Let's Look Out For Each Other: A Distributed Framework for Botcloud Detection

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
IN THE FACULTY OF SCIENCE AND ENGINEERING

2022

By
Twisha
School of Computer Science

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

Agg-VMT	Aggregated Virtual Memory Trust Table
AIS	Artificial Immune System
BM	Bad Mouthing
Bot-VMD	BotVM Detector
bot-VMT-T	botVM Virtual trust Table
Bs	Ballotstuffing
bVMD	botVM Detection
C-OH	Communication Overhead
C&C	Command-and-Control
CAGR	Compound Annual Growth Rate
Com	Combined
D&A	Forensic Virtual Memory Creation, Dispatch and Analysis
DDoS	Distributed Denial of Service
DEofNVMT	Dual Effect of Negative Virtual Memory Trust

DEofPVMT	Dual Effect of Positive Virtual Memory Trust
DEofVMT	Dual Effect of Virtual Memory Trust
EFF	Effective
EFY	Efficient
FN	False Negative
FNR	False Negative Rate
FP	False Positive
FPR	False Positive Rate
FPT	False Positive Time
FUL	Functional
FVM	Forensic VM
GUI	Graphical User Interface
HB-IDS	Host-Based Intrusion Detection System
HTTP	HyperText Transfer Protocol
HVM	Hardware Assisted Virtual Machine
IaaS	Infrastructure-as-a-service
IDS	Intrusion Detection System
IRC	Internet Relay Chat
M&A	Monitoring and Analysis
M&D	Monitoring and Detection

MCC	Matthews' Correlation Coefficient
NB-IDS	Network-Based Intrusion Detection System
NIST	National Institute of Standards and Technology
OTcl	Object Tool Command Language (OTcl)
P2P	Peer-2-peer
PaaS	Platform-as-a-service
PCA	Principal Component Analysis
pNIC	Physical Network Interface Card
pSwitch	Physical Switch
PV	Para Virtualisation
RL	Risk Level
S-OH	Storage Overhead
S-SF	Symptom Specific Forensic Virtual Machines
S-VMD	Suspected Virtual Memory Detector
S-VMT-T	Suspected Virtual Memory Trust Table
SaaS	Software-as-a-service
SCY	Scalable
TN	True Negative
TNR	True Negative Rate
TNT	True Negative Time

TP	True Positive
TPR	True Positive Rate
V-SF	Virtual Machine Specific Forensic Virtual Machines
VMI	Virtual Memory Introspection
VMT-Exc	Virtual Memory Trust Value Exchange
VMT-I	Virtual Memory Trust Value Initialization
VMT-T	Virtual Memory Trust Table
VMT-U	Virtual Memory Trust Value Update
VMT-VE	Virtual Memory Trust Value Estimation
vNIC	Virtual Network Interface Card
vSwitch	Virtual Switch
WPAR	Window Size

Notations

botVM	A VMWatcher infected by a botcloud
$\mathbf{CredV_i}$	Credibility value of VMWatcher i
PARfPVM	Packet Arrival Rate from PeerVM
$\mathbf{PARfPVM_{curr}/_{prev}}$	The current/previous Packet Arrival Rate from PeerVM
standardVM	A VMWatcher which is not involved in the attack process
$\mathbf{T_{comm}}$	Communication time between VMWatcher and its peerVM
$\mathbf{T_{procc}}$	Processing time of a peerVM to process a request & respond
targetVM	A VMWatcher attacked by a botcloud
$\mathbf{VMT_i{}^j}$	The trust value of VM_i given by VM_j
$\mathbf{VMT_i{}^j{}_{max}}$	The maximum trust value of VM_i given by VM_j
$\mathbf{VMT_i{}^j{}_{min}}$	The minimum trust value of VM_i given by VM_j
$\mathbf{W_{curr\text{-}1}}$	The previous time window
$\mathbf{W_{curr}}$	The current time window

Abstract

Let's Look Out For Each Other: A Distributed Framework for Botcloud Detection

Twisha

A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy, 2022

Cloud Computing (CC) has gained increasing attention from the industry for on-demand rapid provisioning of shared pool of resources and services (R&S). This set of R&S is configured as per the users' requirement and are accessible through virtual machines (VMs). This infrastructure enables VMs to access user data, thus increasing the risk of losing it, particularly given the fact that VMs maybe much more vulnerable to theft or loss in comparison with conventional computing devices such as workstations. Therefore, more stringent security provision is needed in this environment.

The initial literature study in the topic of VM security shows that VMs are susceptible to a number of security attacks such as distributed denial of service (DDoS), side channel, man-in-the middle attacks and more, with DDoS as the most common. These attacks can be performed by a malware-infected VM residing on the same system or by an outsider. The attacks performed from within the system are difficult to track. Moreover, the strength of these attacks increases drastically when a group of malicious VMs attack simultaneously. These are the VMs infected by the same malware (bot) known as botVMs and the attack is known as botCloud attack. The high impact of a botCloud attack has motivated us to investigate how to strengthen the security of VMs and minimize the effect of such attacks. To this end, the thesis makes the following novel contributions.

The thesis proposes and evaluates a novel BotVM Detection (bVMD) framework to detect a set of botVMs in an effective and efficient manner. The novelty of the framework lies in that it uses a two-staged approach to botCloud detection: in Stage-1, a peer-VM mutual monitoring based, suspected botVM identification method is used to identify suspected malicious VMs (S-VMs) and, in Stage-2, a detailed examination of run-time state is carried out on each identified suspected

botVM. This two-staged approach to botCloud detection reduces the number of VMs on which run-time state examinations are carried out, thus reducing overhead costs and making the detection more efficient. The peer-VM mutual monitoring based, suspected botVM identification method allows each VM being watched by any peer VM, potentially increasing the true positive detection rate, thus making the detection more effective. In addition, the run-time state examination of each S-VM is done by using an improved method to minimize the number of examiners, thus reducing the overhead costs.
The bVMD framