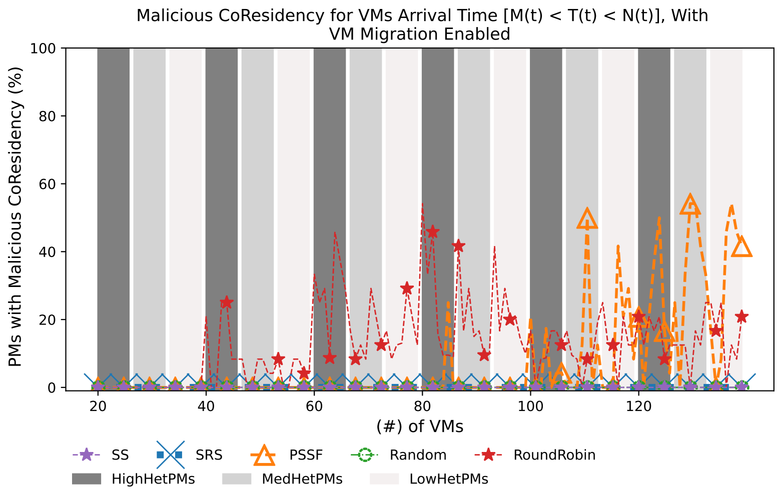
The notation in the title *M(t)* $<$ *T(t)* $<$ *N(t)* means that the malicious VMs arrives and is allocated in a time before the arrival of target and normal VMs.

The same definition applies to the other arrival times. The x-axis represents the number of VMs starting from 20 up to 120, where the y-axis represents the PMs with malicious co-residency percentage, which we will refer to as M_{pms}. The vertical shading colour represents the PMs heterogeneity level, high to low.

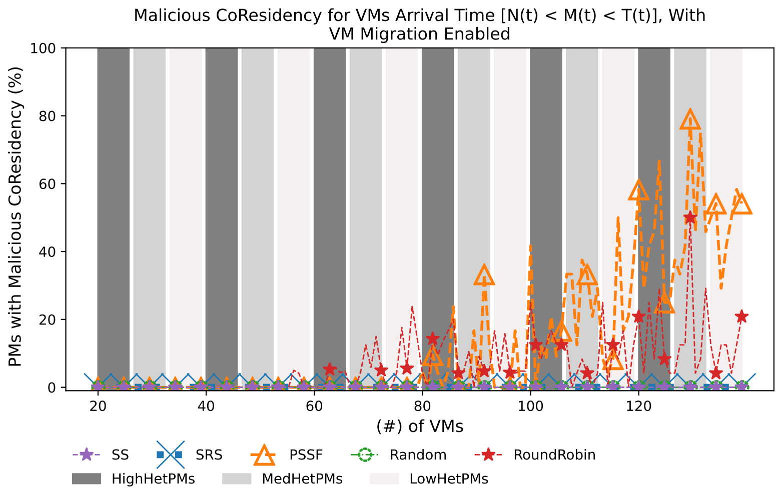
Overall Outcome For Group VMs Arrival

Generally, in Figure 5.11, the outcomes of the M_{pms} , for all the compared algorithms, showing a resembling under the group VMs arrival. The PSSF and RR algorithms show the worst cases due to their spreading behaviour, while SS, SRS and Rand are the best in the three situations. The case where the malicious VMs arrive lastly is considered the worst-case scenario, especially when the available resources are limited for PSSF, Rand and RR algorithms.

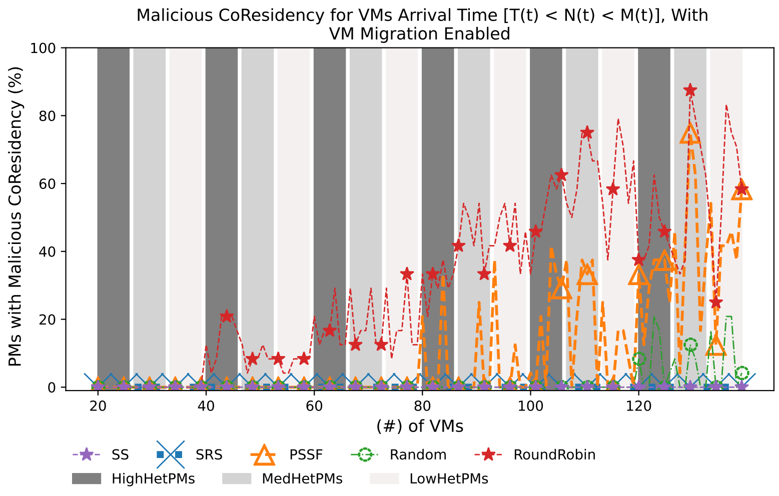
Specificity, in Figure 5.11a, the PSSF and RR are the worst for the M_{pms} due to the spreading behaviour and because the malicious VMs arrived first, which helped to spread the VMs. Nevertheless, PSSF only shows weakness when the number of VMs increases and the available recourse on the PMs start limiting. Moreover, the RR suffers more from increasing M_{pms} when the PMs are **HighHetPMs**, compared to other PMs structures. This increase is because the PMs available for allocation are filled more quickly than other PM structures, which leaves fewer options. The SS, SRS algorithms show the least M_{pms} due to their stacking behaviour, and the malicious and target VMs already found an allocation where the available resources still have more options. In other words, the arrival of normal VMs at the last helped the algorithms, following stacking behaviour, to produce a secure allocation were the targets and malicious allocated separately due to the availability of allocation options. The Rand algorithm also shows similar outcomes to the SS and SRS for the same reason, because the algorithm will easily produce a secure allocation, even if the resources are limited, as the target and malicious VMs already allocated securely.



(a)



(b)



(c)

Figure 5.11: PMs with Malicious Co-residency under GMTN, GNMT and GTNM Arrival Times.

Moreover, in Figure 5.11b, 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. 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, its constraint of keeping 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.

The RR algorithm is still showing a high M_{pms}, but it is lower than the other arrival times, and the cases where the M_{pms} happened are also lower. This outcome happens because when the normal VMs arrived, it first spread its VMs across the available PMs and since the target and the malicious VMs can be allocated with the normal VMs. It becomes easier for the algorithm to obtain a secure allocation by allocating the upcoming VMs, target or malicious, with normal VMs whenever possible. The SS and SRS show the least M_{pms} due to their stacking behaviour which depends on allocating the VMs on the utilised PMs rather than spreading them across the available PMs.

Furthermore, in Figure 5.11c, this is the worst-case scenario for the RR algorithm because the target VMs arrive and then spread their VMs; then, when the malicious VMs arrive later, the M_{pms} significantly increased with limited available resources. What makes it worse for this case is the arrival of normal VMs in between, as it consumes most of the available resources, leading to this higher M_{pms}. For this case, a large number of normal VMs could potentially lead to higher M_{pms}. A notable case for the Rand, as it performed worse here than other situations due to the arrival of malicious VMs at the end, and with the limiting of the resources, the M_{pms} increases. In this case, the options for a secure allocation become narrow as the target and normal VMs already consume most of the resources upon arrival. The PSSF algorithm shows a similar outcome as the other two VMs arrival where the higher M_{pms} starts when the resources are limiting. The SS and SRS show the least M_{pms} among others for the same reasons mentioned earlier.

Overall, the SS and SRS algorithms perform the best due to their stacking-based behaviour, which depends on allocating the VMs by reducing the utilised PMs. The Rand algorithm performs worse when the malicious VMs arrive last and the normal VMs consume the available resources and target VMs allocated. The same case will occur with target VMs when they arrive last while the normal VMs allocated, leading to the same outcome. The other cases of the VMs arrival will lead to a secure allocation for the Rand algorithm. For the PSSF algorithm, the best case, among the three cases, is when the malicious and target VMs allocated firstly, meaning when there are more options for secure allocation and more resources available. The RR shows the lowest M_{pms} when the normal VMs arrive first and spread their VMs across the available PMs, which gives the algorithm easier options to allocate the VMs securely either with the normal VMs or with their own VMs type.

Overall Outcome For Single VMs Arrival

In this part, we will show the result of M_{pms}, where the VMs arrives separately according to their type. For example, in this case, we consider that a normal VM arrives followed immediately by a malicious VM followed immediately by a target VM. We only consider this situation of single arrivals, as the other possible arrivals showed the same behaviour during the experiments, like the way we did in the group arrivals. Thus to avoid a duplication of the results, we only show this type of single VM arrival.

In Figure 5.12, the most important remark that we can obtain is that all the algorithms are performing remarkably better than the group arrivals. In other words, 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.

Moreover, in specific cases, the RR algorithm is the worst among the other algorithms, even when compares to the PSSF algorithm, as it shows more spikes of M_{pms} when the available PMs are not limited. This outcome is the same that we mentioned earlier, which is the spreading of the VMs. Although the PSSF follows partially spreading behaviour, it showed fewer M_{pms} spikes for this case of single VMs arrival. Thus it makes it easier for the allocator to obtain a secure allocation. The SS, SRS and Rand algorithms are the best cases as they showed the lowest M_{pms} among other algorithms.

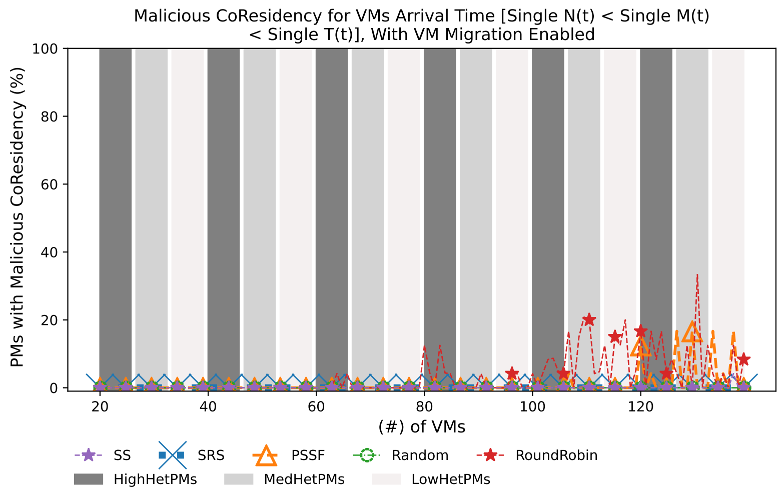


Figure 5.12: PMs with Malicious Co-residency under SNMT Arrival Time.

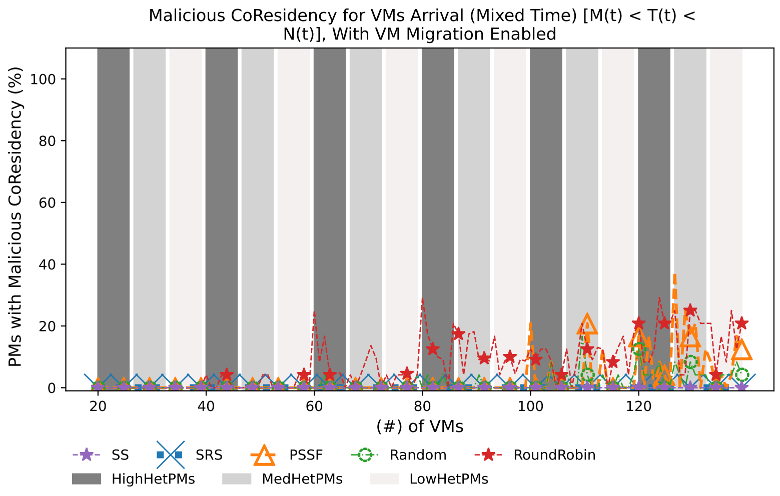
Overall Outcome For Mixed VMs Arrival

Generally, in Figure 5.13, the outcomes of the M_{pms} , for all the compared algorithms, showing a similarity under the mixed VMs arrival. It is better than the group VMs arrival as this type mixed the group with the single VMs arrivals, as described in Table 5.2. Thus the single VMs arrival influences the positive impact of obtaining more secure allocations for all the algorithms.

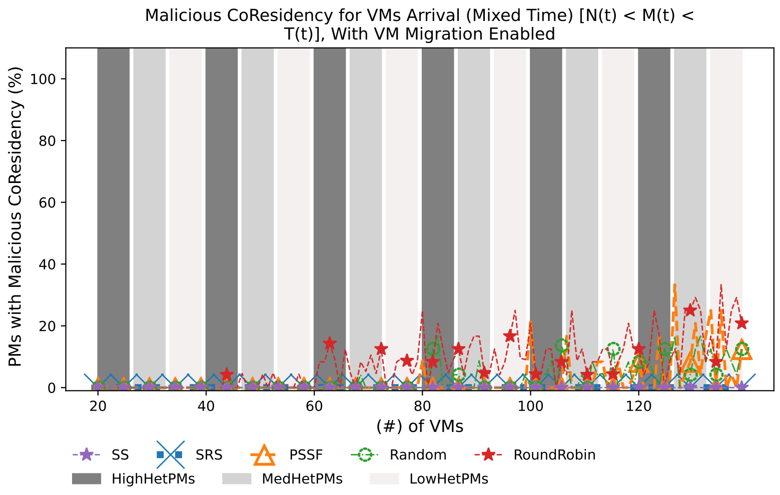
Specifically, in Figures 5.13a and 5.13b, the outcome is almost identical under these two arrival time configurations. These arrivals look similar because the normal VMs allocated lastly and firstly, in Figures 5.13a and 5.13b, respectively. Furthermore, the effect of single VMs arrival structure, that included in this mixed VMs arrival. To clarify, when the normal VMs arrives last, in Figure 5.13a, it did not cause any issues for the any of the algorithms, to obtain a secure allocation, as the normal VMs can be allocated on any PM as long as it follows the algorithm behaviour and the selected PM is suitable. The only issue that causes the spikes of M_{pms} is the mixed VMs arrivals structured, which yields having single VMs arrival between each group. Thus, the spike of M_{pms} will occur more often for this reason, especially when the resources are limited. Regarding the similarity with the VMs arrivals in Figure 5.13b, the same behaviour is repeated but opposite. 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, in Figure 5.13a. Ideally, under a different arrangement of the mixed arrivals, the RR, Rand and PSSF could obtain more secure allocations than those illustrated in these two Figures.

Moreover, We explained in the previous section that the single VMs arrival showed similar outcomes of M_{pms} for all the possible VMs arrival. Thus, we only showed one single VMs arrival case, called SNAT, to avoid duplicating the results. This positive effect of single VMs arrival is clear on these VMs arrival, including the one in Figure 5.13c, especially for the spreading-based algorithms. Because the overall M_{pms} drop significantly for RR, and PSSF algorithms compare to the group VMs arrivals for these algorithms.

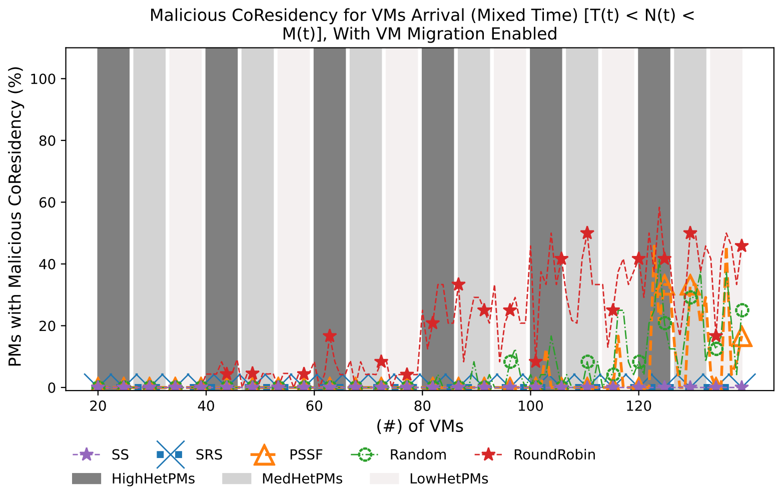
Additionally, in Figure 5.13c, the outcome of the M_{pms} is the worse among the mixed VMs arrivals. However, it is still better when compared with the group VMs arrivals. This worst outcome is due to two reasons; the arrival of malicious VMs last and the allocated normal VMs that consumed most of the available resources before the malicious VMs arrives. When the malicious VMs arrive, at last, the options for the available PMs are fewer as most of the resources are already allocated by the target and normal VMs. Hence, a higher M_{pms} occurring especially for the spreading-based behaviour algorithms.



(a)



(b)



(c)

Figure 5.13: PMs with Malicious Co-residency under Mixed MTN, NMT and TNM Arrival Times.

A notable case is occurring here related to the VMs migration that affects the overall M_{pms}. In this case, Figure 5.13c, the M_{pms} is higher than the case in Figure 5.13b, yet they share the same case of the arrival of either target or malicious VMs at last and the resources already consumed by normal VMs. Otherwise stated, in the first case in Figure 5.13b, the target VMs arrives last when the normal VMs consumed most of the resources and have fewer options for secure allocation. In the second case, in Figure 5.13c, which we explained, the malicious VMs arrive last when the normal VMs consumed most of the resources, with fewer options left. Nevertheless, the M_{pms} for the first case is lower compare to the second case. This outcome is because the VMs migration algorithm integrated with all the compared algorithms, as explained in Section 5.4. Briefly, this algorithm will only migrate the malicious VMs and normal VMs.

Thus, for the first case, when the target VMs arrives, many VMs are available for migration, either malicious or normal VMs, which leads to re-arrange the current allocations to produce a secure allocation for the upcoming VMs. However, for the second case, when the malicious VMs arrive, only the normal VMs were available for migration, leading to fewer options to produce secure allocations.

Overall, the RR and PSSF continue to have the worst cases of M_{pms} due to their spreading behaviours. Unlike the RR, the PSSF algorithm performing better when the available resources are not limited. The Rand algorithm outcome is worse in these mixed arrivals compared to the group and single VMs arrivals. The stacking-based algorithms, SS and SRS, showed the best outcome among the compared algorithms. The SS and SRS depends on allocating the VMs on the utilised PMs rather than spreading the VMs across the PMs, or randomly allocating them. Thus, this behaviour offer many options for the upcoming VMs to allocate securely under different arrivals structures.

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

This section will have a closer look at the M_{pms} concerning the VMs type number. The goal of this evaluation is to examine the effect of VMs type numbers on the M_{pms}. In other words, is the increase or decrease for any VMs, according to their type classification, will impact the M_{pms}. Similarly to what we describe in the previous section, Section 5.6.2, the M_{pms} is the percentage of the infected PMs out of the used ones. In addition, we will examine the effect of PMs heterogeneity level on the overall M_{pms} for the algorithms. Moreover, in this section, we only will show the results when the resources are limited, which means when the number of VMs equal 120 VMs.

Figures Explanation

As an example, in Figure 5.14a, similar to the previous section, we compare five algorithms representing different VMs allocation behaviours. The difference from the previous Figures is that we have three axes, one x-axis and two y-axes. The x-axis represents the number of VMs when they are 120 VMs, and the number of PMs 24, making the resources very limiting and challenging for the algorithms. The first y-axis on the left of the Figure represents the number of VMs according to their type. We have three colours for the VMs type; green for the target VMs, red for the malicious VMs and yellow for the normal VMs. The sum of these three colours always will be 120 VMs, as we only examine the 120 VMs case. The second y-axis on the right of the Figure represents the M_{pms} we explained in the previous section. The horizontal black lines represent the PMs heterogeneity levels of this part of the experiments. We have three PMs heterogeneous levels, which are high, medium and low heterogeneous PMs.

Malicious Co-residency for Group VMs Arrival under Limited Resources

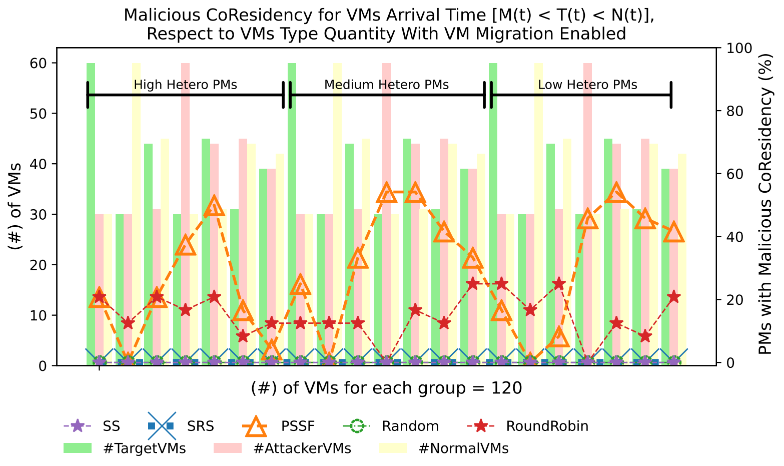
Generally, in Figure 5.14, the highest number of either target VMs or attacker VMs per allocation leads to higher M_{pms} for most algorithms. In other words, for a group of VMs arriving at a specific time, if the majority of those VMs are either malicious or target VMs, in some cases both are high, then the chance of getting malicious co-residency increases. Moreover, the effect of PMs heterogeneity is clear, as the high heterogeneous PMs structure will often lead to a lower M_{pms} and number of malicious co-residency occurrence when compare it with either medium or low PMs structure. Alternatively, the number of cases of where it leads to malicious co-residency is often lower in a high heterogeneous PMs structure than the medium or low heterogeneous PMs structure.

Specifically, 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.

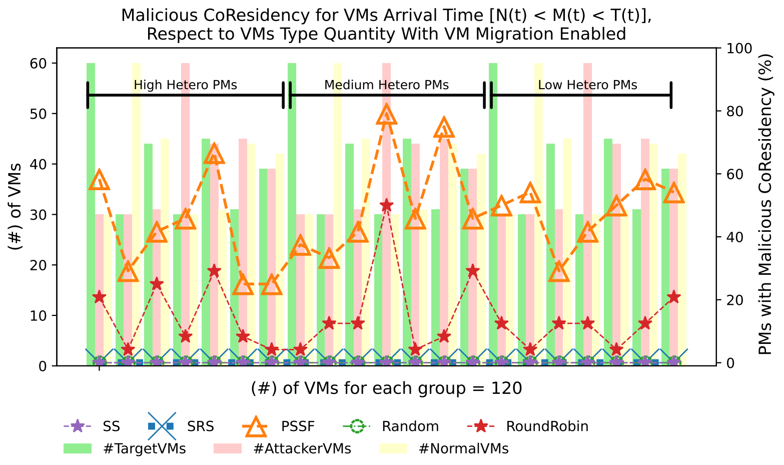
Furthermore, since PSSF spreading the VMs of the same user, and if the user is a malicious one, then the chance of malicious co-residency occurring is very high for such allocation behaviour. The same applies when many VMs belong to a target user or users arrive at the same time with a considerably high number of VMs.

In addition, when comparing the effect of PMs heterogeneity level, the high heterogeneous PMs structure often leads to a better result of M_{pms} than the other PMs structure for PSSF. The diversity of available resources in high heterogeneous PMs structure often leads to group the VMs with the same classification to the same PM. As such, allocating a group of VMs of the same classification type, such as target VMs, becomes easier as the available options have a high diversity of resources.

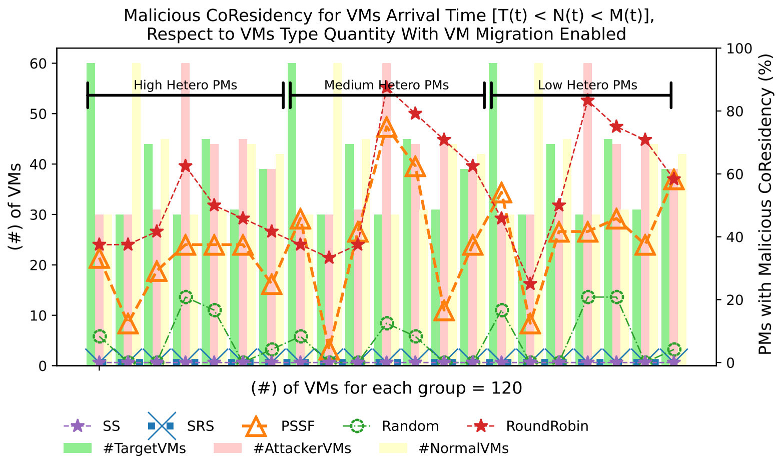
Moreover, the RR and Rand algorithms suffer from high M_{pms} due to the spike of malicious and target VMs. For RR, spreading the VMs is negatively impacting the M_{pms} as it is considered among the worst of compared algorithms. The reason for the high M_{pms} is the same as we described in the PSSF algorithm, as they share the spreading behaviour of allocating the VMs. In the Rand algorithm, specifically in Figure 5.14c, there is a clear effect of the high number of either target or malicious VMs on getting a high M_{pms}. The cases where the M_{pms} occurring in the Rand algorithm is only when there is an increase in the number of VMs causing the malicious co-residency, target and malicious VMs. Moreover, the high number of normal VMs often leads to producing a good result of M_{pms}, as the normal VMs a neutral to all VMs.



(a)



(b)



(c)

Figure 5.14: Malicious Co-residency under GMTN, GNMT and GTNM Arrival Times, When Available Resources Limited.

Overall, the SS and SRS algorithms are the best among the compared algorithms due to allocating the VMs in a stacking-based manner. This result shows that under different PMs and VMs structures, the stacking-based behaviour algorithms will produce secure allocations than spreading or random behaviours algorithms. 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}.

Malicious Co-residency for Single VMs Arrival under Limited Resources

Generally, in Figure 5.15, the single VMs arrivals lead to better results comparing to the group or mixed VMs arrivals. The effect of the VMs number, according to their type, is similar to what we explain in the previous section, in 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, 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.

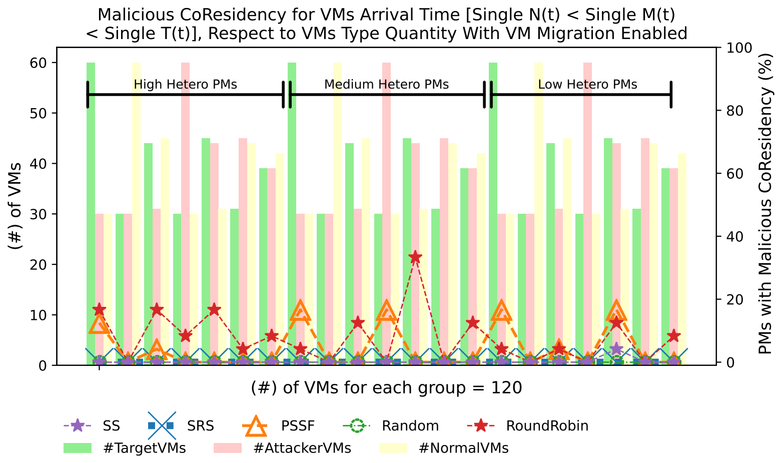


Figure 5.15: Malicious Co-residency under SNMT Arrival Time, When Available Resources Limited.

Malicious Co-residency for Mixed VMs Arrival under Limited Resources

To avoid duplication of the results, we will only show the case of mixed TNM VMs arrival, as shown in Figure 5.16. This case of mixed VMs arrival represents the worst-case scenario for the RR, PSSF and Rand algorithms among the other mixed VMs arrivals.

In most cases, the high number of target and malicious VMs produces allocations with high M_{pms}, as the allocator faces challenging situations in obtaining a secure allocation. In PSSF and Rand algorithms, there is 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, but the M_{pms} did not fall when the number of these two VMs types decreases. It continues to produce a malicious co-residency for all situations. There is a slight impact on the number of normal VMs spikes, which decreases the percentage of M_{pms}, but it is still occurring. The SS and SRS continue to produce the best outcome of the compared algorithms over the examined situations.

Furthermore, PMs heterogeneous structure's effect did not seem to have that great difference between the three types because all the three PMs structures show similar behaviour of malicious co-residency occurring, either in the number of spikes or in the percentages of the M_{pms}. In other words, the number of cases where the malicious VMs produced under each PMs structure is mostly similar.

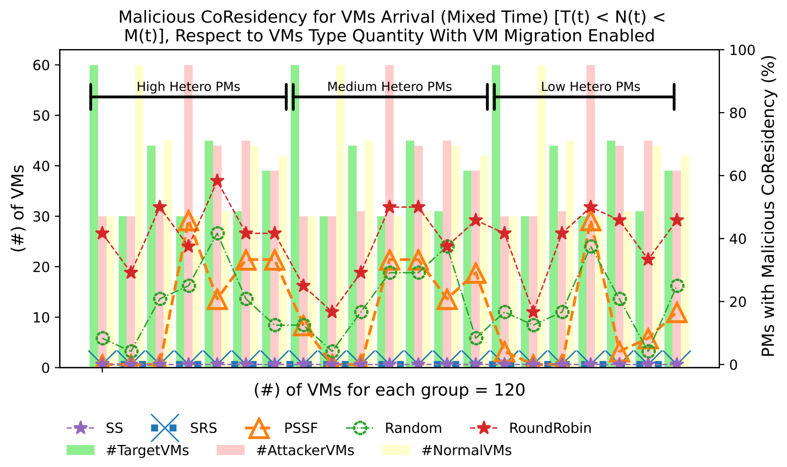


Figure 5.16: Malicious Co-residency under Mixed TNM Arrival Time, When Available Resources Limited.

5.6.4 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:

$$Mig_{vms} = \frac{S_{vms}}{T_{vms}} \tag{5.2}$$

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. The percentage of VMs migrations (Mig_{vms}) is an indication of the processing needed, by any algorithm, to produce a secure allocation.

The results presented in the previous sections, Sections 5.6.2 and 5.6.3, showed the percentage of malicious co-residency for each algorithm under different arrivals times and VMs structure. This part will show the effect VMs migration has on obtaining the secure VMs allocation of the compared algorithms. As stated, the VMs migration indicates the processing needed, by any of the algorithms, to produce a secure allocation. Otherwise stated, how many VMs migration is needed by any algorithm to produce a secure VMs allocations under different scenarios. Thus, this section will link the relationship between the VMs migration with M_{pms} under the different configurations.

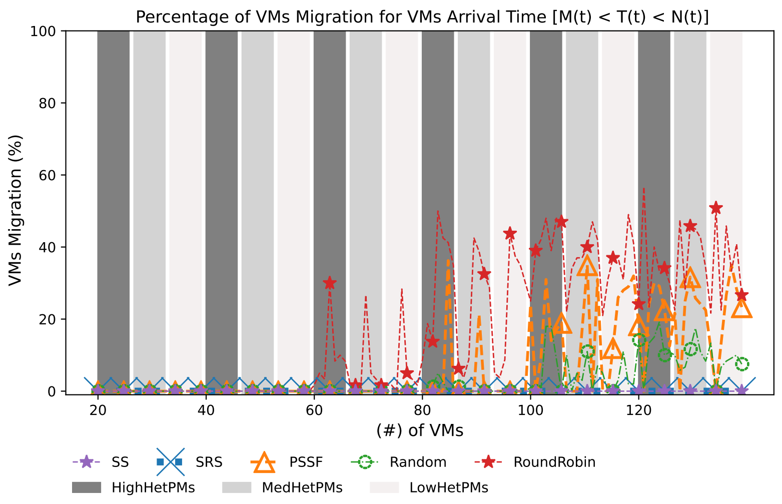
Figures Explanation

As an example, in Figure 5.17a, we compare five algorithms representing different VMs allocation behaviours to examine the impact on the VMs migration for these algorithms. The x-axis represents the number of VMs starting from 20 up to 120, where the y-axis represents the VMs migration percentage, which we will refer to as Mig_{vms}. While the vertical shading colour represents the PMs heterogeneity level, high to low.

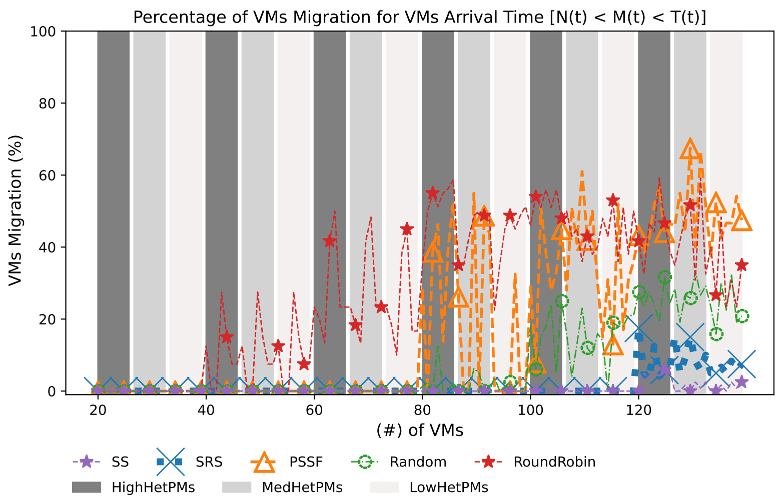
VMs Migrations for Group VMs Arrival

In general, as shown in Figure 5.17, the spreading allocation behaviours algorithms, RR and PSSF, are the worst in Mig_{vms}, especially when the resources are limited. While the random behaviour algorithm, Rand algorithm, have a moderated percentage of VMs migration considering benefits produced by these migrations, which is a lower chance of malicious co-residency, as described in Figure 5.11. 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.

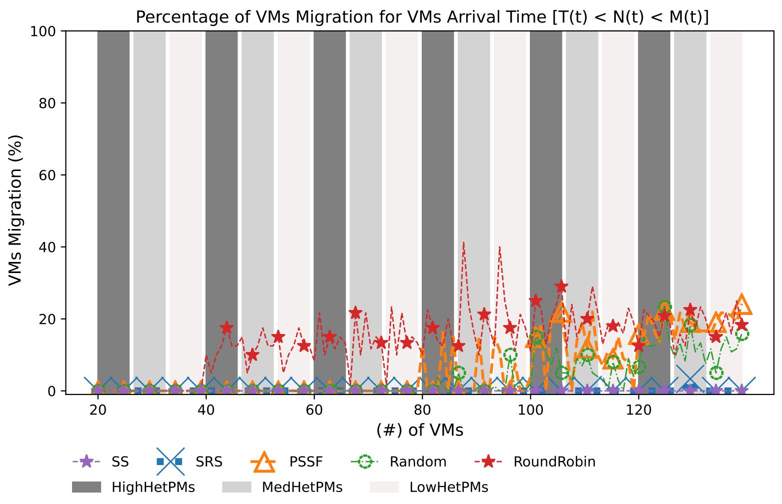
As we explained in the VMs migration algorithm section, in Section 5.4, only the normal and malicious VMs will be selected for VMs migration. Thus, in Figure 5.17c, we can see clearly that the Mig_{vms} is considered the lowest among the three arrival times due to the selection of malicious and normal VMs only. Moreover, the reason for this outcome, low Mig_{vms}, is that the target VMs arrive firstly and allocated in a time before normal and malicious VMs. When the normal VMs arrive, there will be no need to trigger the VMs migration because the normal VMs can be co-located with target VMs. However, when the malicious VMs arrived, VMs migration will be needed at some point, especially because the resources are started to limiting, and the free resources on the PMs are utilised.



(a)



(b)



(c)

Figure 5.17: VMs Migration under Group VMs arrival, GMTN, GNMT and GTNM Arrival Times.

Moreover, the target VMs already allocated, which makes it difficult for the allocator to obtain a secure allocation without triggering the migration. Hence, after the VMs migration triggers, only the normal VMs will be available for migration, and even if they are selected for migration, the percentage of VMs selected for migration will be lower, as the options for new PMs selection is narrow due to the limited of resources at the time of migration. Otherwise stated, the reason that influences the Mig_{vms} to be low in this case 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} showed in Figure 5.11c will produce a higher percentage than other cases of the same algorithm under different arrival times.

On the other hand, in Figures 5.17a and 5.17b, VM migration options for VMs migration is high compared to the previous case. For instance, in the first figure, when the migration is triggering, only the target VMs ask for migration, if needed, and there are many options for VMs migration, as the normal VMs did not arrive yet. The case of high Mig_{vms}, at this stage, will be occurring mostly for the spreading behaviour algorithms, which is explain the reason for PSSF and RR having high Mig_{vms} at this early stages. The early stages mean when the available resources are still not limited and with more free PMs options. Similarly, in the second figure, when target VMs trigger VMs migration, it will be having many VMs options ready for migrations because both the normal and malicious VMs are allocated at this stage.

Furthermore, the relation between the VMs migration and PMs heterogeneous is unclear, as there is no indication or clear pattern that links them. However, when comparing the same number of VMs cases, for example, when the number of VMs is 120, some remarks might be considered for future allocations. For instance, there are slights peaks of the Mig_{vms} in high heterogeneous structures compared to the medium and low structures in the stacking-based algorithms. Similarly, the Rand algorithm shows a higher number of Mig_{vms} peaks than the other PMs types structure.

Overall, from the algorithm perspective, VM migration benefits for VMs migration are high for the random-based and stacking-based algorithms, but for the spreading-based ones, 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 Rand 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

We showed, in Figure 5.12, that M_{pms} for this case is low for most cases for all the algorithms and only showed spikes of malicious co-residency when the resources start to limiting. That explains the reason behind the lower Mig_{vms} in this part, shown in Figure 5.18. However, not all algorithms manage to obtain secure allocations without triggering the VMs migration.

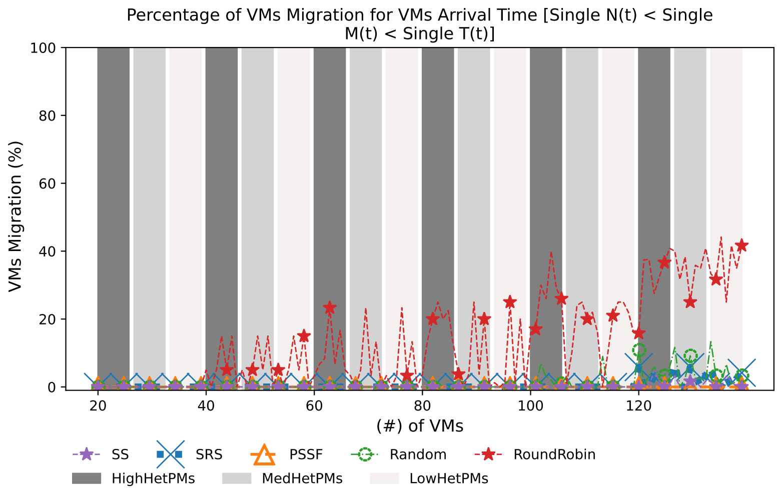


Figure 5.18: VMs Migrations under Single VMs arrival, SNMT Arrival Time.

Specifically, 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, as described in Table 5.2. 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.

Overall, it is normal to see that the Mig_{vms} is high when the resources start limiting, at the 120 VMs case, as it will be challenging for the algorithms to obtain secure allocations without triggering the migration upon the arrival of VMs in a single order.

VMs Migrations for Mixed VMs Arrival

To avoid duplication of similar results, we will only show the case of mixed NMT VMs arrival, as shown in Figure 5.19. This case of mixed VMs arrival represents the worst-case scenario for all the algorithms among the other mixed VMs arrivals.

Overall, 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.

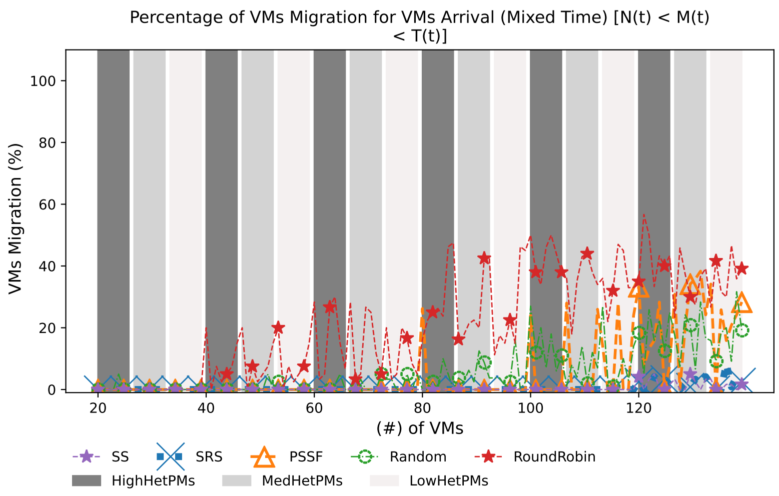


Figure 5.19: VMs Migration under Mixed NMT Arrival Time.

5.6.5 Results of VMs Migration Effect

In the previous sections, we explain the overall malicious co-residency related to the VMs arrival times. Then we show the relationship between the VMs number according to their types on the overall M_{pms}. Afterwards, we present the effect of VMs migration on obtaining secure allocations for the examined cases. However, we did not show how the VMs migration affects the individual VMs and makes them secure. In other words, how does the VMs migration algorithm presented in section 5.4 enhance the overall secure allocation, hence reducing the percentage of the (M_{pms}).

This section aims to evaluate the migration effect of the migration algorithm of all the compared algorithms under different arrivals times. Specifically, we will show how migrating the malicious and normal VMs only enhances the overall secure allocation for the algorithms. Moreover, we will show the algorithms that most benefit from the migration and in which cases.

Furthermore, we calculate the percentage of infected target VMs compared to the total target VMs available for an allocation, denoted as (I_t), as follow:

$$I_t = \frac{I_{tvm}}{A_{tvm}} \tag{5.3}$$

Where the (I_{tvm}) specify the number of target VMs co-located with malicious VMs, and the (A_{tvm}) specify the total number of target VMs available for an allocation.

Figures Explanation

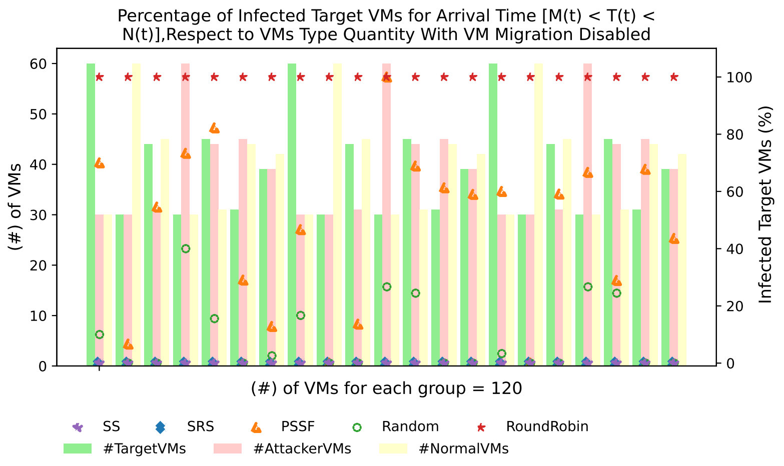
As an example, in Figure 5.20, similar to the previous sections, we compare five algorithms representing different VMs allocation behaviours. However, we will have two figures representing the situations when the migration is disabled and enabled to compare them for all the algorithms. For instance, in Figures 5.20a and 5.20b, we examined the migration effect by studying the percentage of infected target VMs in both situations. We have three axes, one x-axis and two y-axes. The x-axis represents the number of VMs when they are 120 VMs. The first y-axis on the left of the Figure represents the number of VMs according to their type. We have three colours for the VMs type; green for the target VMs, red for the malicious VMs and yellow for the normal VMs. The second y-axis on the right of the Figure represents the percentage of target VMs that co-located with malicious VMs, denoted as (I_t).

VMs Arrival GMTN

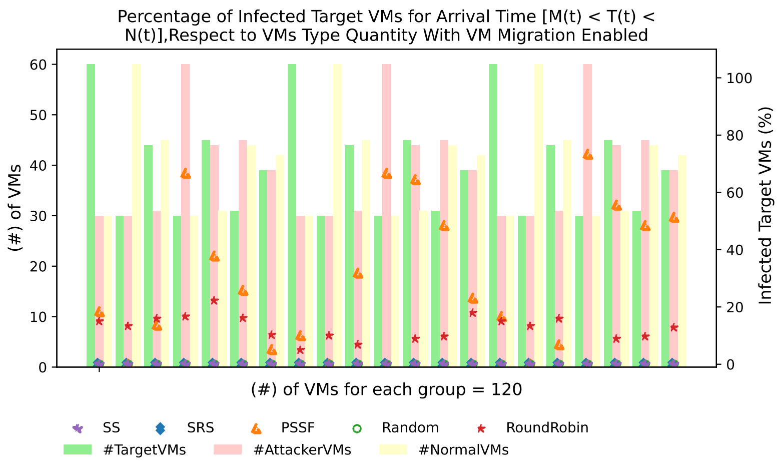
As stated, the goal of this part of the evaluation is to examine the effect of VMs migrations for the individual VMs instead of the PMs, like we did in Sections 5.6.2 and 5.6.3. In other words, examining the effect on preventing the target VMs from being co-located with the malicious VMs, thus enhancing the overall secure allocation. This part will only show and discuss the GMTN arrival time situation, as the other group VMs arrivals show a similar effect.

Generally, in Figure 5.20, there is a clear positive effect of the VMs migration on all the proposed algorithms, which can be seen by comparing the level of I_t on the first figure, Figure 5.20a, with the I_t on the second one, Figure 5.20b. Alternatively stated, for all the algorithms and most situations, the I_t decreases significantly from when the VMs migration is disabled to the case where the VMs migration is enabled. This outcome means that the VMs migration algorithm successfully, to some point, reduces the effect of the malicious co-residency.

Briefly, our proposed solutions aim to defend against side-channel attacks (SCA) by minimising the malicious co-residency between the target and malicious VMs. Thus, reducing the effect of this attack by keeping the VMs classified as malicious VMs apart from the VMs that classified as a target or have highly sensitive data. Moreover, the malicious user who performs SCA depends on collecting information from the target user who shares the same PM through a side-channel. Hence, the malicious user will try to collect as much as possible information from this co-location. In order to accomplish this process, the malicious VM will try to utilise many VMs to have the chance of co-locating with the target VMs and collecting as much as possible information. Therefore, the migration algorithm reduces this effect by reducing the number of malicious VMs sharing the same PMs with the target VMs. In other words, even if the allocation algorithm failed to produce secure allocation, which means having PMs with malicious co-residency, the migration algorithm will try to reduce the number of compromised target VMs, to reduce the effect of SCA.



(a)



(b)

Figure 5.20: The Migration Effect on the Secure Allocation under GMTN Arrival Time.

Specifically, the RR algorithm has the most benefits from the VMs migration among the other algorithms under all the groups of VMs arrivals. The percentage of the I_t dropped significantly in all the cases that have been examined because the RR does not restrict the number of VMs belonging to the same user of allocating together. Thus, the algorithm will migrate the VMs more often to produce secure allocations as long as they have been spread across the entire available PMs. Thus the chance of obtaining a secure allocation for individual VM is greatly high even if the available resources are limited.

On the other hand, the PSSF algorithm did not fully benefit from this VMs migration process, as not all the situations reduce the I_t. On the contrary, a few situations lead to a slight increase in the compromise target VMs.

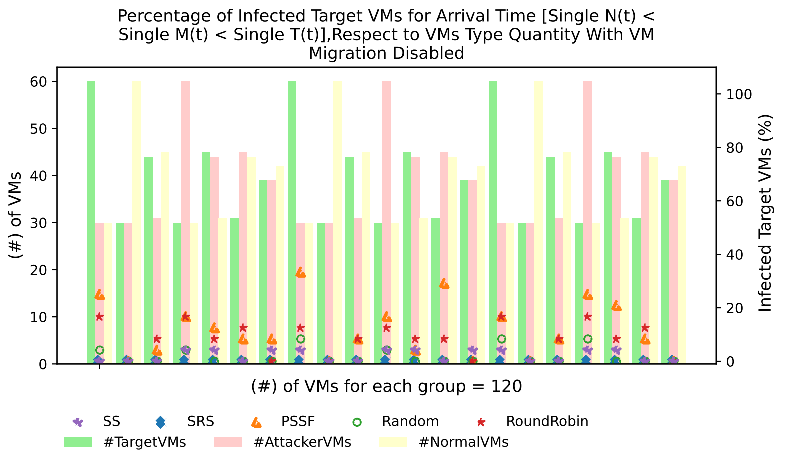
This outcome is because the PSSF constraint the number of VMs, belonging to the same user allocated on the same PM. Thus, leading to fewer options for migration, and in some cases leading to many unwanted migrations. For example, in a case where a PM has a malicious co-residency, one malicious VM sharing this PM with many target VMs, and another PM has available resources, but this PM already has the maximum number of VMs for this malicious user. In this case, the migration will not occur because of the restriction on the user number, thus producing malicious allocation.

The Rand algorithm also benefits from the migration algorithm as all the cases are dropped to the lower levels, which means that the VMs migration successfully produces secure allocations. The effect on SS and SRS algorithms is absent as they did not suffer from malicious co-residency in the first place; thus, the VMs migration is not triggered.

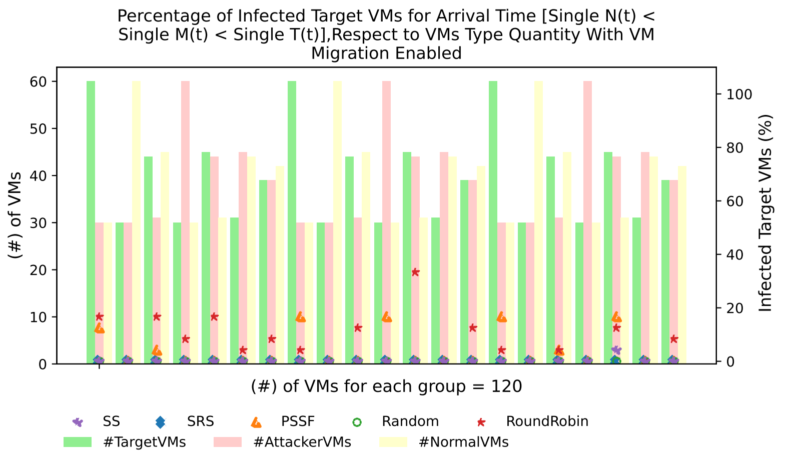
VMs Arrival SNMT

This part will show the impact of our VMs migration algorithm when discussing the single VMs arrival in Figure 5.21. Broadly speaking, developing an efficient algorithm depends on how many situations the algorithm can handle without producing a bad result. In our case, developing a migration algorithm that can lead to secure VMs migrations for all the cases examined. This efficiency level is rarely reached to the perfect level where all the situations can produce a positive result. In other words, it is nearly impossible to develop a VM migration algorithm that always produces a secure VM allocation. Therefore, we are aware of this drawback, and in this section, we will discuss it further.

Overall, the benefits from VMs migrations are limited because of the low percentage of M_{pms} and Mig_{vms} of this situation. The SS and SRS algorithm already produces a secure VMs allocation for these cases; as such, the VM migration was not needed. The Rand and PSSF algorithms produce positive results by reducing the percentage of I_t after the migration for most cases.



(a)



(b)

Figure 5.21: The Migration Effect on the Secure Allocation under SNMT Arrival Time.

However, the RR algorithm was negatively impacted after the VMs migration was triggered, repeated in many cases. This outcome means that the percentage of I_t has increased after the VM migration triggered instead of dropping, as it was intended to do. The possible reasons for such a result are the sequence of performing the allocation and the spreading behaviour of the algorithm. To explain, the sequence of performing the allocation means that each VM will be allocated before the arrival and allocation of the upcoming VM. For example, if we have a queue of unallocated VMs ready for allocation, the algorithm will try to allocate the first VM, trigger the VMs migration if needed, and then try to allocate the second VM by following the same process and so on. The migration at the early stages of this queue of VMs can be useful to produce secure allocations. However, at some point, especially when the resources start limiting and the available PMs options narrowing, the malicious co-residency will occur, even with the migration enabled, leading to high I_t, as we have seen in this situation. This outcome is driven mainly by the behaviour of the RR algorithm that depends on allocating the VMs by spreading them on the available PMs. In other words, the allocation algorithm can not predict, at this stage, whether this VM allocation can eventually lead to a secure allocation or not. Instead, it will try to secure the allocation by following the algorithm's heuristic, which possibly avoids the malicious co-residency occurring. Therefore, at some point, the VMs migration will lead to a secure allocation, but with more VMs arriving, and with the algorithm's spreading behaviour, it will lead to a malicious co-residency.

VMs Arrival (Mixed MTN)

Overall, there is a clear positive effect of the VMs migration on all the compared algorithms. Most examined situations led to I_t reduction, which indicates that the VMs migration algorithm profitably reduces the effect of the malicious co-residency for individual VMs. This part, in Figure 5.22, will only show and discuss the mixed MTN arrival time situation to avoid results duplication as the other mixed VMs arrivals show a similar effect.

The significant positive impact is shown clearly in RR, Rand and PSSF algorithms, as they are the ones that benefit more from this effect compared to SS and SRS algorithms. The reason for this positive impact is the configuration of VMs arrival, where the VMs at some point arrives at groups, then singles, as described in Table 5.2. Group VMs arrivals lead to many options for VM migration and, therefore, much available space for the upcoming VMs after the migration is completed. For example, when a group of VMs classified as malicious arrives at some point, they will be potentially allocated to the same group of PMs due to co-residency constraints forced on all the algorithms. Thus, the migration trigger is triggered when a target VM arrives, and the algorithm could not find a secure allocation due to the limited resources or the available PMs occupied by malicious VMs. At this stage, more VMs will be selected for migration because of the availability of options, leading to more spaces for the upcoming target VM. Therefore, securing the individual target VMs from being co-located with malicious ones and reducing the I_t.

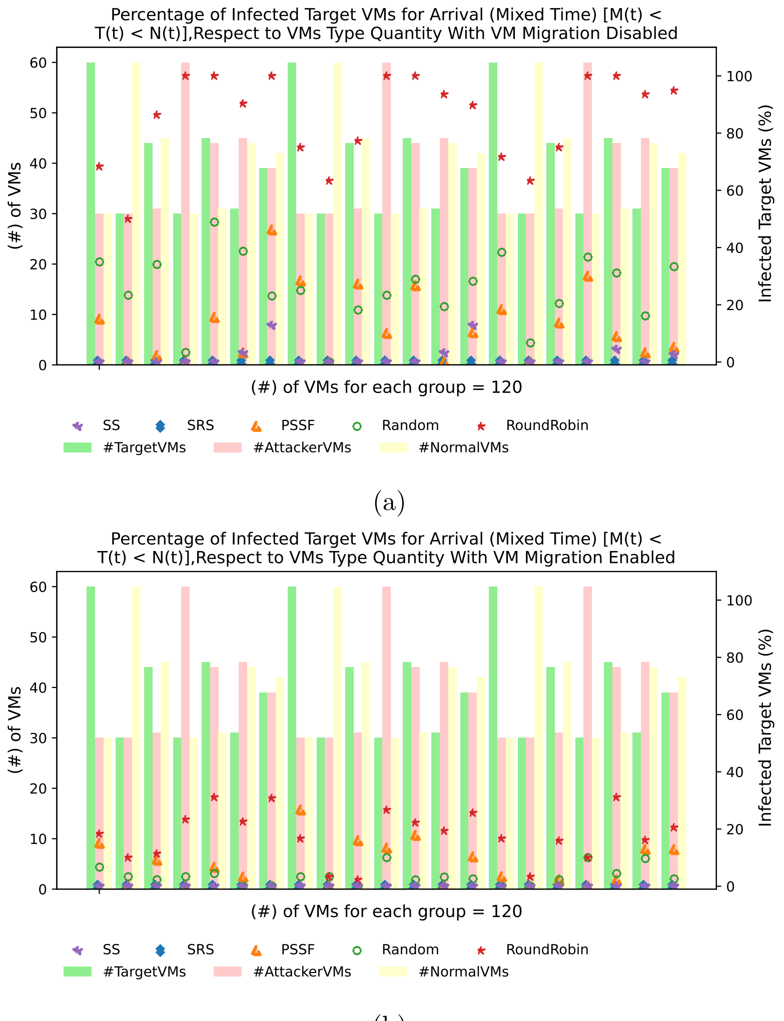


Figure 5.22: The Migration Effect on the Secure Allocation under mixed MTN Arrival Time.

5.6.6 Results of PMs Usage

In our presented algorithms, SS and SRS, we aim to obtain a secure VM allocation while reducing the number of PMs used for an allocation, thus, utilising BPP for this part. Therefore, We utilise the calculation of the percentage of used PMs compared to the total available PMs, presented in Section 4.4.3, in Eq. (4.16). As such, 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.

Figures Explanation

As an example, in Figure 5.23, we compare five algorithms representing different VMs allocation behaviours, as explained earlier. The x-axis represents the number of VMs starting from 20 up to 120, where the y-axis represents the PMs usage or percentage, which we will refer to as ($Usage_{pms}$). The vertical shading colour represents the PMs heterogeneity level, high to low.

VMs Arrival (GNMT)

In this part, we will discuss the effect of the $Usage_{pms}$ for the group VMs arrivals, specifically for GNMT arrivals. The other two types of groups arrival show the same impact; thus, we only consider this case to avoid results duplication.

Overall, in Figure 5.23, there is an indication of the resource usage, efficiency towards obtaining a secure allocation. In other words, in our proposed algorithms, SS and SRS, 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 Rand 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.

Unlike the other algorithms, SS and SRS only use its full capacity of the available PMS, in all VMs arrivals cases, when the VMs reaches 120 VM, when the available resources are limited. Moreover, when the number of options for available PMs is reduced, $Usage_{pms}$ increases for all the algorithms. Thus, making the stacking-based behaviour algorithms the most efficient in reducing the $Usage_{pms}$ and the power consumption of the PMs.

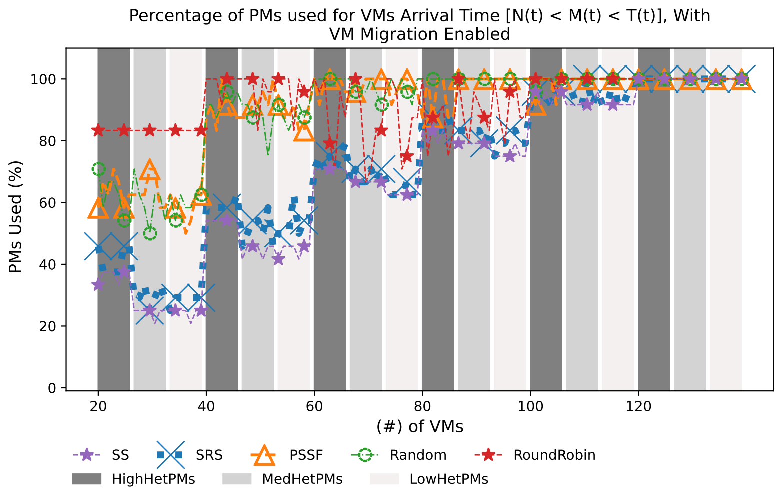


Figure 5.23: PMs Usage under GNMT Arrival Time.

VMs Arrival (SNMT)

Similarly to what we discuss in the previous part, this part of single VMs arrival, in Figure 1, shows a similar impact of the $Usage_{pms}$. The SS and SRS show the best $Usage_{pms}$ among the compared algorithms under most cases, even when the resources start limiting. Moreover, the RR algorithm is the worst case due to its spreading behaviour, while Rand and PSSF are only better when the available resources are not limited.

Moreover, the effect of Mixed VMs arrival is the same; thus, the figures or discussions will not be included.

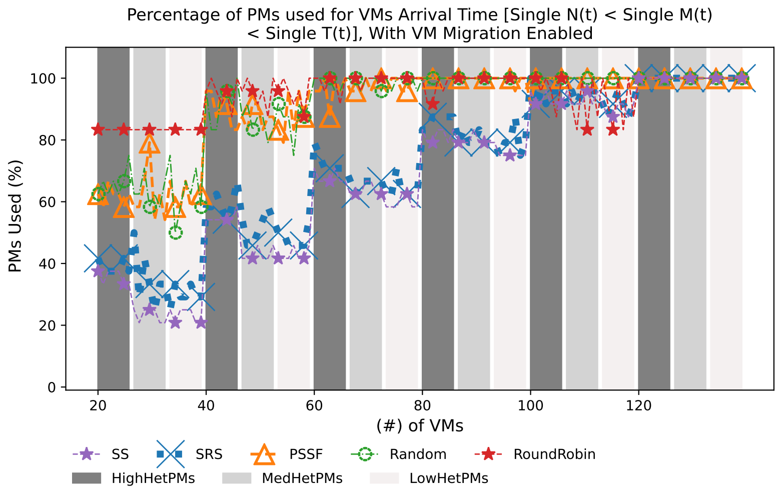


Figure 5.24: PMs Usage under SNMT Arrival Time.

5.6.7 Statistical Analysis of VMs Arrival Time

Here, we examine the data further to evaluate whether VMs arrival time impacts the malicious co-residency outcome. We examine the data to show if there are significant differences between them using statistical analysis. We conduct a *t-test*, a commonly used statistical analysis test that identifies a significant difference between the average of two sets of data, and they are not random or biased [24]. A difference implies that VMs arrival time or allocation behaviour makes a difference. The result from the *t-test* produces a $p - value$, which is the probability value that identifies the confidence level in which the tested data are different from each other. A large $p - value$ reflects that the data results happen by chance, the smaller $p - value$ reflects the confidence in the data tested. For example, if the $p - value <= 0.05$, then we have a confidence interval $>= 95\%$ that the data are significantly different. Moreover, if the $p - value >= 0.05$, the confidence interval is $<= 95\%$ that the data are significantly different, which is statically unacceptable.

We calculated the result of $p - value$ from the *t-test* by conducting a comparison between the algorithms under the defined VMs arrival time. The data set comparison represents the situations in Figures 5.11, 5.12 and 5.13. The situations include the comparison between the five algorithms SS, SRS, PSSF, Rand and RR. The *t-test* compares two sets of data; therefore, as such, we were able to compare two algorithms at each time. For example, compare the data of SS with each algorithm under the types of VMs arrivals. All of the $p - value$ from the *t-test* resulted in ($P < .03$). It indicates a significant difference between the compared algorithms, and there is an impact on their behaviour and VMs arrival times.

5.7 Evaluation of Heterogeneous VMs Allocation

In the previous section, Section 5.6, we presented an evaluation of the proposed VMs allocation algorithms under different scenarios and structures where the resources of the VMs are not heterogeneous. This section will repeat the same steps of the previous evaluation, but with heterogeneous VMs, i.e., the VMs required resources are heterogeneous.

This section aims to evaluate the behaviours of VMs allocation algorithms, of heterogeneous VMs, under different situations and conditions. Thus, evaluating the chance of the malicious co-residency occurring for the proposed algorithms under several VMs structures, PMs structures and VMs arrival times. Moreover, studying the factors that affect overall malicious co-residency when the VMs are heterogeneous. As mentioned before, the factors include; the VMs arrival time, the effect of the heterogeneity level of the PMs, the effect of VMs structure, and the effect of the VMs migration. All these factors and the compared algorithms are explained in detail in Section 5.6.

Moreover, we utilise the same simulation tool, CloudSim, described earlier, but with changing the VMs specification structure. As such, we required realistic traces of the VMs from an existing cloud provider to examine the algorithms under actual heterogeneous VMs traces. Thus, we utilised the Azure VMs traces published by the Microsoft team, which contains the VMs workload of Azure [22]. These VMs traces will be loaded by CloudSim and created on the simulation environment to be simulated under different conditions and structures. The Azure traces contains over two million VMs recorded during 30 consecutive days on the Azure cloud data centre. The data set contains information related to each VM, such as user id, time VM created or deleted, and demanded resources such as CPU and RAM. Each user may have many VMs, and each VM may have a different or same set of resources. More information about Azure traces is described in detail in **Appendix** C.

Overall, and similar to the process of the previous section, this section will examine the effect of the secure VMs allocation, VMs migration, and PMs usage during the allocation for the presented algorithms.

5.7.1 Experimental Setup

The experimental setup and process for this section are similar to the one presented in the previous section, Section 5.6.1, except for changing the structure of VMs, and PMs. In other words, this section utilises the VMs structure of the Azure data set; as such, the VMs number and structure will be changing. Moreover, therefore, the number and structure of PMs will be changing. The following section will describe more about the Azure data set integration with the simulation environment. Furthermore, describing the VMs and PMs structure of the utilises traces.

Azure Traces

As described in **Appendix** C, the workload of the Azure traces contains comprehensive information about the lifetime of the VMs in Azure cloud systems. It includes VMs resources such as VMs lifetime, CPUs and memory utilisation, and the VMs users information. The duration of the VMs traces was collected from the Azure cloud data centre for 30 consecutive days.

The goal of utilising Azure workloads is to produce a set of heterogeneous VMs in the matter of resources structure. However, the problem with the existing tracers is that each cloud user can have many VMs with the same resource structure, making the VMs non-heterogeneous. For example, 150 VMs belonging, and all these VMs have the same CPU and RAM, 2 CPU core and 4 RAM size. In this part of our work, we aim to examine the heterogeneous VMs on the secure VMs allocation; thus, we made further steps to the existing workloads to fit our purpose. As described in **Appendix** C, in Figure C.1, we proposed cleaning the VMs traces to produce a group of VMs with a heterogeneous structure. The cleaning step of the data is simple yet effective to our purpose, which performs a selection of one VM from each cloud user. Alternatively stated, from each user, we only select one VM belonging to this user, producing 6687 VMs with a heterogeneous structure. Performing this step does not imply that the existing VMs traces are not heterogeneous; on the contrary, they are heterogeneous, but not enough to fits our purpose.

VMs Number, PMs Number and PMs Heterogeneity

It is a challenging task to design a simulation where the required resources of the VMs and the available resources of the PMs are heterogeneous. In other words, in this section, our goal is to examine the heterogeneous VMs allocation when the resources are limited on the available PMs. However, defining the number of PMs for such a goal is not a straightforward process, like in the non-heterogeneous VMs evaluation. Because reaching the point where the required resources by the VMs, for example, vRAMs and vCPUs, can match the available resources of the PMs is not easy to accomplish in a heterogeneous environment. However, we design the PMs resources to accommodate the available VMs while limiting the available resources. Therefore, making the allocation process very challenging for the compared allocation algorithms.

The number of VMs is 6687 VMs, as described in the previous section, while the number of PMs is different based on heterogeneity. The number of PMs in high heterogeneous structure is 1120 PMs, in medium heterogeneous is 840 PMs and in low heterogeneous is 560 PMs. We intentionally make the PMs structure to evaluate the effect of the PMs heterogeneity levels on obtaining secure allocations. Moreover, examine the algorithms when the number of PMs is high and diverse and low and less diverse. The sum of available resources of the PMs can accommodate the required resources of the VMs. In the non-heterogeneous VMs evaluation, we study the effect when the available recourse are not limited, thus, in this section and to avoid duplication, we only will examine the situations of limited resources. Moreover, we only consider the case where the available resources are limited.

Additionally, we consider three types of PMs structure, or level of PMs heterogeneity, High, Medium and Low heterogeneous PMs, which indicates how much the PMs are different from each other concerning the available resources. A high heterogeneous PMs means that the majority of the PMs are different from each other. Otherwise stated, upon the arrival of the VMs, under high heterogeneous PMs, the available resources of the PMs will be significantly different. In medium heterogeneous PMs, the available resources will be less different, meaning that only half the available PMs resources will be different, while the other half will have the same set of resources. Lastly, for the low heterogeneous PMs, all the sets of the available PMs resources will be the same. Overall, the available resources of the PMs are configured to accommodate the required resources of the VMs.

From Azure Traces to CloudSim

This part will explain the journey of the VMs from the Azure data set to the cloud simulation environment and how they have been examined and allocated. Moreover, how the VMs are classified into the defined VMs type and sorted according to the VMs arrivals types [22].

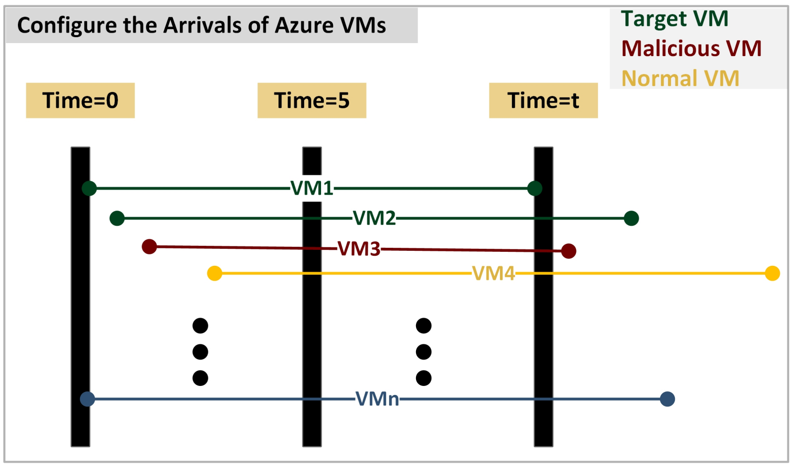


Figure 5.25: Configuring the arrivals of Azure VMs.

As shown in Figure 5.25, we have deployed three steps to configure the VMs loaded from the Azure data set into the CloudSim. The first step is to sort the VMs according to their launch time (arrival time); each VM arrived or started at a specific time; thus, we sorted the VMs in ascending order to define their arrivals. The second step is to define how many of these VMs will be either target, malicious or normal VMs. In other words, we are defining the VMs type structure, as we explain in Section 5.6.1. We configure seven VMs type structure that defines how many VMs, according to classified type, are available in each experiment. The third step is to configure the arrival of these VMs according to their type. As explained in Table 5.2, we have defined several VMs arrivals; thus, we aim to examine the arrival effect on the heterogeneous VMs. If the defined VMs arrival is GMTN, the first group of VMs will be assigned as malicious VMs, the second one as target VMs and the third one as normal VMs. However, this assignment will not change the original form of the VMs arrivals described in the Azure traces; it just classifies them according to the defined arrivals types. Then, defining how many VMs are target, malicious or normal VMs for this arrival type depends on the structure of the VMs, in Section 5.6.1. For example, if we only consider the first ten VMs of azure traces, each VM has a different set of required resources, i.e., heterogeneous. Then, we will perform three steps on them. First, change their classification according to the defined VMs type. Second, specify how many are target, malicious or normal VMs. Third, configure whether the first arrival VMs or last arrival VMs considered targets, malicious or normal VMs, i.e., specifying the VMs arrival. Specifically, if the GMTN is implemented, one possible distribution will put the first three VMs as malicious, the second three as target and the last four as normal VMs. Another possible distribution for the same VMs arrival type put the first two VMs as malicious, the second five as target and the last three as normal VMs. These distributions of VMs type and VMs arrival is explained in Section 5.6.1 and Table 5.2.

5.7.2 Results of Malicious Co-residency For Heterogeneous VMs with Limited Available Resources

This section will examine the M_{pms} respecting the VMs type number. The goal of this evaluation is to examine the effect of heterogeneous VMs according to their type numbers. In other words, is the increase or decrease for any VMs, according to their type classification, will impact the M_{pms}. In addition, we will examine the effect of PMs heterogeneity level on the overall M_{pms} for the algorithms. Moreover, in this section, we only will show the results when the resources are limited, which means when the number of VMs equal 6687 VMs.

Figures Explanation

As an example, in Figure 5.26, similar to the previous section, we compare five algorithms representing different VMs allocation behaviours. The difference from the previous Figures is that we have three axes, one x-axis and two y-axes.

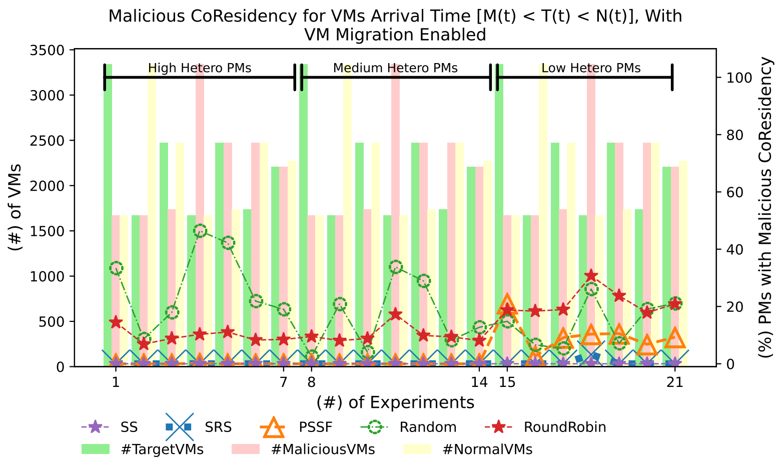
The x-axis represents the number of experiments, starting from 1 up to 21. From 1 to 7, the first seven experiments represent the experiments performed under the high heterogeneous PMs and by trying the seven cases of VMs type structure, described in Section 5.6.1. Also, from 8 to 14, the second seven experiments represent the experiments performed under the medium heterogeneous PMs. Further, from 15 to 21, the third seven experiments represent the experiments performed under the low heterogeneous PMs. In all the experiments, the number of VMs is reached up to 6678, as described in the experimental setup section.

There are two y-axes; the first y-axis on the left of the Figure represents the number of VMs according to their type. We have three colours for the VMs type; green for the target VMs, red for the malicious VMs and yellow for the normal VMs. The sum of these three colours always will be 6687 VMs, as we only examine the 6687 VMs case, i.e., when the resources are limited. The second y-axis on the right of the Figure represents the M_{pms} we explained in the previous sections. The horizontal black lines represent the PMs heterogeneity levels of this part of the experiments. We have three PMs heterogeneous levels, which are high, medium and low heterogeneous PMs.

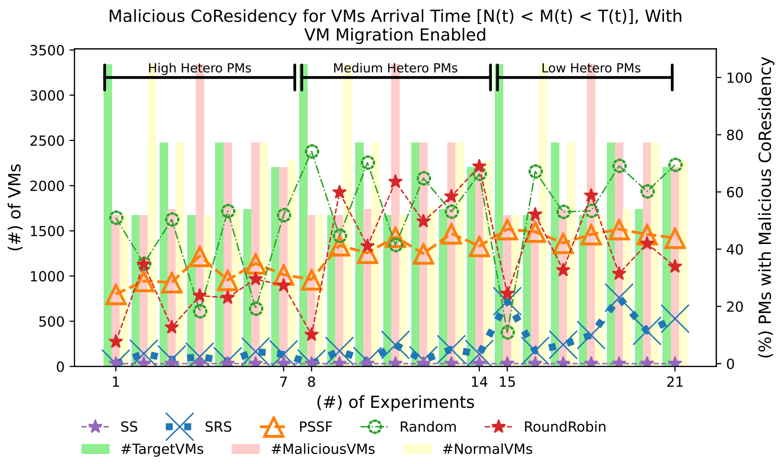
Malicious Co-residency for Heterogeneous Group VMs Arrival

This part will discuss the effect of group VMs arrival on the secure allocation of heterogeneous VMs under different VMs type structures and PMs heterogeneity levels. Generally, in Figure 5.26, the stacking-based algorithms are performing the best among the compared algorithms: the random-based algorithm and the spreading-based algorithms perform poorly under most of the examined cases. Moreover, compared to the evaluation of non-heterogeneous VMs, in Section 5.6, the Rand algorithm shows high M_{pms} in most situations. Furthermore, although they follow the same allocation behaviour, the SRS algorithm sows a higher M_{pms} than the SS algorithm. The PSSF and RR are performing similarly; however, the M_{pms} of the PSSF algorithm is performing better than the RR algorithm.

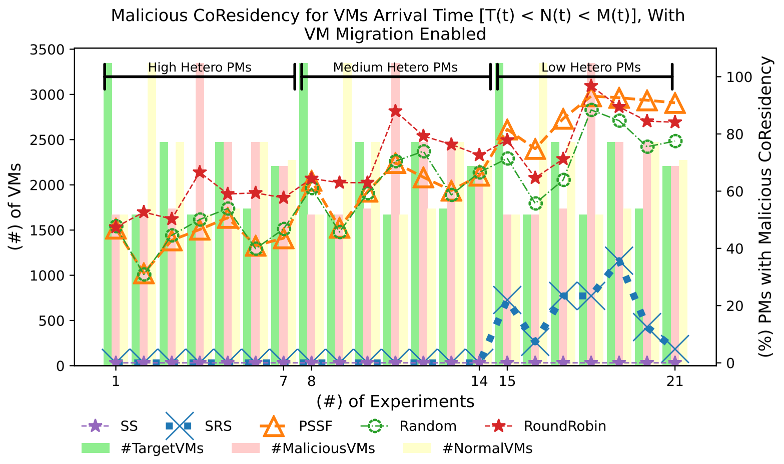
Specifically, the SRS algorithm showing a significant high M_{pms} when the PMs heterogeneity level is low, meaning the PMs configuration is similar, and the number of PMs is limited. As we described earlier in the PMs configuration setup, the number of PMs in low heterogeneous PMs is lower than in high or medium heterogeneous PMs. Thus, the chance of producing malicious co-residency is higher, even for all the compared algorithms, which makes it more challenging. However, the SS, which follows the same behaviour as the SRS, shows very low M_{pms}, indicating the positive impact of the SS algorithm's deterministic approach. The SRS algorithm is non-deterministic, meaning that it has a randomness factor implemented in it, leading to different results in each run. Thus, a different result could be obtained when running the SRS again, but it is highly unexpected.



(a)



(b)



(c)

Figure 5.26: Malicious Co-residency under GMTN, GNMT and GTNM Arrival Time for Heterogeneous VMs.

In addition, the PSSF algorithm is also showing a higher M_{pms} when the PMs heterogeneity level is low compare to the other levels, but it is not significant as it is in the SRS. Also, the RR algorithm shows a higher M_{pms} in this level of PMs heterogeneity but also show a similar outcome on the medium level. Due to its random behaviour, the Rand algorithm did not show a clear pattern of the effect of PMs heterogeneity level. Nevertheless, specificity performs well when the number of PMs is less, and PMs heterogeneity is low, in the Figure 5.26a case.

Moreover, the effect of the VMs arrival on SRS is clear, as the algorithm showing lower M_{pms} when the malicious and target VMs arrives before the arrival of normal VMs, as shown in Figure 5.26a. This outcome means that these two VMs types that lead to high M_{pms} are allocated when the available resources are not limited. Moreover, by the time the normal VMs arrived and the resources are limited, it will not cause any issue as it can be allocated with any of them. The same significant effect and low M_{pms} under this type of VMs arrival can be seen on the other algorithms, RR, PSSF and Rand. Furthermore, the case in Figure 5.26c considered the worst-case scenario of VMs arrivals for all the algorithms. The reason behind this outcome is the arrival of the malicious VMs lastly after the resources consumed and few PMs options available. Unlike the case in Figure 5.26b, the VM migration has its effect on the secure allocation, as, by the time malicious VMs arrived, only the normal VMs can be migrated. Thus, a few VMs can be migrated compare to the total number of VMs, which leads to less available options for producing secure allocation. However, in Figure 5.26b, when the target VMs arrive and the resources are limited, many migrations will be triggered to secure an allocation. Hence, we can see clearly that the M_{pms}, in this case, is lower than the case in Figure 5.26c.

Additionally, the effect of VMs numbers, according to their type, is similar to what was explained in the previous evaluation of non-heterogeneous VMs. Briefly, there is 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.

Malicious Co-residency for Heterogeneous Single VMs Arrival

In general, in Figure 5.27, the single VMs arrival yielding a positive outcome of M_{pms} for most of the examined situations. For instance, the SS and SRS algorithms are the best performing algorithms out of the compared ones as they show the least result of M_{pms}. While the RR, and due to its spreading behaviour, is performing the worst among the algorithms. The Rand and PSSF algorithms are generally better in most cases when comparing them with the same structure of the group VMs arrival.

Despite the slight positive effect in the low heterogeneous level, the overall effect of the PMs heterogeneity levels is not a highly significant result of the M_{pms} for all the algorithms. Moreover, according to their type, the number of the VMs continues to have the same impact as the spike in the malicious or target VMs number could potentially lead to high M_{pms}.

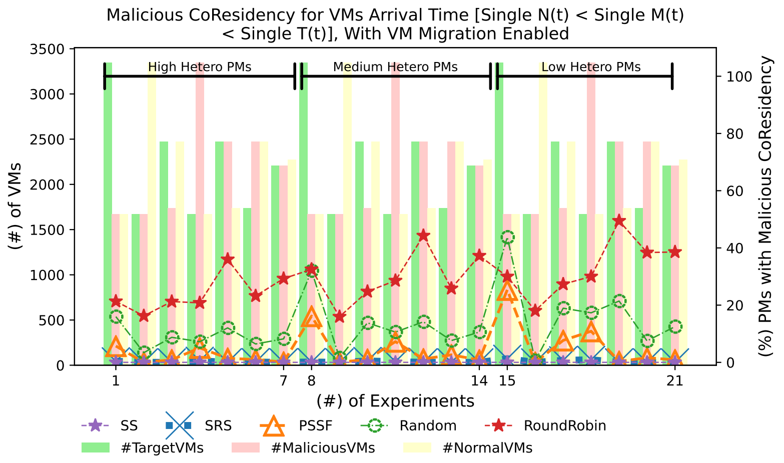


Figure 5.27: Malicious Co-residency under SNMT Arrival Time for Heterogeneous VMs.

Malicious Co-residency for Heterogeneous Mixed VMs Arrival

In this part, in Figure 5.28, we only show the mixed NMT arrivals as the other two arrivals types show similar outcomes. Generally, there is a clear indication that the high level of PMs heterogeneity positively impacts most of the algorithms. Alternatively stated, in this work, we presented three groups of PMs structures that have the same amount of available recourse; however, they have been designed with different resources structures, leading to different outcomes of the M_{pms}. Thus, under the same number of VMs and VMs classification, designing a PMs structure with a high number of PMs and a highly diverse structure is more secure than a low number of PMs with a similar structure.

Furthermore, unlike the RR that share the spreading behaviour, the PSSF algorithm produces more secure allocations than the RR algorithm or even the Rand algorithm. The possible reasons for this situation; the positive impact gained from the VM migration algorithm and the arrival of most target VMs at last. When the target VMs arrive after the malicious and normal VMs, the options for VMs migration will be higher due to how the VM migration algorithm works, which is migrating the malicious and normal VMs only. Thus, creating more chances to produce secure allocations for the algorithms upon the arrival of the target VMs.

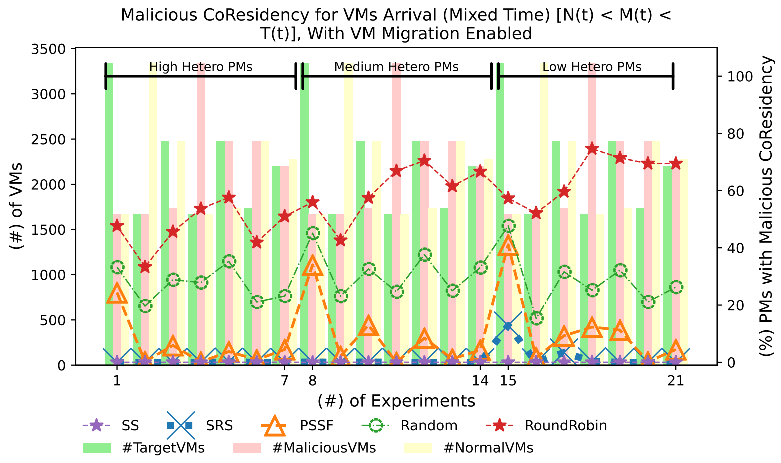


Figure 5.28: Malicious Co-residency under Mixed NMT Arrival Time for Heterogeneous VMs.

5.7.3 Results of Heterogeneous VMs Migrations

This section will introduce an evaluation of the VMs migration for the heterogeneous VMs and study its effect on obtaining secure VMs allocations. The goal and evaluation process for this section is similar to the one presented in Section 5.6.4. Moreover, and as explained, the VMs migration is indication of amount of processing needed, by any of the algorithms, to produce a secure allocation. Thus, our goal is to examine the effect of VMs migration, denoted as Mig_{vms}, on obtaining the secure VMs allocation of the compared algorithms. Hence, studying the relation of migration on the percentage of M_{pms}.

VMs Arrival (GTNM)

In this part, we only examine the GTNM case as it shows the lowest Mig_{vms} for all algorithms and the highest cases of M_{pms}.. This case indicates that migration is not beneficial for all the compared algorithms under this arrival type. Moreover, the other cases, which have a slight increase in the Mig_{vms}, show a similar pattern as the migration outcome.

The most significant of this case is that the percentage of PMs who have malicious co-residency is high, especially in RR, PSSF and Rand algorithms. As shown in Figure 5.26c, in most cases of the three algorithms, the M_{pms}. reached a significantly high M_{pms}. Nevertheless, the migration showing low Mig_{vms} for the same cases, as shown in Figure 5.29. This outcome could be due to multiple factors; the heterogeneity of the required VMs resources and the behaviour of allocation for the algorithms.

The heterogeneous VMs resources make it challenging for the allocator to obtain a suitable, not secure, allocation, especially if the VM requires very high resources, for example, 64 GB of RAM and 32 CPUs, which is what some VMs in the Azure traces required. Thus, obtaining a secure allocation for such VM is more challenging due to the high demanded resources, especially when the resources are limited, like in our case; therefore, the VMs migration will have the same impact.

Moreover, the behaviour of the allocation algorithms contributes to this outcome as it leads to preventing the migration algorithm from obtaining the secure allocation for the migrated VMs. For example, the RR algorithms are the worst-case algorithm among the compared algorithms as it showing the highest M_{pms}. and the highest Mig_{vms}. When the VMs allocated and migrated initially using RR, at the point of time, the PMs options for obtaining a secure allocation for the upcoming VMs will be impossible. Especially when the VMs arrivals configure in this way, meaning the malicious VMs will arrive last after the target VMs already allocated and spread across the entire available resources. In a nutshell, we can conclude that the heterogeneous require resources, allocation behaviours, and VMs arrival type could lead to high M_{pms}. when the resources are limited.

Overall, the stacking-based behaviour algorithms performing the best among the compared algorithms and under the same situations. The SS algorithm shows a very low percentage of M_{pms} and Mig_{vms}. However, the challenge for the SRS is clearer to obtain a secure allocation. The reason for this outcome is that the SRS follow a random stacking behaviour in the allocation of the VMs; thus, the selection of the PMs for an allocation happens randomly after checking the eligibility of these PMs. However, this randomness in the SRS is leading to a higher M_{pms} compare to the SS algorithm. Otherwise stated, the VMs allocation process is a sequence of steps that allocate the VMs step by step, thus, VM by VM. In the SS algorithm, this sequence is deterministic, and it leads to the same result at every run. However, for SRS, and even though it follows a stacking behaviour as the SS does, it has the random factor, which leads to a higher M_{pms} for the upcoming VMs. Therefore, the SRS may lead to different results if the allocations process is executed again because the VMs allocation sequence might change and lead to more secure allocations for the upcoming VMs. Due to the non-deterministic approach of SRS and the sequence of VMs allocation, the VM migration did not lead to a desirable result as intended. It still has the positive effect of reducing the level of the malicious co-residency of individual VMs, which we will explain in the following parts.

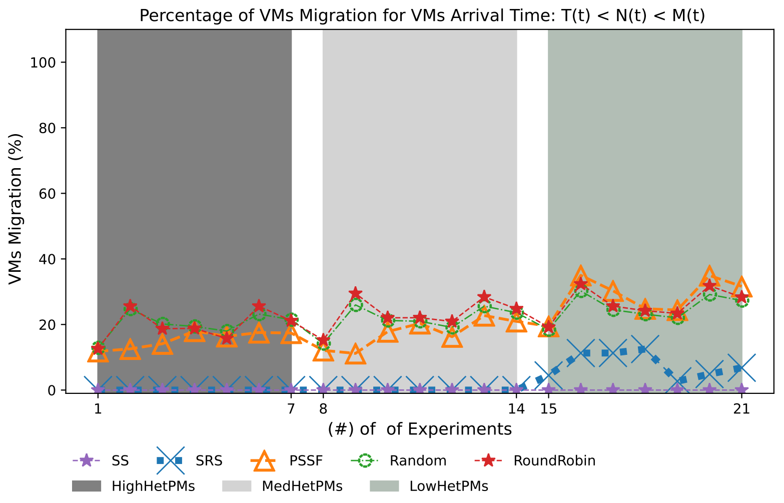


Figure 5.29: VMs Migration under GTNM Arrival Time for Heterogeneous VMs.

VMs Arrival (SNMT)

Generally, in this part in Figure 5.30, the stacking-based algorithms are the ones who most benefit from the migration algorithm, as it leads to more secure allocations. While the spreading one, the RR algorithm, is the worst due to its high Mig_{vms} and high M_{pms}, which indicates few benefits gained from the migration triggering. Overall, the same behaviour described in the previous where the heterogeneity of the VMs resources, especially when they have a high resource demand, and the allocation behaviour, are the most contribution of this outcome.

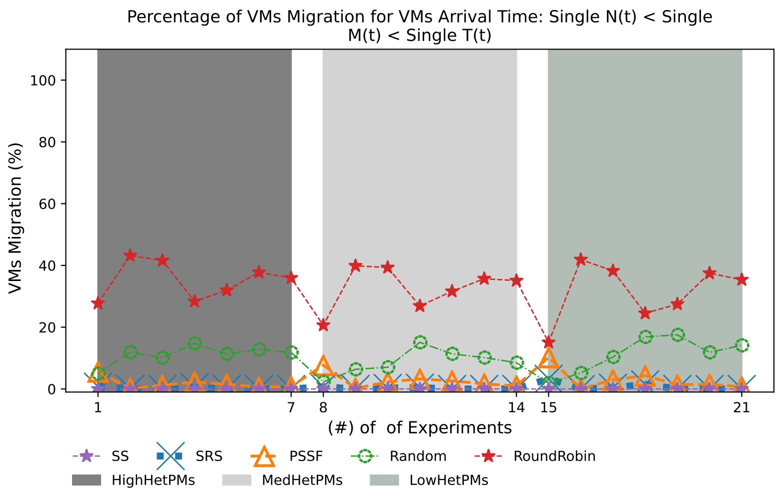


Figure 5.30: VMs Migration under SNMT Arrival Time for Heterogeneous VMs.

VMs Arrival (Mixed NMT)

Similarly, to avoid duplication of the results, the mixed VMs arrivals have similar outcomes in the VMs migration. The most significant outcome of the VM migration is related to the fact that the level of VMs heterogeneity makes it challenging to trigger the VMs migration and therefore obtain secure allocations for the spreading the behaviour algorithms.

5.7.4 Results of Heterogeneous VMs Migration Effect

This section will examine the effect of the VMs migration of the individuals VMs under heterogeneous VMs structure, similar to the study we conducted in Section 5.6.5. Briefly, this section will show how the VMs migration affects the individual VMs and makes them secure, thus enhancing the overall secure allocation. Specifically, we will examine the percentage of infected target VMs compared to the total target VMs available for an allocation, denoted as (I_t).

In the flowing parts, we only will show the effect of the SRS, PSSF and RAND algorithms, as they show the most significant results. Moreover, instead of illustrating the effect per arrival time, we show it for the entire experiments conducted per algorithm, yielding 84 experiments in total as the goal is to examine the effect of the migration in general.

SRS Algorithm

In Figure 5.31, the blue dots indicate the percentage of the target VMs with malicious co-residency when the migration is disabled. In comparison, the yellow dots indicate the percentage of the target VMs that have malicious co-residency when the migration is enabled.

Generally, the SRS algorithms benefit from the migration as most of the individual target VMs, under each situation, manage to decrease the presence of the malicious VMs on the same PM. Alternatively, the percentage of I_t decrees for most cases when the migration is triggered, thus reducing the effect of the malicious co-residency that leads to SCAs. Moreover, the benefits appear clearly in the high and medium PMs heterogeneous and relatively limited for most cases under the low PMs heterogeneity. The possible reason is the number of PMs, and highly diverse resources in high and medium PMs structure, which offers more options for the upcoming VMs and the allocated ones.

Furthermore, the stacking-based behaviour of the SRS algorithm aims to reduce the number of the used PMs while allocating the VMs; this leads to utilising fewer PMs. Thus, leading to more available spaces for the upcoming VMs and more options for the migrated ones.

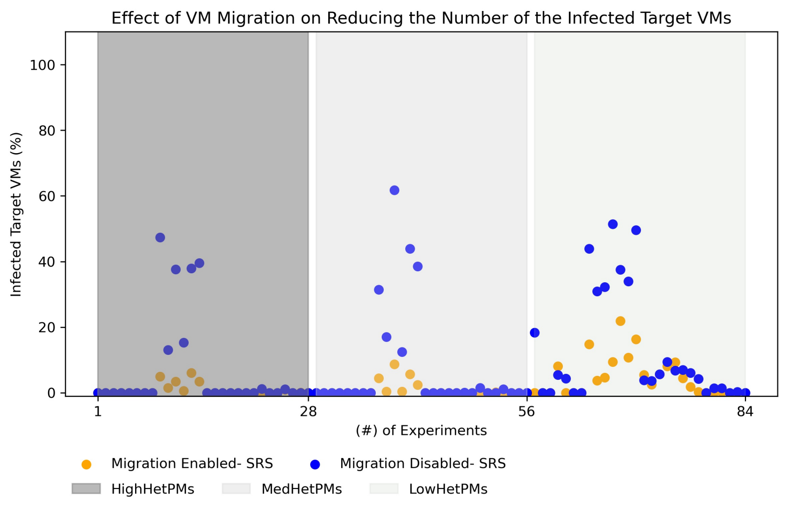


Figure 5.31: The Migration Effect on the Secure Allocation for SRS Algorithm.

PSSF Algorithm

In Figure 5.32, the PSSF benefits from the migration as most cases show low I_t after the migration is complete, which is an improvement. However, for some situations, the migration has the opposite effect as the I_t increases after the migration, indicating that the number of malicious VMs has increased on this parity PM. This case is the sequence of the VMs allocation and possibly the type of VMs arrival.

As we explained earlier, the sequence of VMs arrival is important to obtain a secure allocation for the upcoming VMs. Otherwise stated, the current allocation decision may affect the upcoming allocation negatively under some scenarios. For example, even if the resources are not limited, it is vital to allocate the VMs to ensure the current VMs are secure and the upcoming ones, which we tried to implement in our SS and SRS algorithms. However, PSSF produces many secure allocations, but the algorithm's sequence leads to high MNPS and I_t in most situations, even after the migration.

Overall, the allocation behaviour of the algorithms is crucial for producing secure VMs allocations and secure VMs migrations.

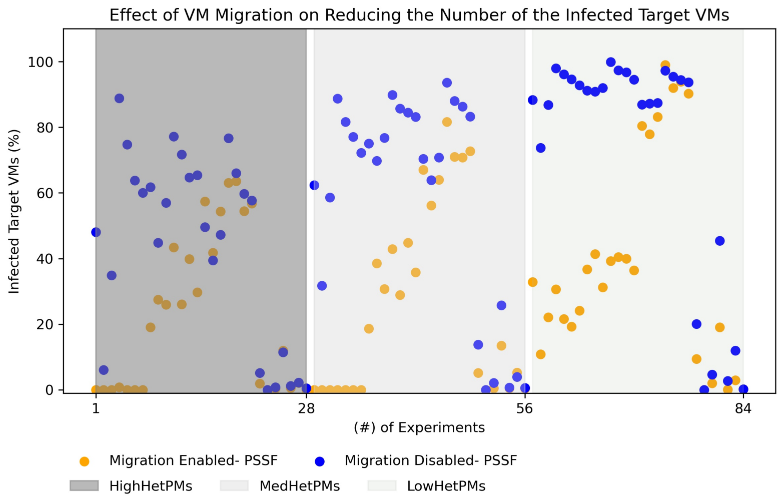


Figure 5.32: The Migration Effect on the Secure Allocation for PSSF Algorithm.

RAND Algorithm

In Figure 5.33, the benefits of migration for the Rand algorithm are significant as most of the infected target VMs end up having a secure allocation after the migration is completed. Moreover, most situations reduce the effect of malicious co-residency remarkably under all the PMs heterogeneous structures. The possible reason is that the Rand algorithm does not follow a specific behaviour in allocating the VMs, which benefits allocating the VMs in any suitable PM. This randomness behaviour can not be guaranteed at all times, but it can produce relatively positive results. For example, as we did in our SRS algorithm, it has the step of selecting the PM randomly, but at some point, this allocation may not be approved, as the algorithm produced another secure allocation due to the iteration process. This kind of behaviour could be implemented in the Rand algorithm to influence it to produce such a positive outcome, but it may suffer from performance issues, which we will discuss in the following section.

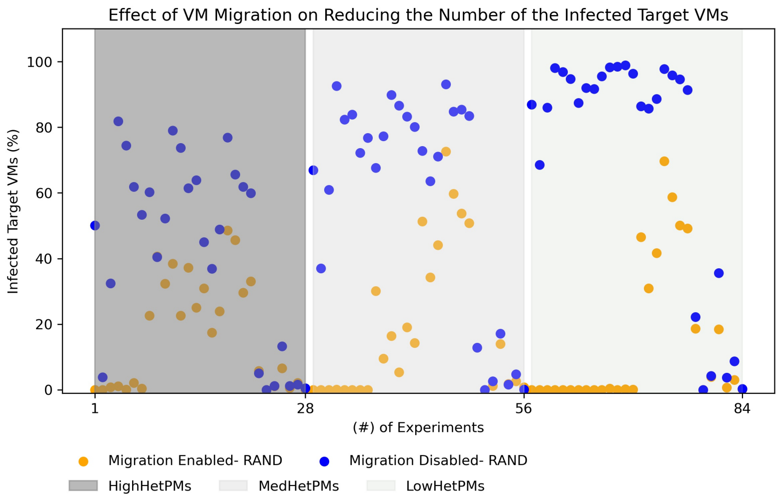


Figure 5.33: The Migration Effect on the Secure Allocation for Rand Algorithm.

5.7.5 Results of Algorithms Performance

This section will discuss the performance of our proposed algorithms, SS and SRS, and compare them with the other algorithms. The performance of the algorithms is measured by calculating the time that each algorithm takes to perform an allocation. For example, if we have the case where the number of VMs is 20, under different VMs arrival times and different PMs structures, the time for the simulation to execute will be calculated. This illustration, in Figure 1, aims to show the performance of the compared algorithms under the defined experimental setup. The average simulation time is only calculated when the VMs are heterogeneous, as the number of VMs processing is large, which causes a performance drawback for some of the algorithms.

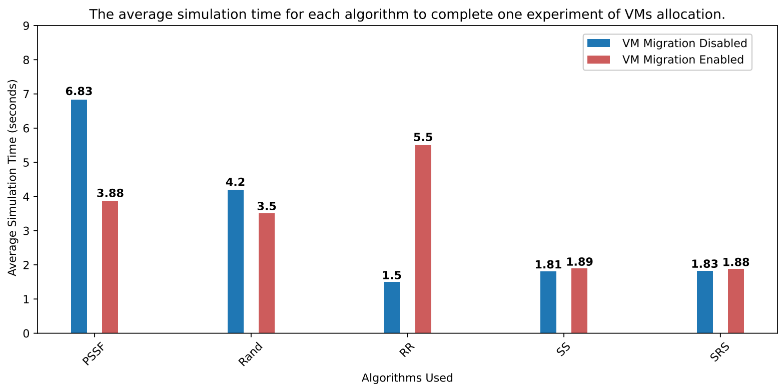


Figure 5.34: The Average Simulation Time for each Algorithm.

Intuitively, the process of the SRS algorithm led to believe that it might be consuming much time in allocating the VMs. However, its average time is considered one of the lowest compared to all the algorithms, even with migration enabled. This outcome is due to the stacking-based behaviour that the algorithm follows, which leads to stack the VMs during the allocation, user fewer PMs, thus leaving many options for the upcoming VMs to obtain a secure allocation.

The PSSF and Rand algorithm showed unexpected results, as the algorithms performed better when the VMs migration was enabled than when it was disabled, which means that the proposed VMs migration algorithm helped to reduce the simulation time needed to obtain a secure allocation by allocating and migrating the VMs in efficient time. However, the RR algorithm suffers the most when migration is enabled, which is the expected outcome. The RR algorithm spread the VMs across the available PMs, thus when the VMs migration trigger, it will be challenging for the migration algorithm to select VMs for migration and find a suitable PM for them.

Summary

This chapter introduces secure VM allocation algorithms, SS and SRS, that aimed to defend against SCAs in CCEs. The presented algorithms aim to find a secure allocation by preventing or reducing the 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 SS and SRS outperform other schemes in obtaining a secure VM allocation. In the following chapter, we will investigate further other factors that affect secure VM allocations by proposing a different data centre model and VMs allocation model that evolve graph-based allocation.

Chapter 6

Developing and Evaluating Graph-based Secure VMs Allocation Algorithms

Preface

In the previous chapter, we introduce SS and SRS algorithms that aim to obtain secure VMs allocation in CCEs. Moreover, we develop a VMs migration algorithm that aims to enhance the secure VMs allocation of the two algorithms.

This chapter extends our previous work to examine further factors and implementations of the VMs and PMs. As such, we introduce a VMs allocation model that includes the PMs architecture and VMs relation, which we view as a graph-based architecture. The PMs architecture, or data centre topology, is modelled as a Fat-Tree architecture to represent the relation between VMs, PMs and network components. Moreover, we consider the VMs graph a weighted undirected graph representing the relationships and interactions between VMs. Therefore, we presented two algorithms called Graph-based Secure Stacking (GbSS) and Graph-based Secure Random Stacking (GbSRS), which are an extension of the original two algorithms presented in the previous chapter; however, they allocate the VMs based on the graph-based model.

In addition, we presented a detailed evaluation of the proposed algorithms under different PMs and VMs structures and different allocation scenarios. Similar to the previous chapter, we study the effect of VM allocation behaviour on obtaining a secure allocation, and the behaviours are stacking, spreading, and random. Also, we investigate the factors affecting the outcome towards obtaining a secure allocation.

6.1 Motivation of this chapter

The previous chapter studies the VMs allocations where the VMs are independent of each other and the PMs are represented without considering their network connectivity. However, in this chapter, we consider the dependent VMs, represented as graphs, to examine the effect on the graph-based allocation. Otherwise stated, the VMs allocation process is represented as a graph showing their relation to load similarity and type. The load similarity of the VMs means the required resources of the connected VMs are similar, while the VMs type means the connected VMs are classified with the same VM type. The classification of the VMs type is explained in detail in Section 4.1.6.

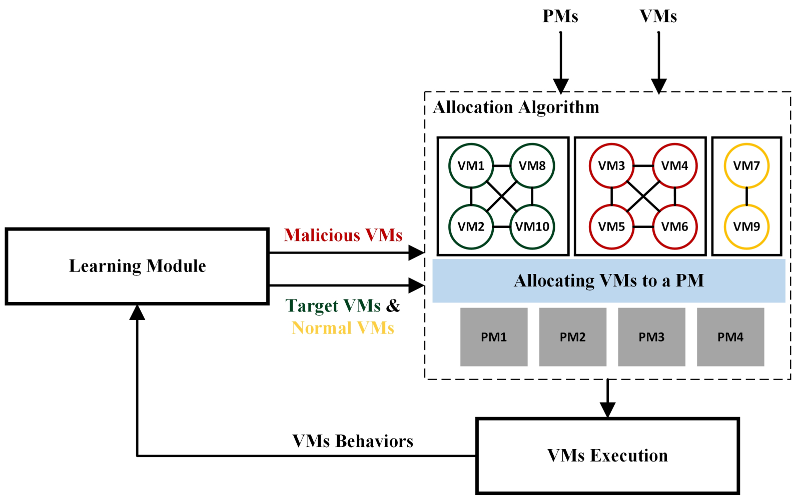


Figure 6.1: The Graph-based VMs behaviours detection framework.

As shown in Figure 6.1, we extended the same learning model presented in Section 4.1.6; however, the allocation architecture of unallocated VMs is modelled as a graph connecting them based on their behavioural activities and load correlation. In other words, the connection of the VMs is determined by their behaviour classification and their resource requests approximation. The allocation algorithm is then responsible for obtaining a secure VMs allocation while satisfying PMs resource constraints. The learning module will categorise the VMs into the malicious, target and normal VMs. An initially secure allocation may become compromised during allocation, leading to some VMs needing to be migrated. As such, a further objective is to reduce the number of VM migrations to reduce VM downtime. We will explain more about the graph model in the following section, Section 6.2.

6.2 Graph-Based Data Centre Model

This section presents the VM allocation model, which includes the PMs architecture and VMs relation, which we view as a graph-based architecture. First, the PMs architecture, or data centre topology, is modelled as a Fat-Tree architecture to represent the relation between VMs, PMs and network components [59]. Second, we consider the VMs graph a weighted undirected graph, representing the relationships and interactions between VMs.

6.2.1 System (Physical) Model

We assume that the CCEs is structured as a Fat-Tree architecture, with three layers of switches: a core switch S_c, a set of aggregation switches S_a and a set of edge switches S_e [38]. In Figure 6.2, the edge switch $s \in S_e$ connects the PMs directly and also connects to upper linked switches S_a. An aggregation switch $s \in S_a$ distributes the communication links from uplinks and downlinks, while the core switch S_c connects the CCEs to the outside world. The switches are the inside vertices of the tree, while the PMs are the tree's leaves.

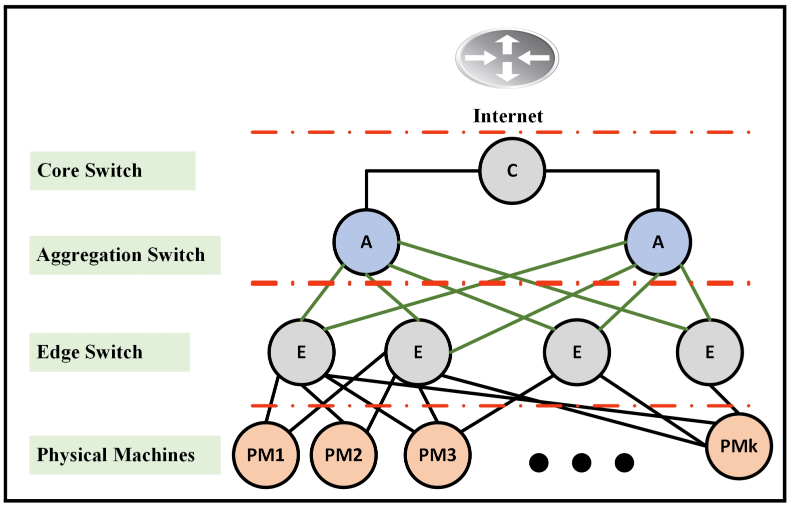


Figure 6.2: The Fat-Tree Architecture of System Model.

Similar to the original model in section 4.3.4, the CCE consists of a set P of $(k+1)$ PMs, and each $PM_i \in P, 1 \leq i \leq k$ has the same set of resources, but in varying quantities, e.g., one PM may have more CPU cores or more memory size than another, i.e., we assume the system to be heterogeneous. We denote by $R(PM_i)$, the amount of physical resources available on machine PM_i.

Additionally, there is a non-empty set V of VMs, and each $VM^j \in V$ may have a different set of resource type requirements, i.e., we assume the VMs to be heterogeneous, or non-heterogeneous. In the evaluation, we will examine both situations where the system resources are heterogeneous and non-heterogeneous. Furthermore, we assume that all the resource needs of a $VM^j \in V$ can be met by any PM in the CCE. We denote by $N(VM^j)$, the amount of resources needed by VM^j. We also assume that $\forall\ 1 \leq i, j \leq k, i \neq j, PM_i$ is able to communicate with PM_j in the CCE through the connected switches, i.e., the CCE is connected. We denote a path between the PMs, PM_i & PM_j, as a sequence of vertices and edges such that, for any subsequence $a \cdot (b, c) \cdot d$, $a = b$ and $c = d$. For example, from Figure 6.2, a path between PM_1 and PM_2 is $PM_1 \cdot (PM_1, e) \cdot e \cdot (e, PM_2) \cdot PM_2$.

6.2.2 VM Allocation Model

The VM model we assume in this chapter is that of a weighted undirected graph $G(V, E, L)$, where the V is the set of VMs, the set E of edges represents the relation between two VMs (VM^i, VM^j) and L is an edge labelling function that returns the label on an edge $e \in E$. We call such a model the VMs load correlation model (see Figure 6.3(C)). Such a model exists for each of the three types of VMs we assume, namely normal (set N), target (set T) and malicious (set M), under the same constraints presented in the equations Eq(4.1, 4.2).

VMs Type

Based on the learning model assumed in this work, as in Figure 6.1, a VM will be classified as either being a target, malicious or a normal VM. Starting with the set of VMs (see Figure 6.3 (A)), we initially assume that all VMs of the same type form a fully connected graph, called a VM Type graph, in that all VMs of the same type can potentially communicate with each other (see Figure 6.3 (B)).

VMs Load Similarity

As a system administrator may wish for related VMs to be co-located, we propose a VM load similarity metric to capture the similarity of two VMs in terms of resource requirements. A VM Type graph is thus refined into a VM load similarity graph by labelling the edge between VMs by the similarity metric between the two VMs (see Figure 6.3 (C)) and only keeping the edge with the highest value for each vertex.

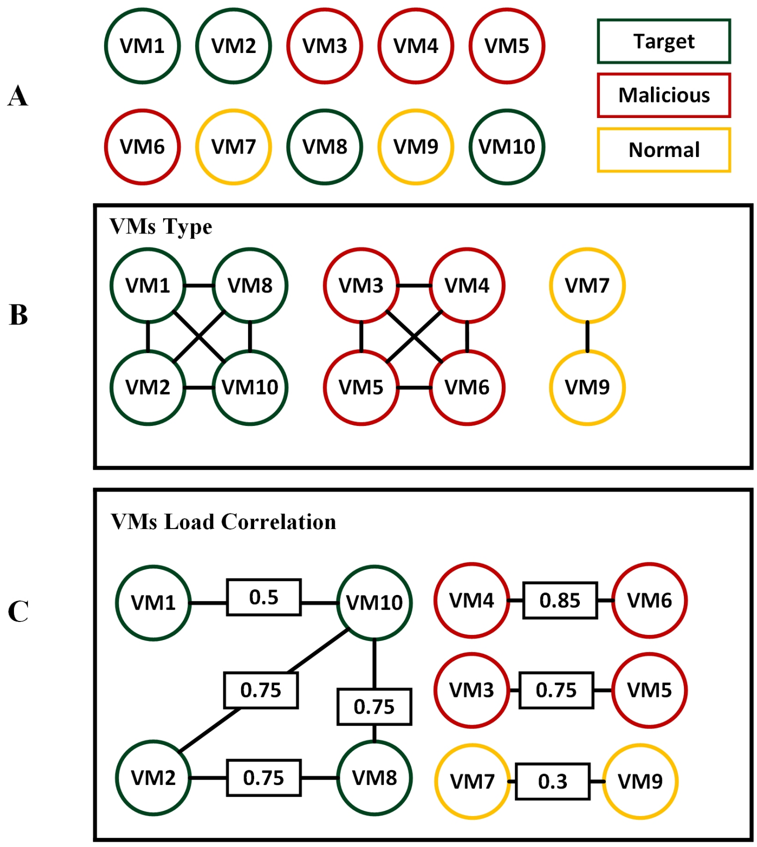


Figure 6.3: The Model for VMs Type and Load Correlation.

The load similarity of two VMs, VM^i and VM^j, denoted by vm_λ is computed as follows:

$$vm_\lambda(VM^i, VM^j) = \frac{N(VM^i)}{N(VM^j)} \tag{6.1}$$

This equation does not consider the effect of the dependency of VMs, for instance, the group of dependent VMs running the platform of applications and databases required to be allocated in adjacent PMs. It only considers the number of resources requested from each VM and models the connection accordingly. This solution could be specified to consider particular communication and computation rules in future work.

The VM Type graph $G_{\lambda,\lambda\in\{T,M,N\}} = (V, E)$ is converted into a VM load similarity graph $G'_{\lambda,\lambda\in\{T,M,N\}} = (V', E', L)$ as follows:

- $V = V'$
- $L(u, v) = vm_\lambda(u, v)$
- $\forall(u, v), (u, v') \in E, v \neq v' \cdot (u, v) \in E' \Leftrightarrow L(u, v) \geq L(u, v')$

As such, VMs with high load similarity may be co-located under appropriate resource availability.

Consequently, we model the VM allocation and migration similar to what is presented in Section 4.3.4, respecting the data centre structure presented in this model. Moreover, we added migration costs to perform the secure VM migration based on cost constraints. As stated, the VMs 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., the cost of $Move(A_i, A_{i+1})$ needs to be minimized. Therefore, we need to reduce the cost of a VM migration. As such, we define the $Cost_\Delta$ of VM migration $\Delta = Move(A_i, A_{i+1})$ as follows: Denoting by $MP(PM_i, PM_j)$, the shortest path between PM_i and PM_j in the CCE graph.

$$Cost_\Delta = \sum_{\forall v \in Move(A_i, A_{i+1}), A_i(v)=s \wedge A_{i+1}(v)=e} cost(MP(s, e)) \tag{6.2}$$

where $cost(MP(s, e))$ is the sum of delay of each switch and link on that path $MP(s, e)$.

6.3 Graph-Based Secure Stacking (GbSS) Algorithm

We aim to develop a secure allocation algorithm while minimising the cost of VM migrations under our assumed system model. Thus, we propose our graph-based security-aware heuristic, an extension of BPP, called GbSS, shown in Algorithm 6.1.

The input of the GbSS algorithm is the unallocated set of VMs, represented in a graph g, and the output is the secure allocation produced under the available resources, denoted as A. The GbSS performs three attempts to allocate a given VM, vm_i, starting from line 3, line 11 and finally in line 22.

Generally, the goal of the first attempt is to try to allocate the unallocated VM, vm_i, on the same PMs of their connected VMs in the load similarity graph. The load similarity graph identifies each VM connection with another VM by how similar they are, based on their required resources. Therefore, the algorithm will allocate the vm_i on the PM, pm_j that already hosts the connected VM. At some point in time, the connected VMs maybe still not be allocated yet, or not having sufficient resources to host the upcoming VM; thus, in this case, the algorithm will move on to the next attempt.

Therefore, if the first attempt fails to achieve an allocation, then the second attempt will start. It will try to allocate the vm_i to one of the PMs connected to the edge switch of the connected VMs. Alternatively stated, it will try to allocate the vm_i on one of the PMs on the higher level of the data centre topology, hence the PMs on the edge switch levels. The selection of the edge switch, thus the selection of the PMs, depends on the connected VMs already been allocated. If one of the connected VMs already allocated, the algorithm will obtain its PM and edge switch. We can obtain all the PMs connected to this edge from the edge switch, thus many PMs options for allocating the vm_i. The motivation of this step is to stack the connected VMs on the same network side of the data centre topology, thus keeping them secure by reducing the chance of malicious co-residency and reducing the VMs migration cost if triggered.

Afterwards, if the second attempt failed to obtain an allocation, the third attempt will start. This attempt is similar to the second one; however, instead of trying on the PMs on the edge switch, it will try the PMs on aggregation switch, thus more PMs options at this level than the edge switch level.

Algorithm 6.1: Graph-based Secure Stacking (GbSS) VMs Allocation

```
Input: $g = G(V, E, L)$, $P$: Set of PMs
Output: $A$: Secure Allocation
1  A ← ∅
2  $p \leftarrow null$
3  for $vm_i \in g$ do
       // first Attempt
4      $vm_\lambda \leftarrow getConnectedVM(vm_i)$
5      if $vm_\lambda.getPM() \neq null$ then
6          $pm_j \leftarrow vm_\lambda.getPM()$
7          if $(pm_j.suitablePM(vm_i))$ then
8              $A \leftarrow Assign(vm_i, pm_j)$
9              break
10         end
11     end
       // second Attempt
12     if $vm_i.getPM() = null$ then
13         $vm_\tau \leftarrow getAllocatedConnectedVM(vm_i)$
14         if $vm_\tau \neq null$ then
15             $edge_i \leftarrow vm_\tau.getEdgeSwitch()$
16             $pms \leftarrow edge_i.getPMsList()$
17             $sortedPMsList \leftarrow getSortedFRPMs(vm_i, pms)$
18             $p \leftarrow getFirstPM(sortedPMsList)$
19             $A \leftarrow Assign(vm_i, p)$
20             break
21         end
22     end
       // third Attempt
23     if $vm_i.getPM() = null$ then
24         rerunAgg() // repeat steps 11 - 19 on the aggregation switch
25         vmMigration(P)
26         rerun() // repeat steps 11 - 19 on edge and aggregation switches
27     end
       // last Attempt
28     if $vm_i.getPM() = null$ then
29         for $h$ in $P$ do
30             if $p.suitablePM(vm_i) = true$ then
31                 $A \leftarrow Assign(vm_i, h)$
32                 break
33             end
34         end
35     end
36 end
37 return $A$
```

Then, if these attempts also failed, the VMs migration will be triggered to restructure the current allocation and obtain a secure allocation for the vm_i. After the migration is completed, the second and third attempts will be repeated to allocate the vm_i. The current allocation has changed after the migration is completed; thus, there is a chance to obtain a secure allocation. At last, if all the attempts fail, then the algorithm will allocate the vm_i on one of the suitable PM in the data centre, regardless of any constraints.

The details of the three attempts and the VMs migration will be explained in detail in the following sections.

6.3.1 First Attempt

As stated, the first attempt aims to allocate the unallocated VMs on the same PMs as their connected VMs in the load similarity graph. This attempt will start in line 3, where the first unallocated VM arrived for allocation. In line 4, the function $getConnectedVM(vm_i)$ will be triggered to retrieve the VM with the highest load similarity of the connected VMs. As mentioned in Section 6.2, the VMs are initially connected based on the learning model's classification. Then, there is further sub-connection which identifies the relationship between each VMs based on the load similarity. As such, the function $getConnectedVM(vm_i)$ will return the VM with the highest load correlation, or similarity, among the connected VMs.

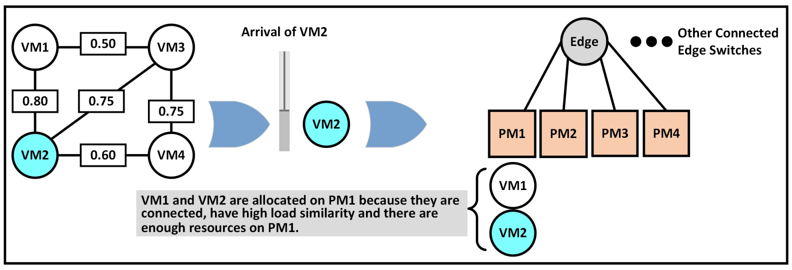

VM1
0.50
VM3
0.80
0.75
0.75
VM2
0.60
VM4
Arrival of VM2
VM2
Edge
Other Connected Edge Switches
PM1
PM2
PM3
PM4
VM1
VM2
VM1 and VM2 are allocated on PM1 because they are connected, have high load similarity and there are enough resources on PM1.

Figure 6.4: The First Attempt of GbSS Algorithm.

As shown in Figure 6.4, upon the arrival of the VM2, it has connections with VM1, VM3 and VM4, based on their type classifications. Furthermore, they have a sub-connection based on the load similarity, the value on the line between each vertex. Thus, based on the calculated value, the VM2 will be connected to VM1, thus allocating the VM2 on the same PM as VM1. The motivation behind this step is to keep the VMs allocation stacked and secure by reducing the chance of malicious co-residency. Moreover, in the model evaluation result, in Section 4.4.3, we establish that the optimal allocation tends to allocate the VMs in a stacking fashion by reducing the number of used PMs, and therefore, reduce the unnecessary VMs migration.

Therefore, in line 6, the selected PM will be the same PM that hosted the connected VM, VM1, in the example. Afterwards, if the selected PM has sufficient available resources, it will be selected and host the unallocated VM. The assignment function in line 8 is already explained in the previous chapter, in section 5.2.

6.3.2 Second Attempt

The second attempt will be triggered if the first attempt fails to allocate the unallocated VM. As stated, this attempt aims to allocate the unallocated VM to one of the PMs connected to the edge switch of the connected VMs. It starts at line 13; the function $getAllocatedConnectedVM(vm_i)$ is triggered to return the connected VMs already been allocated, regardless of their load similarity. Thus, this function will search for all the connected VMs, and return the already allocated one. This step aims to know the edge switch linked to the connected VMs, to retrieve all PMs connected to this edge switch. As such, in line 15, the function $getEdgeSwitch()$ will be triggered to return the edge switch of the connected VM. From this information, in line 16, the list of PMs linked to the selected edge switch will be retrieved and therefore considered for an allocation.

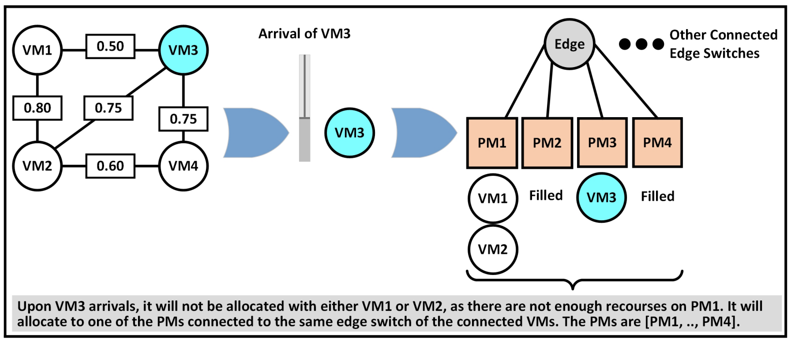
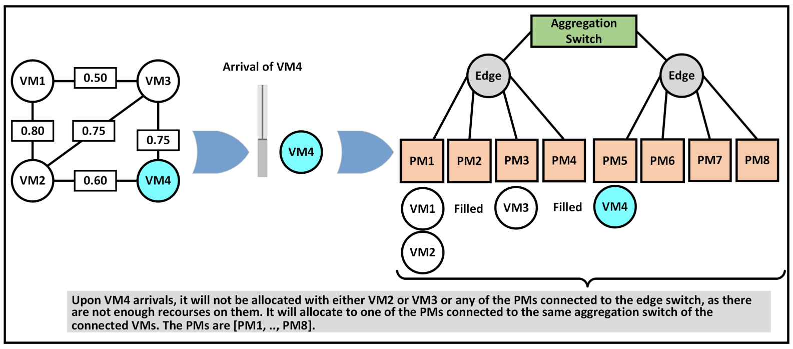

VM1
0.50
VM3
0.80
0.75
0.75
VM2
0.60
VM4
Arrival of VM3
VM3
Edge
Other Connected Edge Switches
PM1
PM2
PM3
PM4
VM1
Filled
VM3
Filled
VM2
Upon VM3 arrivals, it will not be allocated with either VM1 or VM2, as there are not enough recourses on PM1. It will allocate to one of the PMs connected to the same edge switch of the connected VMs. The PMs are [PM1, .., PM4].

Figure 6.5: The Second Attempt of GbSS Algorithm.

As illustrated in Figure 6.5, the previous steps are explained with an example. Upon VM3 arrivals, it will not be allocated with VM1 or VM2, as there are not enough resources on PM1. It will allocate to one of the PMs connected to the same edge switch of the connected VMs. The selected PMs for an allocation are PM1 to PM4. After this step, line 17, these PMs will be sorted according to their FR, to force the stacking behaviour during the allocation, as explained in detail in Section 5.2.1. Lastly, in line 19, one of the sorted PMs will be selected for an allocation if there are enough resources on the selected PM.

6.3.3 Third and Last Attempts

The last attempt, in line 22, follows the same step as the second one, except that the PMs selected for an allocation, in line 15, are retrieved from the aggregation switch level. Thus, the number of PMs options increases, and the chance for obtaining an allocation increases. For example, in Figure 6.6, upon VM4 arrivals, it will not be allocated with either VM2 or VM3 or any of the PMs connected to the edge switch, as there are not enough resources on them. Instead, it will be allocated to one of the PMs connected to the same aggregation switch of the connected VMs. The PMs at this level is ranging from PM1 to PM2.



Upon VM4 arrivals, it will not be allocated with either VM2 or VM3 or any of the PMs connected to the edge switch, as there are not enough recourses on them. It will allocate to one of the PMs connected to the same aggregation switch of the connected VMs. The PMs are [PM1, .., PM8].

Figure 6.6: The third Attempt of GbSS Algorithm.

If the above three attempts failed to obtain a secure allocation, the VMs migration would be triggered. The VMs migration aims to securely change the current allocation with less migration cost to avoid unnecessary migration. The details of the VMs migration will be explained in Section 6.5. Afterwards, the last step in line 29, and if the above attempts fail, the algorithm will attempt to allocate the unallocated VM to any suitable PM available in the data centre.

6.3.4 Time Complexity Analysis for GbSS Algorithm

This section will analyse the time complexity for the GbSS algorithm similarly to the analyses we perform in the previous chapter, in Section 5.2.3. The GbSS has four attempts to be completed; as such, we will calculate each attempts worst-case running time. Two main inputs affect the complexity: the VMs, denoted as N, and the PMs, denoted as M.

The time-consuming step in the first attempt is the function $getConnectedVM(vm_i)$, which is responsible for retrieving the VM with the highest load similarity of the connected VMs. As such, it will take $O(N)$ for this step. The other steps will take constant time as they either assignment or comparison steps. This attempt is nested in the loop responsible for allocating all the VMs; hence the big-O will be $O(N^2)$.

The second attempt has two major steps contributing to the running time. The first one is the $getAllocatedConnectedVM(vm_i)$ function, which takes $O(N)$ to retrieve the allocated VMs. The second one is in $getSortedFRPMs$ function, which takes $O(M + M * logM)$ as explained in Section 5.2.3. The big-O for this attempt is $O(N(N + M + M * logM))$.

The third attempt will repeat the previous one with adding the running time of the VMs migration function. As such, the $rerunAgg()$ function will take $O(N + M + M * logM)$, plus the VMs migration function, which will take $O(M + N * M)$ as explained in Section 5.2.3. In addition, the $rerun()$ function will take $O(2 * (N + M + M * logM))$, which will repeat the second attempt twice, on both edge and aggregation switch, after the VMs migration. Therefore, the big-O for this step is the summation of the mentioned running time, and for simplicity, it will take $O(N * (3N + 4M + NM + 3M * logM))$.

The last attempt will take $O(N)$ and $O(N^2)$ considering being nested in the loop of allocating all the VMs. As such, the time complexity of GbSS is the summation of the calculated running time, which is $O(N * (6N + 5M + N * M + 4M * logM))$ or $O(6N^2 + 5M * N + M * N^2 + 4M * logM)$.

6.4 Graph-Based Secure Random Stacking (GbSRS) Algorithm

The GbSRS algorithm process is similar to the original SRS algorithm presented in the previous chapter in Section 5.3. Thus, this section only explains the difference between the original SRS and the GbSRS. As stated, the general idea of the GbSRS is to produce as many secure allocations as possible within a given time limit, then checks if the generated allocations are secure. If the generated allocations reach the lowest level of malicious co-residency, this allocation is considered a final allocation. Figure 5.4, in Section 5.3, explains the process of GbSRS, which is the same as the SRS algorithm.

We only explain the significance of this algorithm, which is the *oneAllocation* function responsible for generating the secure VMs allocation. Like the GbSRS algorithm, the *oneAllocation* function performs three attempts to produce a secure allocation; however, the algorithm follows a stacking random behaviour in allocating the VMs. Unlike the GbSS, the GbSRS is a non-deterministic algorithm due to the randomness factor of selecting the VMs in the second and third attempts.

The input of the *oneAllocation* is the unallocated set of VMs, represented in graph g, and the output is the secure allocation produced under the available resources, stored in *oneAllocationList*. As mentioned, the *oneAllocation* performs three attempts to allocate a given VM, similar to the GbSS algorithm.

The difference is on the second attempt, starting in line 5, where the algorithm will try to allocate the vm_i on one of the PMs that shares the same edge switch of the connected VMs in the VM Type graph. In contrast to GbSS algorithm, in line 10, the list of PMs that shares the same edge switch of the connected VM is sent a *getHighestFRPMs* function. This function aims to select the most two highest fullness ratio PMs only out of the available PMs. The HFR function's outcome is then stored in a list of elected PMs, denoted as *ElectedPMsList*. The details of the HFR step is explained in Section 5.3.3. Then, in line 12, the algorithm randomly selected the PMs among the elected PMs and assigned them as candidates for allocation. The function *getRandomPM*(*ElectedPMsList*) is responsible for selecting a random PM among the list of the elected PMs and store as a candidate PM. Subsequently, if the candidate PM is suitable, it will be considered a host for the vm_i and added to the *oneAllocationList*, in line 14.

Afterwards, the third attempt will start if the second attempt failed to obtain an allocation and repeat the same process as the second attempt but with PMs linked to the same aggregation switch. This step will increase the number of available PMs to be selected for an allocation. If these attempts fail to obtain a secure VM allocation, the VM migration will be triggered to produce another allocation. In other words, the VM migration will change the current allocation to another, based on the constraint of the migration algorithm. Finally, if all the above attempts fail, including the VMs migration, the GbSRS will try to allocate the vm_i on any PMs in the PMs regardless of graph connection constraints.

Algorithm 6.2: Graph-Based Secure Random Stacking (GbSRS).

```
   Input: $g = G(V, E, L)$, $P$: Set of PMs
   // We only explaining what is different from the original SRS in
      Chapter 5
1  Function oneAllocation($g$,$P$):
2      $oneAllocationList \leftarrow \emptyset$
3      $ElectedPMsList \leftarrow \emptyset$
4      for $vm_i \in g$ do
           // First attempt is similar to GbSS algorithm
           // Second attempt
5          if $vm_i.getPM() = null$ then
6              $vm_\tau \leftarrow getAllocatedConnectedVM(vm_i)$
7              if $vm_\tau \neq null$ then
8                  $edge_i \leftarrow e_\tau.getEdgeSwitch()$
9                  $pms \leftarrow edge_i.getPMsList()$
10                 $ElectedPMsList.add(getHighestFRPMs(vm_i, pms))$
11                 for $Counter < ElectedPMsList.size()$ do
12                     $pm_i \leftarrow getRandomPM(ElectedPMsList)$
13                     if $(pm_i.suitablePM(vm_i))$ then
14                         $oneAllocationList.add(Assign(vm_i, pm_i))$
15                         break
16                     end
17                 end
18             end
19         end
           // Third attempt will be on aggregation switch
           // Last attempt steps is similar to GbSS algorithm
20         return oneAllocationList
21     end
```

6.4.1 Time Complexity Analysis for GbSRS Algorithm

To avoid repeating the same calculating steps, the time complexity analysis of GbSRS is similar to GbSS but with adding the effect of two parts. Three main inputs affect the complexity in SRS: the VMs, denoted as N, the PMs, denoted as M, and the time limit, denoted as K. The first one is the calculation of the FR function. The second one produces many allocations within a time limit; the running time is (one allocation) time multiplied with a time variable; we define it as K, as explained in Section 5.3.5. Therefore the time complexity of this algorithm is $O(K * (6N^2 + 6M * N + M * N^2 + 4M * logM))$ considering the nesting loops, the FR function calculation difference and the time limit for generating many allocations.

6.5 Graph-based VM Migration

The graph-based VMs migration is an extension of the algorithm presented in Section 5.4, but with adding constraints to control the migration number, hence the migration cost. This algorithm aims to minimize the number of VM migrations because migrating the VMs from one PM to another may result in some downtime.

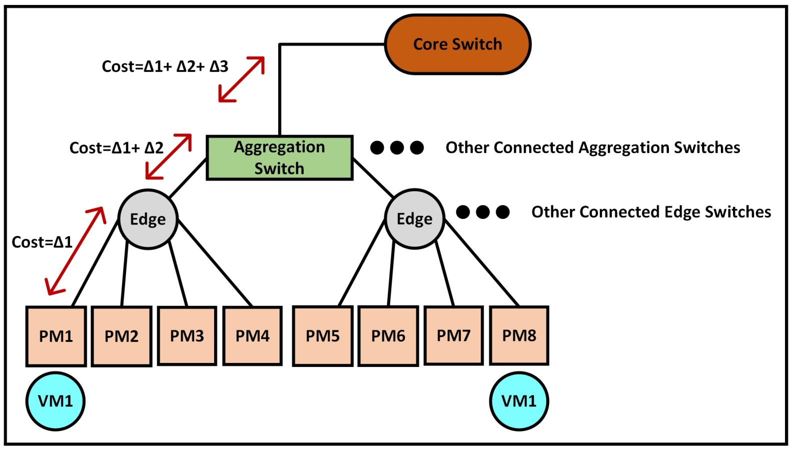


Figure 6.7: The cost of Graph-based VM Migration Algorithm.

As stated earlier, the VM migration Algorithm will be triggered if all three attempts fail to obtain a secure VM allocation. As such, after selecting the VMs to migrate, as described in Section 5.4, the algorithm finds a suitable PM for the selected VMs only if the *Cost* of migration to the selected PM is less than Δ. We define the cost Δ in Eq 6.2 as the cost of VM migration from one PM to another, and where it should not exceed the predefined threshold.

As shown in Figure 6.7, the VM migration for VM1 will start by migrating the VM into one of the PMs connected to the same edge of the source PM, which is PM1. If the attempt fails, the algorithm will migrate it to one of the PMs on the same aggregation switch until it reaches the core switch. If it reaches the core switch, the algorithm will try to obtain the migration with less cost. Alternatively stated, the PM8 will be selected as a destination PM for VM1 migration if the migration cost is less than the specified threshold. Thus, in this case, reducing the number of VMs migrated while reducing their migration cost.

6.6 Evaluation of Non-Heterogeneous Graph-based VMs Allocation

This section will evaluate the proposed algorithm and compare it with existing ones, following a different VMs allocation behaviour. As stated earlier, our proposed algorithm follows a stacking-based behaviour; therefore, we compare it with state-of-the-art algorithms that follow a spread and random behaviour. The experiment setup and evaluation process are similar to the section presented in the previous chapter, Section 5.6, but with a few additions regarding the data centre setup. The difference in the experimental setup will be described in the following section. Moreover, the evaluation continues to the one introduced in Section 5.6; thus, we will not repeat the results that have already been highlighted. Instead, the evaluation will be related to the allocation algorithms behaviours after the graph-based implementation.

Moreover, we utilise a simulation environment called a Network CloudSim, an open-source cloud simulation environment specified for network-related simulations in the cloud environment [36]. For example, it can simulate a VM allocation for defined network topology; as in our case, we design our simulation based on Fat-Tree architecture, as shown in Figure 6.2. Overall, it contributes to providing a valuable estimate of the expected outcome of certain situations and an initial understanding of the behaviour of the allocation process. Refer to **Appendix** B.3 for more information about NetCloudSim, its structure and allocation process.

Furthermore, we study the effect of different VM allocation behaviours on obtaining a secure VM allocation under certain situations. These situations are the VMs arrival times, the number of VMs for each type considered in this work, and the PMs' heterogeneity. The VMs allocation behaviours that we compare our algorithms, GbSS and GbSRS, with are the spreading behaviour, denoted as RR, the random behaviour as Rand and combines the spreading and random behaviours denoted as PSSF.

6.6.1 Experimental Setup Overview

The experimental setup is similar to the one described in the previous chapter, in Section 5.6.1, except for changes regarding the data centre topology and PMs distribution. The VMs and PMs number, VMs arrival time setup, the VMs type structure, the PMs heterogeneity levels, and the experiments' methodology are all the same.

The difference is on the data centre topology and the relation between PMs and network entities such as edge, aggregation and core switches. As such, we structured the linkage between PMs and switches as follow: A maximum of four PMs are connected to each edge switch. Each aggregation switch is connected by a maximum of two edges, which means that each aggregation switch can be connected by eight PMs maximum. Finally, all the aggregation switches are connected to one core switch. The number of PMs in each experiment will accommodate the required resources of the VMs. Moreover, we designed the available resources to be limited compare to the demanded resources. Hence, it makes it challenging for the algorithms to find a secure allocation.

6.6.2 Changes of Alternative Allocation Behaviours

Similarly to the previous chapter, in Section 5.5, we will describe the alternative VMs allocation behaviours and their changes that are considered in this work for comparison purposes. The three algorithms, PSSF, RR and Rand, will follow their behaviour in this section but with changes on which level these algorithms will be applied. As we described on the GbSS and GbSRS algorithms, we have introduced an allocation on three levels, the edge, the aggregation and the core switches. However, we cannot apply the same steps to the alternative algorithms as they will lose their allocation behaviour, which will affect the overall result of the comparison. We only started the allocation process of the three alternative algorithms on the aggregation switch, where there are many PMs to select for allocation. Moreover, the high number of PMs on the aggregation switch will maintain the allocation behaviour of the alternative algorithms.

6.6.3 Results of Malicious Co-residency under Limited Resources Availability

The section will discuss the effect on the M_{pms} when the resources are limited, which means when the number of VMs equal 120 VMs. The M_{pms}, as stated in Section 4.4.3, Eq. (4.15), is the percentage of the infected PMs out of the used ones. The discussion will include examining the VMs arrival times, PMs heterogeneity level and the VMs type numbers.

Malicious Co-residency for Group VMs Arrival under Limited Resources

To avoid repeating some of the same general outcomes, we only show one type of group VMs arrival time, GTNM.

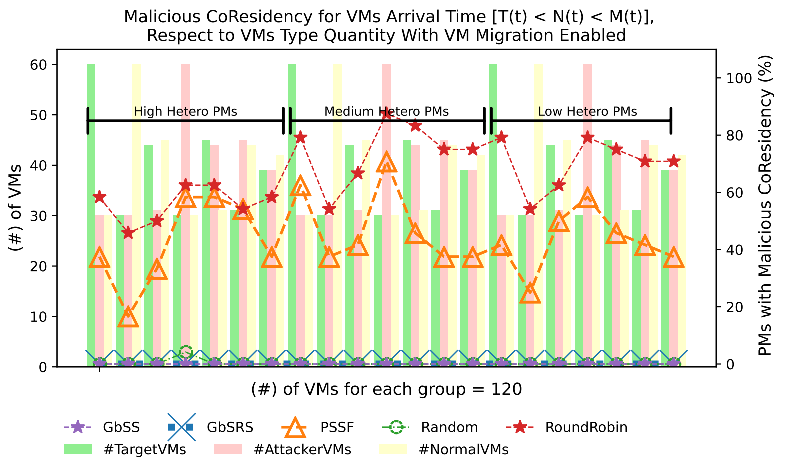


Figure 6.8: Malicious Co-residency under GTNM Arrival Time, When Available Resources Limited.

In general, as shown in Figures 6.8, algorithms GbSS, GbSRS and Rand perform better than the PSSF and RR in most situations. This initial observation is an indication that the spreading behaviour allocation often leads to higher levels of malicious co-residency than the stacking or the random behaviour.

Arguably, the Rand algorithm will always produce a different allocation in each simulation run due to its random behaviour. However, the M_{pms} are lower than anticipated in most situations, which is good for this behaviour. The possible reason for this outcome is the data centre design for the experiment and how the VMs are allocated. As mentioned earlier, all the algorithms, except GbSS and GbSRS, start from the third attempt on the aggregation switch. In other words, each VM allocation has a limited number of PMs on each aggregation switch. This limitation, alongside the predefined VMs type from the learning model, influences the Rand algorithm to behave partially in a stacking-based manner while randomly selecting from the PMs available in each aggregation switch. If the Rand algorithm has a larger amount of PMs to select from, it could perform differently.

Moreover, the M_{pms} of the spreading behaviour algorithms, RR and PSSF, are often higher, even if the data centre design yields many PMs available. Because the VMs spreading across the entire available PMs, and the chance for a malicious user to obtain a malicious co-residency is relatively higher. The more the malicious user has many VMs, the more the chance of malicious co-residency occurs. In other cases, the number and spreading of target VMs help increase the M_{pms}, even if the malicious user has fewer available VMs than other types.

In addition, our proposed algorithms, GbSS and GbSRS, are performing well along with the Rand algorithm. Definitely, with more available PMs, the M_{pms} will be lower, and while with limited available PMs, the M_{pms} could perform worse in some situations. However, the lower M_{pms} is due to the stacking behaviour conducted during the allocation, not only because of the data centre design. To put it differently, regardless of the available PMs, either higher or lower, the algorithm's behaviour leads to stack the connected VMs as much as possible while leaving available PMs freely for future allocation.

Even if the malicious user can increase the VMs amount, it will be challenging to achieve malicious co-residency.

Furthermore, no clear patterns link the number of each VMs type with the M_{pms} for GbSS, GbSRS and Rand algorithms. However, in RR and PSSF, the spike of the amount malicious or target VMs could potentially lead to higher M_{pms}, in some cases, the spike of both together. Because it occupied the available PMs, depending on the time of VMs arrival, that potentially needed for obtaining a secure allocation.

Overall, in RR and PSSF, the M_{pms} is relatively higher when the majority of either the malicious or target VM arrive firstly. As we stated previously, the reason is that the VMs, for example, the malicious ones, are initially spread across the available PM. Then, after some time, when the target VMs arrive, fewer spaces will be available for secure allocation. While our algorithm, which will stack the VMs to allow more spaces for future VMs, thus, reducing the M_{pms}.

Malicious Co-residency for Single VMs Arrival under Limited Resources

Overall, as shown in Figure 6.9, unlike the group VMs arrival where there is a clear distinction between the VMs allocation outcomes, the single VMs arrival tends to make the allocation behaviour relatively similar for all algorithms. However, the outcome is much better for the spreading-based allocation behaviours algorithms, PSSF and RR. In comparison, it worsens for the stacking and random based allocation behaviours. A possible reason for this outcome is that allocating the VMs, according to their types, when they arrive separately at a different time, is easier for an allocation to obtain the secure allocation. In contrast to the group VMs arrival, it will be difficult to obtain a secure allocation because the available PMs options that lead to secure allocation are lower, even for the staking-based algorithms for some cases.

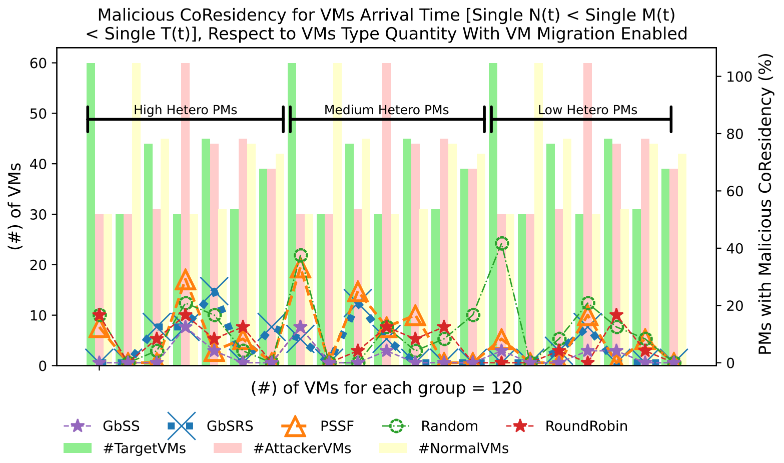


Figure 6.9: Malicious Co-residency under SNMT Arrival Time, When Available Resources Limited.

Moreover, the M_{pms} is better in the low heterogeneous structure than the high and medium structures for most cases. The reason is that the high and medium structures PMs are filled quicker than the low PM structure, which leaves fewer options for secure allocation or migration.

Furthermore, this outcome is affected by the VMs migration constraints, which initially prevent VMs from migrating to remote PMs where the cost exceeds the specified threshold. This constraint affects the VMs arriving later in the sequence of the unallocated VMs as it prevents them from having more options for secure allocation with their connected VMs. For example, if connected VMs allocated under the same aggregation switch PMs, a VM migration is triggered. Then, they will be migrated within this group of PMs under the same aggregation switch, if there are free or available PMs options, which does initially. This behaviour is good for keeping fewer PMs utilised to keep the power utilisation to a minimum and reduce unnecessary migrations. However, with the arrival of more VMs, the need to allocate them to remote PMs with disconnected VMs increases, as the options to allocate them with their connected VMs decreases. This outcome leads to more M_{pms} than usual and more unnecessary VMs migrations.

Malicious Co-residency for Mixed VMs Arrival under Limited Resources

Generally, in Figure 6.10, most of the reasons behind the allocation outcomes have been discussed. Specifically, the RR, PSSF and Rand have the worst cases of M_{pms}, and the PSSF algorithm performs better in most cases. The Rand algorithm outcome is worse in the mixed arrivals compared to the group VMs arrivals and similar to the single VMs arrival. The stacking-based algorithms, GbSS and GbSRS, showed the best outcome among the compared algorithms. The GbSS and GbSRS depend on allocating the VMs on the utilised PMs rather than spreading the VMs across the PMs or randomly allocating them. Thus, this behaviour offers many options for the upcoming VMs to allocate securely under different arrivals structures.

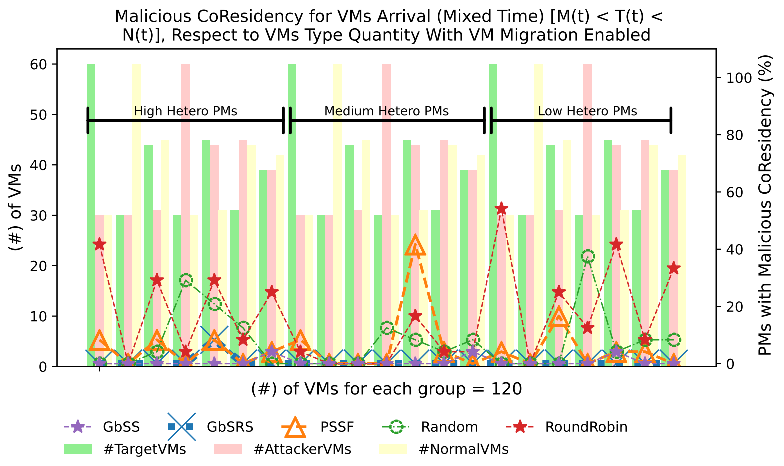


Figure 6.10: Malicious Co-residency under Mixed MTN Arrival Time, When Available Resources Limited.

6.6.4 Results of Graph-based 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 (Mig_{vms}) is an indication of the processing needed, by an algorithm, to produce a secure allocation, as stated in Section 5.6.4, Eq. (5.2). We are only showing one situation per VMs arrival type to void repeating similar situations results.

VMs Migrations for Group VMs Arrival

Overall, in Figure 6.11, RR, PSSF, and Rand algorithms have a higher Mig_{vms} compared to our proposed algorithms. The migration Mig_{vms} reflect the amount of processing needed to obtain a secure allocation for each algorithm. It also reflects the cost percentage for each algorithm, where it is clear that allocating the VMs in a stacking-based manner leads to a lower cost of VMs migration.

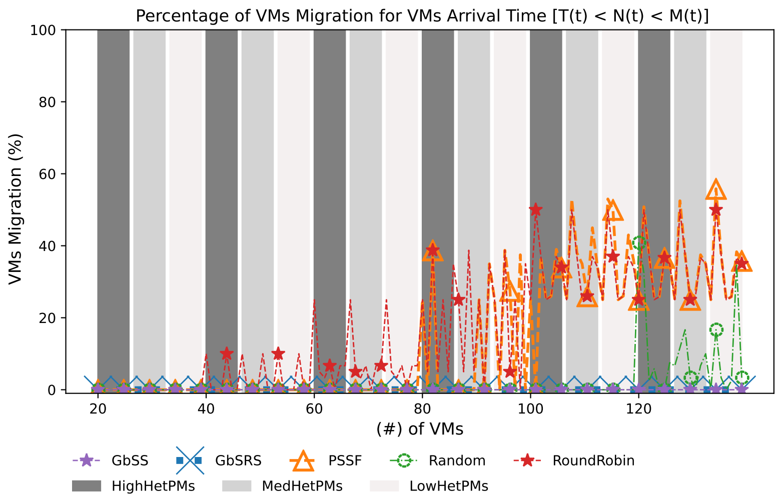


Figure 6.11: VMs Migration under GTNM Arrival Time, When Available Resources Limited.

Therefore, GbSS and GbSRS perform better than other algorithms due to allocating the VM using the fullness ratio function. This stacking-based behaviour produces a fitted allocation for each VM, thus minimising the need for migration and VMs migration cost. On the other hand, PSSF and RR algorithms show a similar behaviour because they share similar allocation behaviour, spreading the VMs. This leading to a significant amount of VMs migration, hence, a high cost of migrating the VMs. In contrast to the Rand, it shows an irregular pattern of Mig_{vms} for the situation due to its randomness behaviour. Nevertheless, it has a high cost for VMs migration in most cases examined, similar to RR and PSSF.

VMs Migrations for Single and Mixed VMs Arrival

This part will show the results of the two single and mixed VMs arrivals as they exhibit similar outcomes. Overall, in Figure 6.12, the Mig_{vms} is expected to be high when the resources start limiting, at the 120 VMs case, as it will be challenging for the algorithms to obtain secure allocations without triggering the migration upon the arrival of VMs in a single order. In specific cases, the PSSF benefits from the migration as it leads to more secure allocations than the RR and Rand algorithms, as their migration did not contribute positively in these cases. Our algorithms, GbSS and GbSRS, have high Mig_{vms} in single and mixed VMs arrival compared to the group VMs arrival, which indicates the challenge of obtaining secure allocations under these two cases.

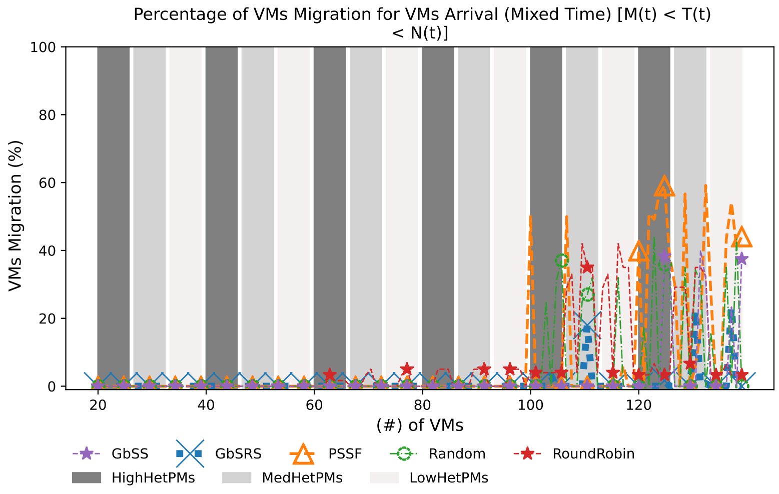


Figure 6.12: VMs Migration under Mixed MTN Arrival Time, When Available Resources Limited.

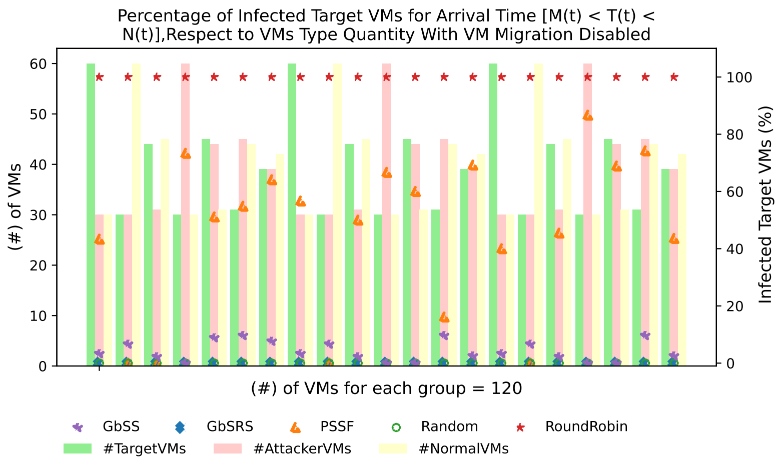
6.6.5 Results of Graph-based VMs Migration Effect

Similar to Section 5.6.5, this section aims to evaluate the migration effect of the migration algorithm of all the compared algorithms under different arrivals times. As such, we calculated the percentage of infected target VMs compared to the total target VMs available for an allocation, denoted as I_t in Eq. (5.3).

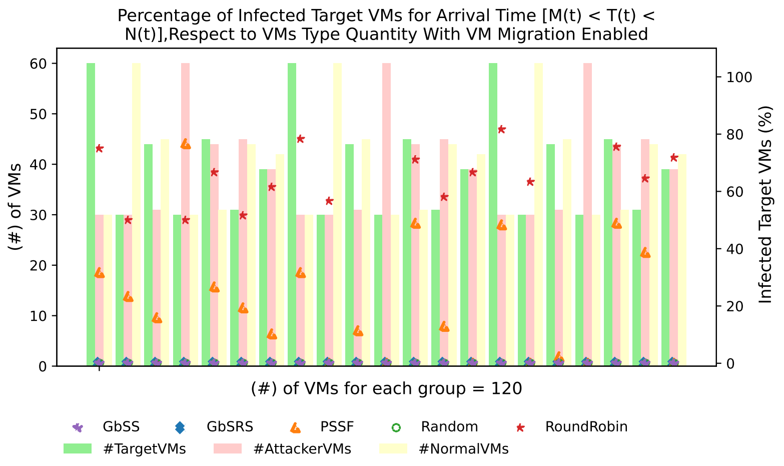
The following parts aim to examine the effect on preventing the target VMs from being co-located with the malicious VMs, thus enhancing the overall secure allocation.

VMs Arrival Group (GMTN)

This part will only show and discuss the GMTN arrival time situation, as the other group VMs arrivals show a similar pattern. Overall, the RR has the most benefits from the VMs migration among the other algorithms, while the PSSF shows a rise of I_t in some situations. The GbSS, GbSRS and Rand have also benefited from the VMs migration to reduce the I_t significantly.



(a)



(b)

Figure 6.13: The Migration Effect on the Secure Allocation under GMTN Arrival Time.

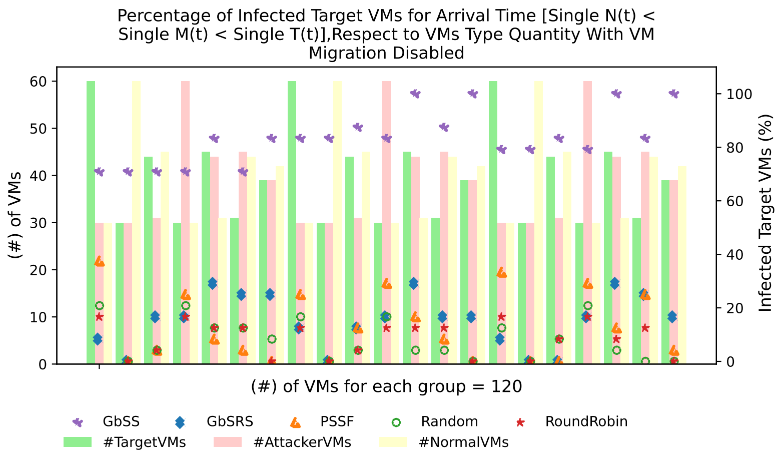
As shown in Figure 6.13, the most significant observation at this part is the behaviour of the Rand algorithm. It shows low I_t when the VMs migration is disabled and enabled, respectively. This outcome leads to conclude that when utilising the Rand algorithm for VMs allocation, controlling the number of available PMs during the allocation yields to reducing the malicious co-residency, hence secure an allocation. Otherwise stated, in the graph-based model, we constraint the algorithms to have fewer PMs available at the aggregation switch level, which influences the Rand algorithm to select from these available PMs initially. This selection of a fewer number of PMs contributes positively to the result of the lower I_t.

In comparison, the PSSF did not show positive results where the I_t increases after the migration. The reasons for this outcome are the algorithm's spreading behaviour, the random selection of PMs that the algorithm follows, and the migration process. The spreading of VMs leads to spread the VMs across the available PMs, resulting in difficulties obtaining secure allocations for the upcoming VMs. The random selection often leads to unexpected outcomes at every run, leading to different results from the one shown, worse or better.

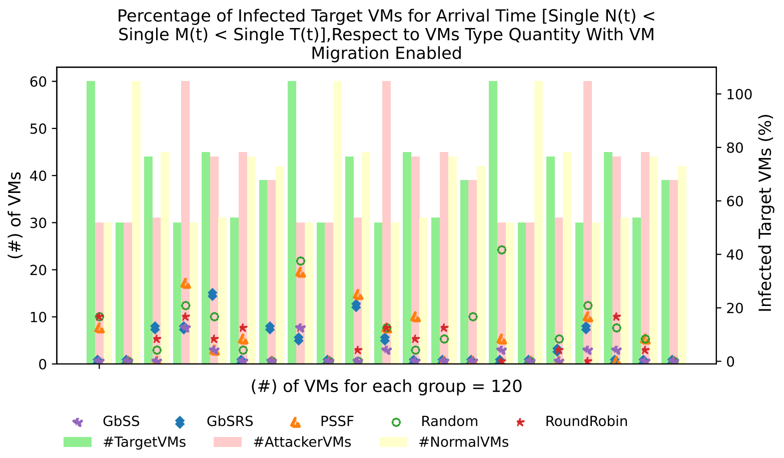
Single and Mixed VMs arrivals

In this part, we are only showing the result of single VMs arrival in Figure 6.14, illustrating a similar effect as the mixed VMs arrivals.

Generally, a significant positive impact is shown clearly to our algorithm GbSS where the I_t dropped sharply after the VMs migration is enabled, resulting in reducing the malicious co-residency. In other words, the effect of VMs migration can be effective when the structure of the VMs arriving tends to be singular than group based on the GbSS algorithm. The GbSRS has better results than the GbSS before the migration is enabled, and the enhancement after the migration is minor. The other algorithms are generally better in obtaining secure allocations; however, the migration impact is not significant in most cases.



(a)



(b)

Figure 6.14: The Migration Effect on the Secure Allocation under SNMT Arrival Time.

6.6.6 Results of Graph-based PMs Usage

This section present the result of PMs utilisation compared to the total available PMs, similar to the one presented in Section 4.4.3, in Eq. (4.16). As such, this section aims to examine the PMs utilisation ($Usage_{pms}$), during the VMs allocations for all the compared algorithms.

We are only showing one situation among the examined ones, as mostly all the other situations showing a similar pattern in the ($Usage_{pms}$ results. As illustrated in Figure 6.15, the utilisation of the GbSS and GbSRS algorithms are the lowest among the compared algorithms. The most significant part of this evaluation is the outcome of the Rand algorithm, which shows unexpected lower $Usage_{pms}$ results. These results are affected by the design of the proposed data centre, which influenced the Rand algorithm to utilise fewer PMs during the allocation and migration. In other words, the structure of the PMs with their connectivity with the switches and the constraint of the VMs migration to remote PMs led to having fewer options of PMs.

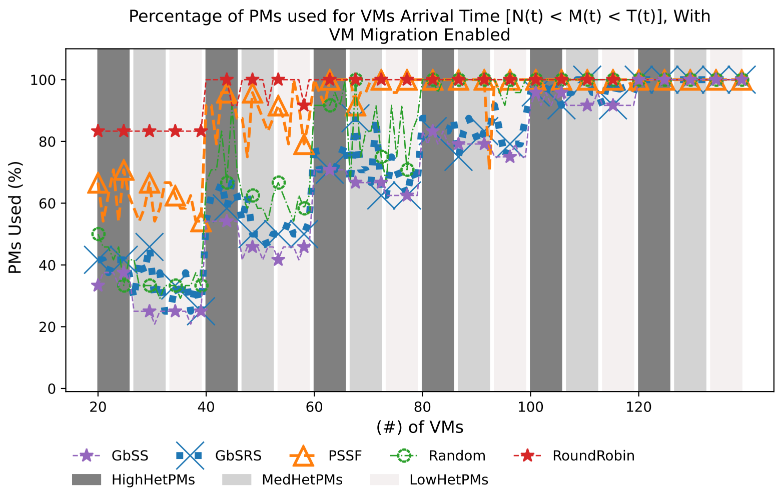


Figure 6.15: VMs Migration under GNMT Arrival Time, When Available Resources Limited.

6.6.7 Effect of Data Centre Architecture

This section will highlight some of the effects on VMs allocation when implementing the Fat-tree architecture. As we stated in Section 6.1, the motivation of this chapter is to examine the effect on the graph-based allocation. Hence, we are not expecting a benefit from utilising different cloud architecture; however, we expect an effect that needs to be shown and examined. Therefore, we will compare the results of malicious co-residency, VMs migration counts, VMs migration effect and PMs usage for SS and SRS against GbSS and GbSRS algorithms.

From the malicious co-residency perspective, the group VMs arrival, in Figures 5.14 and 6.8, show no effect between them as the outcome is similar. However, the single VMs arrival, in Figures 5.15 and 6.9, showed a worse outcome because of the effect of the data centre architecture. This outcome is affected by the migration constraints imposed on the graph-based allocation, which controls the VMs migration according to the cost threshold. It reduces the availability for allocating the upcoming VMs on secure PMs because the migration effect that is supposed to allow for more secure options is reduced. The same effect applies to the mixed VMs arrival because they show a higher M_{pms} outcome.

In contrast, the VMs migration count improves the group VMs arrival, specifically between Figures 5.17c and 6.11. Even though controlling the VMs migration leads to a higher malicious co-residency for some cases, it showed a better outcome under the group VMs arrival for both malicious co-residency and VMs migration. Many factors contribute to improving this outcome, the FR function, the time of the VMs arrival, the migration of specific VM at a particular time and the availability of resources. On the other hand, the single VMs arrival is getting worse in Figure 6.12 than 6.12, which is an expected outcome due to the migration threshold, leading to fewer secure PMs migrating, thus leading to more VMs triggering. This effect did not appear when the VMs arrival grouped as the distribution of the VMs in the group allows the algorithms to find secure allocation easier than the single VMs arrival. Lastly, the PMs usage shows similar results between the two data centre architectures because the algorithms utilise the same behaviour of stacking VMs.

6.7 Evaluation of Heterogeneous Graph-based VMs Allocation

This section will evaluate the proposed algorithms under heterogeneous VMs structure, similar to the evaluation proposed in Section 5.7. Unlike the experimental setup presented in the previous section, Section 6.6, this section aims to utilise a simulation environment and actual data centre traces obtained from a real-world cloud data centre provider. As such, we utilised the Azure VMs traces published by the Microsoft team, which contains the VMs workload of Azure [22]. These VMs traces will be loaded by Network CloudSim and created on the simulation environment to be simulated under different conditions and structures. More information about Azure traces is described in detail in **Appendix** C.

6.7.1 Experimental Setup Overview

The experimental setup and process for this section are similar to the one presented in the previous section, Section 6.6, except for changing the structure of VMs, and PMs. The VMs structure of the Azure data set; as such, the VMs number and structure will be changing. Moreover, therefore, the number and structure of PMs will be changing. The detail of this structure is explained in Section 5.7.1.

6.7.2 Results of Malicious Co-residency For Heterogeneous VMs under Limited Available Resources

Similar to Section 6.6.3, we will discuss the effect on the M_{pms} when the resources are limited under heterogeneous VMs.

Malicious Co-residency for Heterogeneous Group VMs Arrival

Overall, in Figure 6.16, the GbSS, Gb SRS, and Rand algorithms have lower M_{pms} than the PSSF and RR algorithms in most cases. Also, the M_{pms} in low PMs heterogeneous structure is worst than the other PMs structure for all the compared algorithms. This outcome is because the VMs resources are structured as heterogeneous, which means they have highly diverse resources demands, which affect the allocation of resources. In other words, a group of VMs arriving with high demands-resources leads to utilising a higher number of PMs, potentially leading to unfit VMs allocation resulting in resource wastage. For example, in the case of the spreading behaviour algorithms and low heterogeneous PMs, if a VM with 8 GB of RAM is allocated to a PM with 12 GB of RAM available, it leaves 4 GB of RAM on this PM. Then, the next VM arrives and demands more than 4 GB RAM; then, it will be allocated to the next PM with possibly high available PM, leading to more resources wastage and unfitted allocation. However, if the PMs are structured in a more diverse method, as described in high heterogeneous in Section 5.7.1, then the chance of having more fitted allocation is higher, leading to potentially lower M_{pms}.

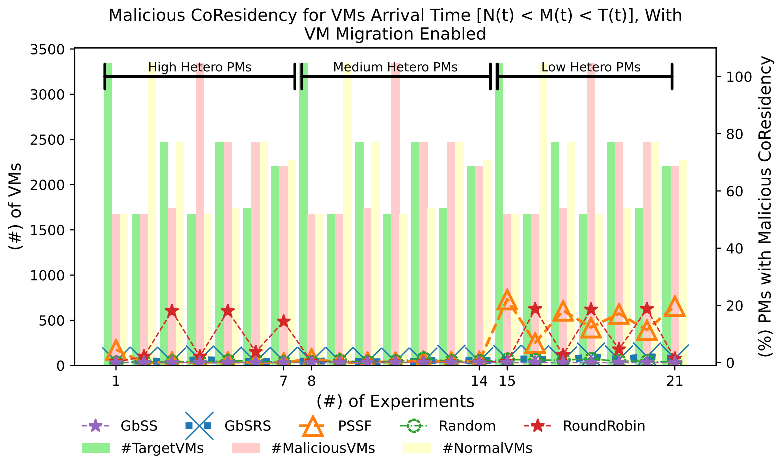


Figure 6.16: Malicious Co-residency under GNMT Arrival Time for Heterogeneous VMs.

Specifically, the Rand algorithm continues to have better results due to the low number of PMs options during the allocation. This constraint of PMs number with the predefined VMs type from the learning model influences the Rand algorithm to behave partially in a stacking-based manner while randomly selecting from the PMs available in each aggregation switch, which produces a better outcome for all shown cases. Moreover, the RR and PSSF are the worst among the compared algorithms because the VMs is spreading across the entire available PMs, and the chance for a malicious user to obtain a malicious co-residency is high. Furthermore, our algorithms have lower M_{pms} due to the stacking behaviour conducted during the allocation, which leads to perfect match allocations, i.e., fitted allocations.

Malicious Co-residency for Heterogeneous Single VMs Arrival

Generally, in Figure 6.17, the single VMs arrivals lead to a better outcome of M_{pms} than other VMs arrival structures. Moreover, the medium and high structure generally yields lower M_{pms} for some of the algorithms. As stated, a possible reason for having either high or low M_{pms} on the three PM structure is that each algorithm fits a VM into a PM. Unfitted (VM to PM) allocation may skip a perfect allocation due to spreading or random behaviour of the algorithm, which leads to an increase in the M_{pms}. To put it differently, our algorithms GbSS and GbSRS use a fullness ratio function that calculates the available space of PM compares to the demanded resources from the VM. Thus, it is a calculated relation between each VM with the available PMs.

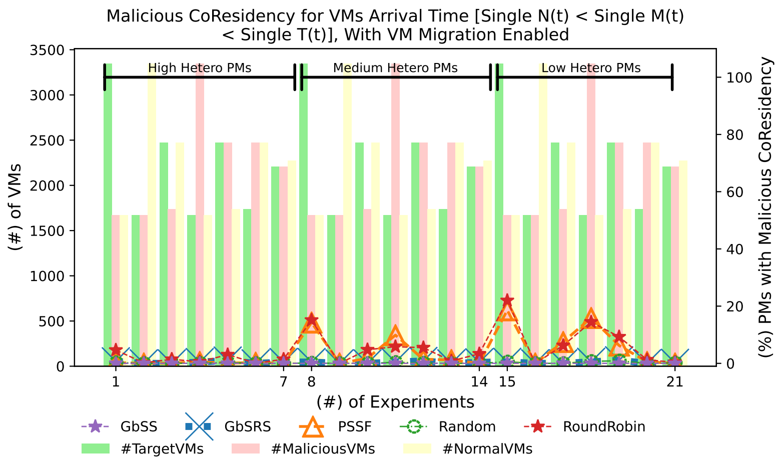


Figure 6.17: Malicious Co-residency under SNMT Arrival Time for Heterogeneous VMs.

Furthermore, the RR and PSSF algorithms having similar and high M_{pms}. The reason is that the VMs, for example, the malicious ones, are initially spread across the available PM. Then, after some time, when the target VMs arrive, fewer spaces will be available for secure allocation. While our algorithms, which will stack the VMs to allow more spaces for future VMs, thus, reducing the M_{pms}.

Malicious Co-residency for Heterogeneous Mixed VMs Arrival

Overall, in Figure 6.18, the PSSF algorithm is worse when most target VMs arrive right after the malicious ones for two reasons. First, the malicious VMs will spread across the available VMs, and the seconds; the migration options are only limited to the malicious VMs. As we explained in the migration algorithm, selecting VMs to migrate is the only constraint to the malicious and normal VMs to avoid interruption to the target VMs. Moreover, the RR suffers from the same outcome on the high heterogeneous PMs for the same reasons.

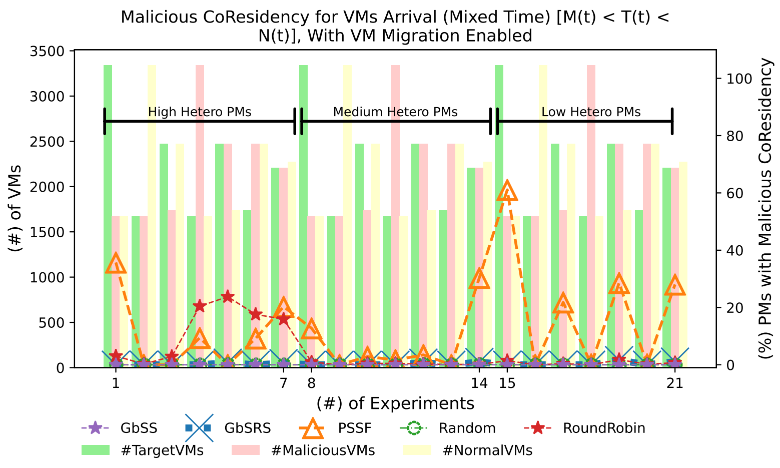


Figure 6.18: Malicious Co-residency under Mixed MTN Arrival Time for Heterogeneous VMs.

6.7.3 Results of Graph-based Heterogeneous VMs Migrations

In this part,in Figure 6.19 we are only showing the VMs migration for the mixed VMs arrivals as it shows the most interesting results, and the other VMs arrival showing similar patterns or results already been discussed.

In general, RR, PSSF, and Rand algorithms have a higher Mig_{vms} than our proposed algorithm. The migration Mig_{vms} reflects the cost percentage for each algorithm, where it is clear that allocating the VMs in a stacking-based manner leads to a lower cost of VMs migration. In other words, due to the stacking-based allocation of our algorithms, GbSS and GbSRS, the processing needed and the cost of migration to obtain a secure allocation is far less than the other allocation behaviours. Therefore, our algorithms performing better than other algorithms due to allocation that depends on the fullness ratio calculation—leading to a fitted allocation for each VM, thus minimising the need for migration and VMs migration cost. In contrast, the other algorithms show a high cost of migrating the VMs, which indicates the unfitted allocation performed initially by these algorithms.

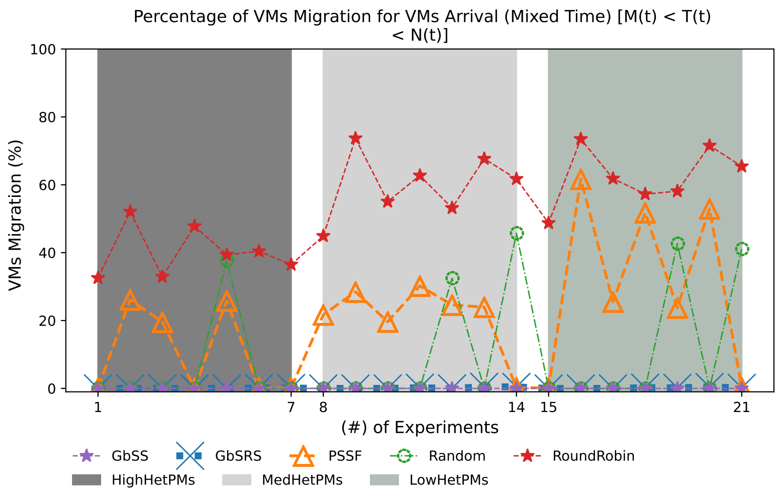


Figure 6.19: VMs Migration under Mixed MTN Arrival Time for Heterogeneous VMs.

Summary

This chapter introduces graph-based secure VM allocation algorithms, GbSS and GbSRS, extending the original proposed SS and SRS algorithms. The presented algorithms aim to find a secure allocation by preventing or reducing the co-residency of a target VM with a malicious VM. Our results show that the data centre topology impacted the overall outcomes of malicious co-residency for the compared algorithms. Moreover, the VM arrival times have a significant impact on obtaining a secure allocation. Overall, the algorithms that follow a stacking behaviour in VM allocations are more likely to return secure allocations than spreading or random-based algorithms.

Chapter 7

Conclusions and Future Work

This thesis 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 solutions to defend against this attack by developing models and algorithms and examining them under different situations. This chapter will present a summary of the major contributions made by this thesis and the possible future directions of this work.

7.1 Conclusion

Cloud systems offer the ability to share the computing resources among cloud users efficiently to reduce the wastage of the resources. However, sharing physical computing resources opens the door for security threats on cloud systems and users. Specifically, the threats arise from SCAs when the malicious users can extract private information from other cloud users by merely sharing the physical resources.

The previous work focused on proposing formulating solutions under particular configurations and assumptions. In contrast, we followed a holistic approach to study the VMs allocation behaviours impact on securing the VMs allocation. As such, we studied the SCAs in CCEs and subsequently developed a model for secure VMs allocation that identify the objectives and constraints of the secure VMs allocations. Then, we developed algorithms to produce secure VMs allocations to protect cloud users against the threats from SCAs and examined them under different scenarios and structures to understand the possible situations that lead to securing the VMs. Moreover, we conclude that the stacking-based behaviours algorithms are more likely to produce secure allocations than those with spreading-based or random-based allocation behaviours algorithms. Furthermore, the VMs arrival time and the high available resources diversity between the available resources increase the chance of obtaining secure allocations. In addition, our stacking-based algorithms show the lowest PMs usage and VMs migration number leading to higher availability of the VMs and less power consumption.

We summarise the key steps toward finding the introduced solution of the thesis as follow:

- **Chapter** 2: This chapter presented a description of the attack model considered in this thesis, which is SCAs. Specifically, we describe the SCAs in CCEs and how the attacker utilises the sharing capabilities of virtualisation technology to form malicious attacks against cloud users. We provided recent examples of the SCAs published by cloud providers to demonstrate the serious impact of this attack on cloud systems. Moreover, we describe the cloud computing structure with its main components, including service and deployment models. Furthermore, we describe other security issues in CCEs resulting from the sharing of resources and remote access to the computing resources.

- **Chapter** 3: This chapter introduced related works of the proposed solution: obtaining a secure VMs allocation to defend against SCAs in CCEs. It includes classifying the presented solutions based on domains depending on how they contribute to defending against SCAs. The considered domains focus on allocating the VMs securely while considering other factors that potentially affect the performance, power consumption, or resources wastage of the cloud system. Furthermore, and from the studied related works, we presented a taxonomy of the factors affecting the secure VMs allocation in CCEs, including the effect of the attacker's behaviour and the attack impact and on which level of cloud system the attack is vulnerable. Moreover, we analysed the countermeasures of the SCAs based on the type of attack and the level of virtualisation.

- **Chapter** 4: This chapter defined and developed the model of secure VMs allocation in CCEs, including its objective and constraints. Specifically, the objective is to produce a secure VMs allocation to defend against SCAs in CCEs under the constraint of cloud resources. Moreover, we introduce the framework of the learning model that classifies the cloud users into particular types based on their behavioural analysis, hence, allocating their VMs based on the defined classifications. In addition, we evaluate the presented secure VMs allocation model by utilising a linear programming solver. This evaluation aims to examine the behaviour of secure optimal allocations under the defined model constraints. As such, the evaluation contributed to identifying the behaviour of the optimal secure VMs allocation in CCEs, thus developing algorithms following the optimal behaviour.

- **Chapter** 5: This chapter developed secure VMs allocation algorithms that aim to reduce the chance of malicious co-residency while using fewer available PMs in the cloud system. Specifically, developed two algorithms follow a stacking-based behaviour called SS and SRS, aiming to produce secure VMs allocation in CCEs. Furthermore, developed a VMs migration algorithm that aims to enhance the secure VMs allocation and keep the VMs allocation secure as possible. Additionally, we extensively evaluate the proposed algorithms under situations and scenarios to examine the behaviour of the VMs allocation algorithms leading to secure allocations under a variety of scenarios and structures. Thus, we evaluate the stacking, spreading and random VMs allocation behaviours under different PMs heterogeneity levels, diversity of available resources, various VMs arrival times and under different numbers of VMs according to their classified type.

- **Chapter** 6: This chapter developed a model and algorithms that consider the dependent VMs represented as graphs to examine their effect on the graph-based allocation. We consider the load similarity of the VMs and the type classification of the dependent VMs during the allocation. Therefore, we presented two algorithms called GbSS and GbSRS, which are an extension of the original two algorithms presented in the previous chapter; however, they allocate the VMs based on the graph-based model. Furthermore, we evaluated the graph-based VMs allocation behaviour on obtaining a secure allocation and investigated the factors that affect producing malicious co-residency in CCEs.

7.2 Future Work and Limitations

The researcher intends to continue exploring possible directions of this proposed work and enhance them accordingly. There are possible future work directions related to the proposed thesis and specificity to the secure VMs allocation in CCEs that the presented work has not tackled. The summarization of the possible future directions is as follows:

- Extending the study related to the proposed learning model framework that classifies the VMs based on their activities into specific types and allocates them subsequently. With the advancement of machine learning tools and detailed datasets related to cloud activities, it becomes possible to formulate and develop a model for secure VMs allocation accordingly. It includes developing an algorithm that dynamically allocates the VMs based on the classification model. Moreover, the classification can be extended to consider more than three types of VMs or classification. It could consider factors related to network connection, power consumption, and resource utilisation of the VMs. For instance, it cloud understands the behaviour of the malicious users according to their activities and footprint related to their resource's utilisation or network activities. This direction requires a detailed dataset that describe the users' activities during the life of their VMs in cloud systems. The limitation of the current proposed learning model's problem is that it can produce a false negative (FN) or false positive (FP) classification. We aim to add a layer that controls the classification's palpability failure, thus controlling the allocation outcome.
- Considering different allocation algorithms behaviour that potentially leads to reduce the chance of malicious co-residency. It includes consider-

ing other allocation models such as the Knapsack model. The Knapsack depends on allocating the VMs with the highest value considered on the given PMs as long as the total weight of the VMs not exceeding the selected PM. Knapsack's idea is to consider the value of the VMs during the allocation, which is related to the classification that we consider in this work. For example, if we assume that the value of the VMs is related to their activities, the VMs that are behaving suspiciously can be considered with a specific value that ensures that they are allocated with similar VMs. On the other hand, the VMs that are considered legit VMs can specify a different level of values, leading to allocating them together. As such, this model, with other allocation behaviour study is worth considering for future direction.

- Extending the proposed model to include tasks allocation on the hardware level in addition to the VMs level, as proposed. 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.

Appendix A

PuLP: Linear Programming Solver

A.1 Definition of PuLP

PuLP is an open-source linear programming solver package that utilises python programming to solve an optimisation problem. It allows solving the optimisation problem represented in the form of mathematical programs to produce an optimal solution. To clarify, mathematical programming (MP) represents a desirable situation in a mathematical model to find the optimal solution.

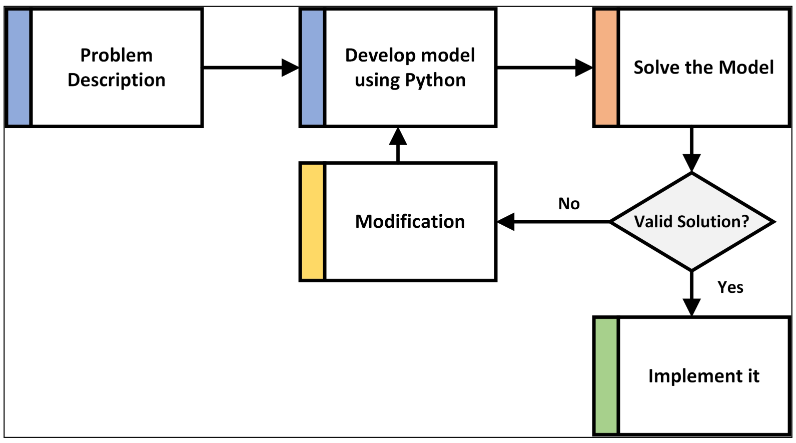


Figure A.1: The Process of Implementing PuLP

It starts by defining the problem of a current situation, such as a scheduling problem of an item. Then, create a model of this problem in mathematical form, subject to a set of constraints, to produce many potential acceptable solutions. One solution will be selected from the produced solutions and considered the optimal solution, thus calling this process an optimisation. Therefore, PuLP is a tool that specified in solving optimisation problems and contributes to helping of decision-making process using Python programming [70].

A.2 The Process of Implementing PuLP

PuLP follows a defined process to obtain the optimal solution of a given situation, as illustrated in Figure A.1.

A.2.1 First Step

The first step of the PuLP process starts by describing the problem that represents the optimisation objective, which is subject to constraints. This step, describing the model, includes describing the data and variables contributing to the optimal solution. For example, in this work, the problem description obtains a secure VMs allocation where the target VMs are allocated separately from the malicious VMs. In other words, the objective is to minimise the situations where the malicious co-residency may occur in the VMs allocation process. This objective is subject to constraints such as the availability of resources and utilising few numbers of the available PMs.

A.2.2 Second Step

The second step of the PuLP process is to develop the mathematical model using Python language. This process involves identifying the decision variables, the objective function, the constraints and the data needed for solving the model. The decision variables are the variables that decide to consider or not consider a potential solution based on the optimisation problem objective. For example, the decision variables will aim to select the VMs allocation with minimum malicious co-residency in our work. The decision variable could be a variable defined from given data or produced function.

Furthermore, developing the mathematical model includes defining the objective function representing the desired optimal solution from the problem description. Alternatively stated, in our work, for example, the goal is minimising the malicious co-residency of a given potential allocation. Other situations may require maximising the objective function depending on the description of the problem and the desired outcome. Moreover, in the second step, defining the constraints contributes to selecting the optimal solutions for the optimisation objective. In other words, the constraints are the ones related to bound the model to achieve its purpose. For example, the allocation of VMs should not exceed the available resources of the PMs. Lastly, defining the data needed for the model or the variables needed to formulate the objective function, constraints, and optimal solutions. For instance, in this work, the variables are the VMs with all their related data, such as the VMs type, the number of VMs, the resources of each VM. Also, the PMs data such as number of PMs, available resources.

A.2.3 Third Step

The third step is to solve the model to have the optimal solution to the described problem. The produced solution considers, in many cases, the best possible solution of the given described problem and constraints. The solution produced in this process, optimal solution, relies on describing and developing the problem, or in some cases, the quality of the available data, i.e., the best solution may not necessarily be the optimal one. For this reason, it is vital to perform an extensive analysis of the process for producing the optimal solution in order to examine the produced solution. Thus, possible modification of the developed model, including the objective function, or constraints may introduce to refine the solution and obtain better results. Further, the analysis includes examining the different situations in which the optimal solutions may change, hence, altering the given problem's constraints and performing a continuous comparison. Moreover, the validation must be made to understand the meaning of the produced solution and the behaviour of different situations.

A.2.4 Fourth Step

The last step is to implement the solution if the solution is valid and does not require any modification. The most challenging of this step is to translate the produced solution from mathematical form and raw data to a meaningful behaviour to the described problem. For example, in this work, the produced output is set of VMs allocation presented in a form of raw data that shows each selected VM and its allocated PM.

Appendix B

CloudSim: Simulation tool for cloud environments

Cloud Simulation (CloudSim) is an open-source cloud simulation environment that builds based on cloud system workloads that aim to simulate the provisioning of cloud computing systems. Therefore, it contributes to providing a valuable estimate of the expected outcome of certain situations and an initial understanding of the behaviour of the allocation process. Generally, cloud simulators are helpful to provide a solution and projection of the real-world scenarios; thus, we utilise CloudSim to examine our proposed algorithms [18]. This section will explain the components of CloudSim utilised by our work and the process flow that CloudSim follows to allocate the VMs in the simulation environments.

B.1 CloudSim Architecture

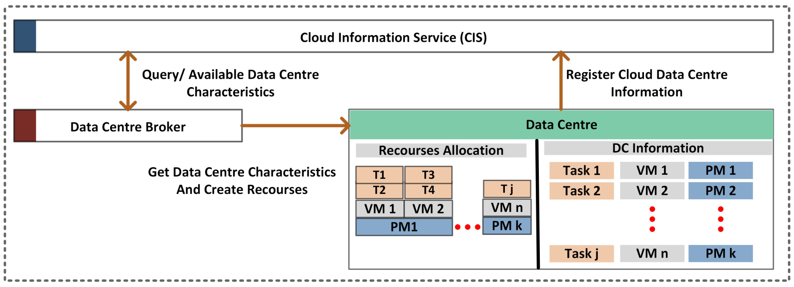

Cloud Information Service (CIS)
Query/ Available Data Centre Characteristics
Register Cloud Data Centre Information
Data Centre Broker
Data Centre
Recourses Allocation
DC Information
Get Data Centre Characteristics And Create Recourses
T1
T2
T3
T4
T j
VM 1
VM 2
VM n
PM1
PM k
Task 1
Task 2
Task j
VM 1
VM 2
VM n
PM 1
PM 2
PM k

Figure B.1: The Data flow of CloudSim Components

CloudSim is a powerful simulation tool that can simulate different aspects related to the CCEs architecture. For instance, it can simulate the performance, power consumption, SLAs violation and security of different levels of the CCEs. Including, but not limited to, Data Centre (DC), PMs, VMs and VMs tasks simulations. It also can simulate the VMs allocation and VMs migration of different objectives. For example, a VMs allocation that aims to allocate the VMs to balance the power consumption of the selected PMs, or as in our work, a secure VMs allocation that aim to minimise the malicious co-residency.

As shown in Figure B.1, CloudSim has several components and a defined process in the simulation environment. These components relate to VMs allocation and migration, including Cloud Information Service (CIS), DC broker, and DC resources.

B.1.1 Cloud Information Service

The CIS act as a resources registration for the components in the simulation environments where all the cloud resources information are registered and discovered by the cloud broker. In other words, the cloud broker, labelled as data centre broker in Figure B.1, can discover the resources current situations and allocation process through the CIS. The CIS is also responsible for notifying the other components of simulation completion.

B.1.2 Data Centre Broker

The DC broker acted as an intermediate between cloud users' demanded infrastructure resources and available resources in the cloud environment. For example, when a user requires specific cloud services to be hosted on the cloud, the cloud broker will then work as a cloud consultant for the user. It will start the negotiation with the CIS to obtain the required resources information and allocate them for this user if they comply with the objective of the allocation, for example, security objectives.

B.1.3 Data Centre

The data centre resources include the information related to the computing resources of the PMs and VMs and the processing of the tasks. Firstly, the data centre has information related to the infrastructure resources of the data centre. Including the characteristics of the data centre, for example, the system architecture, the operating systems, the hypervisors, the PMs, the storages, the VMs allocation policy utilised, the time zones for each cluster of the data centre and the cost of utilising data centre entities.

Secondly, the information about the PMs, which is also part of the data centre, includes the PMs' hardware information such as the number and size of CPUs, PMs storages, the bandwidth of PMs, and the VM processing scheduling policy for each PM. Thirdly, the information about the VMs, which includes the VMs, require resources information such as the number and size of CPUs, PMs storage, the bandwidth of VMs and the task scheduling policy for each task allocated to be processed on the VM.

Finally, the tasks that need to be processed act as the intended application from each VM. For example, it can be considered a web application hosted on a VM or a database environment. The tasks include the task size, processing time, processing element (CPUs core of the hosted VM), and the ID for the task. The tasks can be created as a synthetic unit acting as an actual processing data or imported externally from other sources.

B.1.4 Scheduling and Allocation Policy

The scheduling policy on the PMs is different from the allocation policy mentioned under the data centre characteristics. The allocation policy, or as we refer to as the VMs allocation policy, is responsible for allocating the VMs into the available PMs based on the defined allocation policy.

On the other hand, the scheduling policy, defined under the PMs characteristics, is responsible for managing and controlling the processing of the VMs sharing the same PM. In other words, if we have two VMs sharing the same PM, then the defined scheduling policy will decide which VM can allocate more resources than the other VM and at which time.

There are two types of scheduling policy, time-shared and spaced-shared; time-shared means that the VMs sharing the same PM can be processed on the same computing hardware, for example on the same CPU core, but in different time stamps. These VMs can be scheduled based on the defined requirement for each VMs and the scheduling policy. However, the space-shared allocate a specific space for each VM, and it will be allocated for this VM all the processing time.

B.2 CloudSim Data Flow

As illustrated in Figure B.1, the process of CloudSim starts by registering each data centre entity, or component, in the CIS. The registration includes all the information related to the DCs and their available resources such as PMs, VMs and other components. Moreover, it includes registering the detailed characteristics of each entity of the DC, such as network topology, storage structure and the DCs architecture.

After registering the components in CIS, the DC broker will start to collect the user requirements for computing resources, which includes the characteristics of the tasks, the required VMs and PMs. The user requirements vary based on the simulation objectives and the assumption made for utilising the simulation. For instance, some of the simulation scenarios only require defining the characteristics of the tasks; then, the DC broker will identify the computing resources requires for these tasks. Meaning, it will define the DC in which the tasks will be processed, if there are many DCs defined, and the required number of VMs and PMs. On the other hand, in our work, we study the secure VMs allocation behaviour. Thus we defined the tasks, the VMs and the PMs characteristics to study how the allocation behaviour yields to a secure allocation under different situations.

Afterwards, based on the requirements collected by the user and the DC characteristics defined in the CIS, the DC broker will communicate with the selected DC to start the process of the resource's allocation. The selection of which resources are to host the user requirements depends on different levels and factors. For instance, if the simulation environment is defined with many DCs and the objective is to obtain a DC that can host specific user requirements related to data levels, such as data availability or geographic location, the DC broker will select the DC to complies with the data requirements.

Additionally, as in our work, if the objective is to obtain a VM allocation, that result in achieving a desirable goal, such as secure VM allocation, power consumption minimisation, or load balancing of the VMs. Then, the DC broker will select the PMs based on the defined VMs allocation algorithm that complies with the requirements of the allocation algorithm. Further, the allocation could be implemented to the lower level of the cloud environments, i.e., on the tasks level. In this case, the DC broker decides which tasks can be hosted and processed on which DC, PMs and VMs, depends on the defined task allocation policy.

Ultimately, the allocation and selection of the computing resources, or DCs, depends on the objective of the allocation policy and at which level of the cloud environment needs to be simulated.

B.3 NetworkCloudSim: Network-based simulation extended from CloudSim

In our work, we utilised a network-based simulation called Network Cloud Simulation (NetCloudSim) [36]. It is an extension of the CloudSim that can support network entities simulation of the cloud environment. NetCloudSim follows the same process described earlier in CloudSim; however, it can simulate networks entities. These entities include network topologies, network switches levels, the latency of the switches, bandwidth of network levels, and resource allocation based on the network specifications.

Specifically, NetCloudSim offers the ability to modify the architecture of the DC to a network-aware DC to obtain more realistic results. As shown in Figure 6.2, it includes adding the switches entities that are utilised to define the connectivity between the hardware components of the DC. Alternatively stated, identifying the network topology of the entire DC, including the PMs, the switches connectivity. Hence, defining the network relation between each VM hosted in the cloud environment. NetCloudSim defines the switches into three levels; root, aggregation and edge switch.

Firstly the root switch, or the core switch, is the switch that connects the internal network of the DC to the external network. Thus, the packet flow of this switch is downlink to other network entities defined, for example, aggregation and edge switches. The downlink means that the flow of the packet is directed from the internet, external network, to the devices in the internal network. In contrast, the upper link is in the opposite direction. Secondly, the aggregation switch, which is defined in NetCloudSim as the switch entity, connects the switches in the upper level with the switches in the lower level of network topology. Therefore, the packet flow of this switch is either a downlink or upper link, depends on the traffic requirements for each situation. Lastly, the edge switch, which is the switch that connects directly to the PMs or other hardware components; hence, the packet flow of this switch is an upper link.

Overall, NetCloudSim can be utilised for many objectives related to resources allocation or optimisation. For instance, it is helpful for resources delay calculation when the packet flow is measured between cloud entities. In our work, we utilised the NetCloudSim to define a network-based DC with specific network topology and defined the relation between hardware entities. Therefore, the defined DC network topology controls the resources allocation and migration between cloud components based on the allocation and migration policy objectives.

B.4 Algorithms, VMs Arrival and Network Integration to CloudSim

The previous sections explained the CloudSim data flow and their main components, including user requirements specification to VMs allocation. Moreover, we explained the process of the network-based extension of CloudSim, which is NetCloudSim that simulate networks entities. This section will explain how the integration has been made to the CloudSim environment to simulate the defined algorithms, the scenario of VMs arrivals, and the network topology. Overall, most of the components of CloudSim, in Figure B.2, have been modified during this project's development, but here we only focus on the major components.

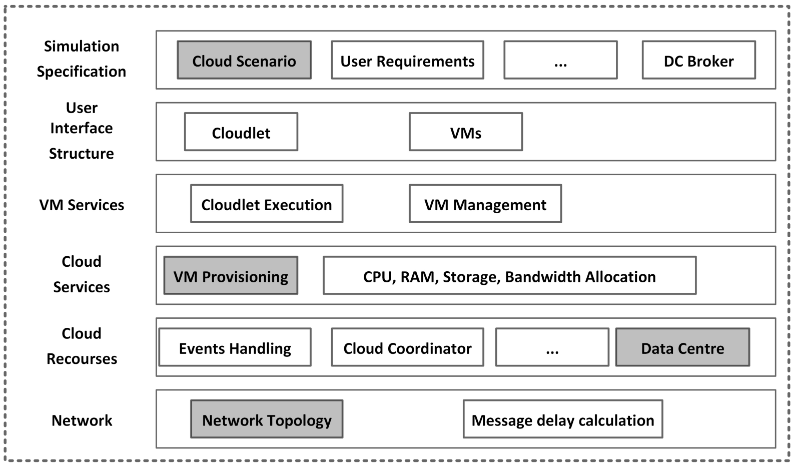


Figure B.2: CloudSim Architecture [18].

B.4.1 VMs Arrival Integration

The VMs arrival scenarios defined in section 5.6.1 have been integrated into the simulation specification layer, specifically in the cloud scenario components, which means that before starting the execution of the VMs allocation, these requirements are identified. Therefore, the VMs arrival is defined at this stage.

B.4.2 VMs Allocation Integration

The five algorithms SS, SRS, PSSF, Rand and RR has been implemented in the VM provisioning component under the cloud service layer. The provisioning includes creating VMs on the selected PMs according to the defined steps of each algorithm, i.e., VMs allocation behaviour. Moreover, according to the availability of the resources such as CPU, RAM and storage.

B.4.3 VMs Migration Integration

The VMs migration algorithm is implemented under the data centre component in the cloud resources layer and the VM provisioning components under the cloud service layer. The VMs migration only triggers if the allocation algorithm cannot obtain a secure allocation. As such, the VMs provisioning will notify the data centre of the failed allocation, which will trigger the VMs migration by the data centre, then the event will return to the VMs provisioning for allocating the VMs selected for migration. Afterwards, the VM provisioning will continue to allocate the VM that triggers the migration again, as explained in each algorithm step.

B.4.4 Network Integration

The network topology, utilised by NetCloudSim for simulating the graph-based allocation, is implemented under the network topology component in the network layer. In this component, we created the data centre topology to identify the network connection between the PMs based on Fat-Tree architecture. The VM provisioning and data centre components will utilise the topology during the VMs allocation and migration.

B.4.5 CloudSim Validation

We validated the results generated by CloudSim to identify two main outcomes. First, identify whether the produced allocation followed the identified behaviour by the VM provisioning component. Second, whether the requested VMs allocated and followed the allocation scenarios, such as the arrival of VMs. In the first part, we started with a small number of VMs for each algorithm and examined them against many allocation scenarios. Each scenario is printed and validated manually to ensure that the allocation steps are followed. In the second part, we created many functions responsible for validating that each VM is allocated, the resources are met, the structure of PMs are, VMs for each scenario are met, and the migration validation. For example, if some VMs are failed to obtain allocation, the function responsible for this part will be triggered. It will show a failure in the result; as such, the outcome will be disqualified, and the failure will be fixed.

Appendix C

Microsoft Azure: Traces of VMs Workload in Azure Data Centre

In our proposed algorithms and part of the experiments' examination, one of the goals is to examine the effect of VM allocations when the resources of the VMs are heterogeneous. Subsequently, we required realistic traces of the VMs from an existing cloud provider to examine our algorithms under actual heterogeneous VMs traces. Thus, we utilised the Azure VMs traces published by the Microsoft team, which contains the VMs workload of Azure DC [22].

C.1 VMs workload characteristics

The Azure's VM workload contains comprehensive information about the lifetime of the VMs in Azure cloud systems. It includes VMs resources such as VMs lifetime, CPUs and memory utilisation, and the VMs users information. The duration of the VMs traces was collected from the Azure cloud data centre for 30 consecutive days. The related details of the VMs trace to our work as follow:

- **Subscription ID**: The subscription refers to the user who owns the VM; each user has a unique subscription identification. A group of VMs might share the same subscription ID if they belong to the same user.
- **Deployment ID**: The deployment refers to the nature of the VMs launched for a specific purpose by the user. For example, a user may wish to select a group of VMs to perform specific tasks, a web applica-

tion hosting. Furthermore, these VMs can be deployed under specific requirements; for instance, the location of these VMs are chosen to be in a specific place. Thus, the user, or CSP consultant, may host these VMs under the exact deployment identification. Subsequently, one user with one subscription ID may have many deployment IDs based on the needed requirements.

- **Count VMs created**: The count of the VMs in the Azure traces are 2.6 million VMs with 6687 subscription IDs. We only utilised one VM from each user to increase the heterogeneity level between VMs upon their arrival in our work. We will explain further in the following section.
- **VM lifetime**: The VMs lifetime includes the arrival of the VMs, which is the created time of the VM until the VMs terminated. As we study the effect of VMs arrival under different scenarios and structures, this traces provide a realistic VMs arrival with heterogeneous structure. We focus mainly on the arrival time of the VMs with their resources requirements.
- **VMs CPUs and RAM**: The VMs trace provides extensive information about the structure of the CPUs and RAMs assigned to the VMs. The CPU cores requirements range from 2-30 CPU cores, while the size of the RAMs requirements ranges from 2-70 RAM size. Moreover, the traces provides information about CPU and RAM utilisation for each VM, including the maximum and minimum utilisation. The utilisation information is essential for research focusing on power consumption optimisation or VMs migration based on utilisation threshold. However, in our work, we only required the information related to the RAM sizes for each VMs with their CPU cores count.

C.2 Integration with CloudSim

The motivation behind utilising Azure workloads is to produce a set of heterogeneous VMs in the matter of resources structure. As we described earlier, we have a group of subscription IDs, refer as cloud users, and each one can have many VMs. The problem with the existing tracers is that each cloud user can have many VMs with the structure of the same resources, making the structure of the VMs non-heterogeneous. For example, a user X owns 150 VMs, and all these VMs have the same CPU and RAM, 2 CPU core and 4 RAM size. In this part of our work, we want to examine the effect of the heterogeneous VMs on the secure VMs allocation; thus, we made further steps to the existing workloads to fits our purpose.

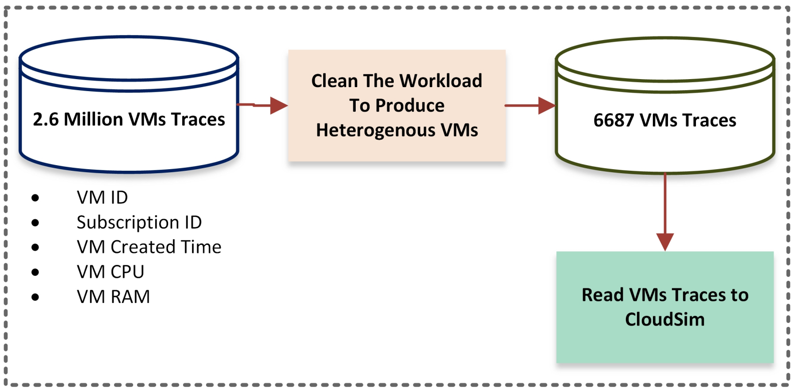


Figure C.1: Integration of Azure VMs workload with CloudSim

As illustrated in Figure C.1, we purposed a cleaning of the VMs traces to produce a group of VMs with a heterogeneous structure. The cleaning step of the data is simple yet effective to our purpose, which performs a selection of one VM from each cloud user. Alternatively stated, from each subscription ID, we only select one VM belonging to this ID, producing 6687 VMs with a heterogeneous structure. Performing this step does not imply that the existing VMs traces are not heterogeneous; on the contrary, they are heterogeneous, but not enough to fits our purpose. For instance, when a group of VMs, belonging to the same user with similar arrival time examine and VMs recourse structure, they will perform as non-heterogeneous VMs. Our goal is to examine the heterogeneous VMs from a realistic cloud provider, and the cleaning step contributes to this objective. Afterwards, these VMs traces will be loaded by CloudSim and created on the simulation environment to be simulated under different conditions and structures.

Bibliography

[1] Zaina Afoulki, Aline Bousquet, Jonathan Rouzaud-Cornabas, et al. A security-aware scheduler for virtual machines on iaas clouds. *Report 2011*, 2011.

[2] Amit Agarwal and Ta Nguyen Binh Duong. Secure virtual machine placement in cloud data centers. *Future Generation Computer Systems*, 100:210–222, 2019.

[3] Farhad Ahamed, Seyed Shahrestani, and Bahman Javadi. Security aware and energy-efficient virtual machine consolidation in cloud computing systems. In *2016 IEEE Trustcom/BigDataSE/ISPA*, pages 1516–1523. IEEE, 2016.

[4] Raja Wasim Ahmad, Abdullah Gani, Siti Hafizah Ab Hamid, Muhammad Shiraz, Feng Xia, and Sajjad A Madani. Virtual machine migration in cloud data centers: a review, taxonomy, and open research issues. *The Journal of Supercomputing*, 71(7):2473–2515, 2015.

[5] Saeed Al-Haj, Ehab Al-Shaer, and HariGovind V Ramasamy. Security-aware resource allocation in clouds. In *2013 IEEE International Conference on Services Computing*, pages 400–407. IEEE, 2013.

[6] Mansour Aldawood and Arshad Jhumka. Secure allocation for graph-based virtual machines in cloud environments. In *2021 18th International Conference on Privacy, Security and Trust (PST)*. IEEE, Accepted on 22 October 2021.

[7] Mansour Aldawood, Arshad Jhumka, and Suhaib A Fahmy. Sit here: Placing virtual machines securely in cloud environments. In *CLOSER*, pages 248–259, 2021.

[8] Mansour Aldawood, Arshad Jhumka, and Suhaib A Fahmy. The study of secure vms allocation behaviours in the cloud computing environments. In *Communications in Computer and Information Science (CCIS)*. Under Review, Submitted to Springer, 2021.

[9] Alejandro Cabrera Aldaya, Billy Bob Brumley, Sohaib ul Hassan, Cesar Pereida García, and Nicola Tuveri. Port contention for fun and profit. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 870–887. IEEE, 2019.

[10] Mohamed Almorsy, John Grundy, and Ingo Müller. An analysis of the cloud computing security problem. *arXiv preprint arXiv:1609.01107*, 2016.

[11] Yossi Azar, Seny Kamara, Ishai Menache, Mariana Raykova, and Bruce Shepard. Co-location-resistant clouds. In *Proceedings of the 6th Edition of the ACM Workshop on Cloud Computing Security*, pages 9–20, 2014.

[12] Mehdi Bahrami, Abhishek Malvankar, Karan K Budhraja, Chinmay Kundu, Mukesh Singhal, and Ashish Kundu. Compliance-aware provisioning of containers on cloud. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pages 696–700. IEEE, 2017.

[13] Taghreed Balharith and Fahd Alhaidari. 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, 2019.

[14] Mohammad-Mahdi Bazm, Marc Lacoste, Mario Südholt, and Jean-Marc Menaud. Side channels in the cloud: Isolation challenges, attacks, and countermeasures. 2017.

[15] Mouhebeddine Berrima, Aïcha Katajina Nasr, and Narjes Ben Rajeb. Co-location resistant strategy with full resources optimization. In *Proceedings of the 2016 ACM on Cloud Computing Security Workshop*, pages 3–10, 2016.

[16] Khalid Bijon, Ram Krishnan, and Ravi Sandhu. 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, 2015.

[17] Harold Booth, Doug Rike, and Gregory A Witte. The national vulnerability database (nvd): Overview. 2013.

[18] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. 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, 2011.

[19] Eddy Caron, Anh Dung Le, Arnaud Lefray, and Christian Toinard. 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*, pages 125–131. IEEE, 2013.

[20] Kim-Kwang Raymond Choo. Cloud computing: Challenges and future directions. *Trends and Issues in Crime and Criminal justice*, (400):1–6, 2010.

[21] Luigi Coppolino, Salvatore D'Antonio, Giovanni Mazzeo, and Luigi Romano. Cloud security: Emerging threats and current solutions. *Computers & Electrical Engineering*, 59:126–140, 2017.

[22] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 153–167, 2017.

[23] Wesam Dawoud, Ibrahim Takouna, and Christoph Meinel. Infrastructure as a service security: Challenges and solutions. In *2010 the 7th International Conference on Informatics and Systems (INFOS)*, pages 1–8. IEEE, 2010.

[24] Joost CF De Winter. Using the student's t-test with extremely small

sample sizes. *Practical Assessment, Research, and Evaluation*, 18(1):10, 2013.

[25] D Dhanya and D Arivudainambi. 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.

[26] Weichao Ding, Chunhua Gu, Fei Luo, Yaohui Chang, Ulysse Rugwiro, Xiaoke Li, and Geng Wen. Dfa-vmp: An efficient and secure virtual machine placement strategy under cloud environment. *Peer-to-Peer Networking and Applications*, 11(2):318–333, 2018.

[27] Yucong Duan, Guohua Fu, Nianjun Zhou, Xiaobing Sun, Nanjangud C Narendra, and Bo Hu. Everything as a service (xaas) on the cloud: origins, current and future trends. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 621–628. IEEE, 2015.

[28] Samuel J Dwyer III, Alfred C Weaver, and K Knight Hughes. Health insurance portability and accountability act. *Security Issues in the Digital Medical Enterprise*, 72(2):9–18, 2004.

[29] VMware ESXi. VMware ESXi: The Purpose-Built Bare Metal Hypervisor. `https://www.vmware.com/uk/products/esxi-and-esx.html`, 2021. [Online; accessed 31-July-2021].

[30] Fairouz Fakhfakh, Hatem Hadj Kacem, and Ahmed Hadj Kacem. Simulation tools for cloud computing: A survey and comparative study. In *2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS)*, pages 221–226. IEEE, 2017.

[31] Margaret M Fleck. Building blocks for theoretical computer science (version 1.3). 2013.

[32] David Freet, Rajeev Agrawal, Jessie J Walker, and Youakim Badr. Open source cloud management platforms and hypervisor technologies: A review and comparison. In *SoutheastCon 2016*, pages 1–8. IEEE, 2016.

[33] Xing Gao, Zhongshu Gu, Mehmet Kayaalp, Dimitrios Pendarakis, and Haining Wang. Containerleaks: Emerging security threats of information leakages in container clouds. In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 237–248. IEEE, 2017.

[34] Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

[35] Robert S Garfinkel and George L Nemhauser. The set-partitioning problem: set covering with equality constraints. *Operations Research*, 17(5):848–856, 1969.

[36] Saurabh Kumar Garg and Rajkumar Buyya. Networkcloudsim: Modelling parallel applications in cloud simulations. In *2011 Fourth IEEE International Conference on Utility and Cloud Computing*, pages 105–113. IEEE, 2011.

[37] Paul R Genssler, Oliver Knodel, and Rainer G Spallek. Securing virtualized fpgas for an untrusted cloud. In *Proceedings of the International Conference on Embedded Systems, Cyber-physical Systems, and Applications (ESCS)*, pages 3–9. The Steering Committee of The World Congress in Computer Science, Computer . . . , 2018.

[38] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: A scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 51–62, 2009.

[39] Berk Gulmezoglu, Thomas Eisenbarth, and Berk Sunar. Cache-based application detection in the cloud using machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 288–300, 2017.

[40] Talal Halabi and Martine Bellaiche. Towards quantification and evaluation of security of cloud service providers. *Journal of Information Security and Applications*, 33:55–65, 2017.

[41] Jin Han, Wanyu Zang, Songqing Chen, and Meng Yu. Reducing security risks of clouds through virtual machine placement. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 275–292. Springer, 2017.

[42] Jin Han, Wangyu Zang, Li Liu, Songqing Chen, and Meng Yu. Risk-aware multi-objective optimized virtual machine placement in the cloud. *Journal of Computer Security*, 26(5):707–730, 2018.

[43] Yi Han, Tansu Alpcan, Jeffrey Chan, and Christopher Leckie. Security games for virtual machine allocation in cloud computing. In *International Conference on Decision and Game Theory for Security*, pages 99–118. Springer, 2013.

[44] Yi Han, Jeffrey Chan, Tansu Alpcan, and Christopher Leckie. Virtual machine allocation policies against co-resident attacks in cloud computing. In *2014 IEEE International Conference on Communications (ICC)*, pages 786–792. IEEE, 2014.

[45] Yi Han, Tansu Alpcan, Jeffrey Chan, Christopher Leckie, and Benjamin IP Rubinstein. 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, 2015.

[46] Yi Han, Jeffrey Chan, Tansu Alpcan, and Christopher Leckie. 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.

[47] M Azua Himmel and F Grossman. Security on distributed systems: Cloud security versus traditional it. *IBM Journal of Research and Development*, 58(1):3–1, 2014.

[48] Helmut Hlavacs, Thomas Treutner, Jean-Patrick Gelas, Laurent Lefevre, and Anne-Cecile Orgerie. Energy consumption side-channel attack at virtual machines in a cloud. In *2011 IEEE Ninth International Conference*

on Dependable, Autonomic and Secure Computing, pages 605–612. IEEE, 2011.

[49] Dijiang Huang, Tianyi Xing, and Huijun Wu. Mobile cloud computing service models: a user-centric approach. *IEEE network*, 27(5):6–11, 2013.

[50] Jinho Hwang, Sai Zeng, Frederick y Wu, and Timothy Wood. A component-based performance comparison of four hypervisors. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pages 269–276. IEEE, 2013.

[51] Microsoft Hyper-V. Hyper-V Technology Overview. `https://docs.microsoft.com/en-us/windows-server/virtualization/hyper-v/hyper-v-technology-overview`, 2021. [Online; accessed 31-July-2021].

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

[53] Ming Jiang, Tom Kirkham, and Craig Sheridan. An evolutionary cultural algorithm based risk-aware virtual machine scheduling optimisation in infrastructure as a service (iaas) cloud. In *CLOSER (1)*, pages 267–272, 2016.

[54] Taesoo Kim, Marcus Peinado, and Gloria Mainar-Ruiz. {STEALTHMEM}: System-level protection against cache-based side channel attacks in the cloud. In *21st {USENIX} Security Symposium ({USENIX} Security 12)*, pages 189–204, 2012.

[55] Vladimir Kiriansky, Ilia Lebedev, Saman Amarasinghe, Srinivas Devadas, and Joel Emer. Dawg: A defense against cache timing attacks in speculative execution processors. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 974–987. IEEE, 2018.

[56] WMCJT Kithulwatta, Wiraj Udara Wickramaarachchi, KPN Jayasena, BTGS Kumara, and RMKT Rathnayaka. Adoption of docker containers as

an infrastructure for deploying software applications: A review. *Advances on Smart and Soft Computing*, pages 247–259, 2022.

[57] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, et al. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1–19. IEEE, 2019.

[58] Richard E Korf. A new algorithm for optimal bin packing. In *Aaai/Iaai*, pages 731–736, 2002.

[59] Charles E Leiserson. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE transactions on Computers*, 100(10):892–901, 1985.

[60] Min Li, Yulong Zhang, Kun Bai, Wanyu Zang, Meng Yu, and Xubin He. Improving cloud survivability through dependency based virtual machine placement. In *SECRYPT*, pages 321–326, 2012.

[61] Xin Liang, Xiaolin Gui, AN Jian, and Dewang Ren. 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, 2017.

[62] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown. *arXiv preprint arXiv:1801.01207*, 2018.

[63] Haikun Liu, Hai Jin, Xiaofei Liao, Liting Hu, and Chen Yu. Live migration of virtual machine based on full system trace and replay. In *Proceedings of the 18th ACM international symposium on High performance distributed computing*, pages 101–110, 2009.

[64] Vu Duc Long and Ta Nguyen Binh Duong. 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)*, pages 64–69. IEEE, 2020.

[65] Fabio Lopez-Pires and Benjamin Baran. Virtual machine placement literature review. *arXiv preprint arXiv:1506.01509*, 2015.

[66] Olivier Markowitch, Liran Lerman, and Gianluca Bontempi. Side channel attack: an approach based on machine learning. In *Proceedings of 2nd International Workshop on Constructive Side-Channel Analysis and Security Design, COSADE 2011*, pages 29–41. Schindler and Huss, 2011.

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

[68] Peter Mell, Tim Grance, et al. The nist definition of cloud computing. 2011.

[69] Stuart Mitchell. An introduction to pulp for python programmers. *Python Papers Monograph*, 1(14):2009, 2009.

[70] Stuart Mitchell, Michael OSullivan, and Iain Dunning. Pulp: a linear programming toolkit for python. *The University of Auckland, Auckland, New Zealand*, page 65, 2011.

[71] Chirag Modi, Dhiren Patel, Bhavesh Borisaniya, Hiren Patel, Avi Patel, and Muttukrishnan Rajarajan. A survey of intrusion detection techniques in cloud. *Journal of network and computer applications*, 36(1):42–57, 2013.

[72] Soo-Jin Moon, Vyas Sekar, and Michael K Reiter. 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, 2015.

[73] Varun Natu and Ta Nguyen Binh Duong. Secure virtual machine placement in infrastructure cloud services. In *2017 IEEE 10th Conference on Service-Oriented Computing and Applications (SOCA)*, pages 26–33. IEEE, 2017.

[74] Anthony Nicolaides. *Combinations, Permutations, Probabilities*, volume 10. PASS PUBLICATIONS, 1994.

[75] Yuqin Qiu, Qingni Shen, Yang Luo, Cong Li, and Zhonghai Wu. A secure virtual machine deployment strategy to reduce co-residency in cloud. In *2017 IEEE Trustcom/BigDataSE/ICESS*, pages 347–354. IEEE, 2017.

[76] Mahdi Rabbani, Yong Li Wang, Reza Khoshkangini, Hamed Jelodar, Ruxin Zhao, and Peng Hu. A hybrid machine learning approach for malicious behaviour detection and recognition in cloud computing. *Journal of Network and Computer Applications*, 151:102507, 2020.

[77] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. 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*, pages 199–212, 2009.

[78] Korir Sammy, Ren Shengbing, and Cheruiyot Wilson. Energy efficient security preserving vm live migration in data centers for cloud computing. *IJCSI International Journal of Computer Science Issues*, 9(2):1694–0814, 2012.

[79] Laura Savu. Cloud computing: Deployment models, delivery models, risks and research challenges. In *2011 International Conference on Computer and Management (CAMAN)*, pages 1–4. IEEE, 2011.

[80] Alexander Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, 1998.

[81] Read Sprabery, Konstantin Evchenko, Abhilash Raj, Rakesh B Bobba, Sibin Mohan, and Roy H Campbell. A novel scheduling framework leveraging hardware cache partitioning for cache-side-channel elimination in clouds. *arXiv preprint arXiv:1708.09538*, 2017.

[82] Raphael Spreitzer, Veelasha Moonsamy, Thomas Korak, and Stefan Mangard. Systematic classification of side-channel attacks: A case study for mobile devices. *IEEE Communications Surveys & Tutorials*, 20(1): 465–488, 2017.

[83] Qian Sun, Qingni Shen, Cong Li, and Zhonghai Wu. Selance: Se-

cure load balancing of virtual machines in cloud. In *2016 IEEE Trustcom/BigDataSE/ISPA*, pages 662–669. IEEE, 2016.

[84] Rich Uhlig, Gil Neiger, Dion Rodgers, Amy L Santoni, Fernando CM Martins, Andrew V Anderson, Steven M Bennett, Alain Kagi, Felix H Leung, and Larry Smith. Intel virtualization technology. *Computer*, 38 (5):48–56, 2005.

[85] Venkatanathan Varadarajan, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. A placement vulnerability study in multi-tenant public clouds. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*, pages 913–928, 2015.

[86] Pier Luigi Ventre, Claudio Pisa, Stefano Salsano, Giuseppe Siracusano, Florian Schmidt, Paolo Lungaroni, and Nicola Blefari-Melazzi. Performance evaluation and tuning of virtual infrastructure managers for (micro) virtual network functions. In *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 141–147. IEEE, 2016.

[87] Ziqi Wang, Rui Yang, Xiao Fu, Xiaojiang Du, and Bin Luo. A shared memory based cross-vm side channel attacks in iaas cloud. In *2016 IEEE conference on computer communications workshops (INFOCOM WKSHPS)*, pages 181–186. IEEE, 2016.

[88] Mark A Will and Ryan KL Ko. Secure fpga as a service—towards secure data processing by physicalizing the cloud. In *2017 IEEE Trustcom/BigDataSE/ICESS*, pages 449–455. IEEE, 2017.

[89] Yuchen Wong and Qingni Shen. 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, 2018.

[90] VMware Workstation. VMware Workstation Pro Documentation. `https://docs.vmware.com/en/VMware-Workstation-Pro/index.html`, 2021. [Online; accessed 31-July-2021].

[91] Yuping Xing and Yongzhao Zhan. Virtualization and cloud computing. In *Future Wireless Networks and Information Systems*, pages 305–312. Springer, 2012.

[92] Si Yu, Xiaolin Gui, Feng Tian, Pan Yang, and Jianqiang Zhao. 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*, pages 1078–1083. IEEE, 2013.

[93] Xuebiao Yuchi and Sachin Shetty. Enabling security-aware virtual machine placement in iaas clouds. In *MILCOM 2015-2015 IEEE Military Communications Conference*, pages 1554–1559. IEEE, 2015.

[94] Ji-Liang Zhang, Wei-Zheng Wang, Xing-Wei Wang, and Zhi-Hua Xia. Enhancing security of fpga-based embedded systems with combinational logic binding. *Journal of Computer Science and Technology*, 32(2):329–339, 2017.

[95] Tianwei Zhang, Yinqian Zhang, and Ruby B Lee. Cloudradar: A real-time side-channel attack detection system in clouds. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 118–140. Springer, 2016.

[96] Yulong Zhang, Min Li, Kun Bai, Meng Yu, and Wanyu Zang. Incentive compatible moving target defense against vm-colocation attacks in clouds. In *IFIP international information security conference*, pages 388–399. Springer, 2012.

[97] YongBin Zhou and DengGuo Feng. Side-channel attacks: Ten years after its publication and the impacts on cryptographic module security testing. *IACR Cryptology ePrint Archive*, 2005:388, 2005.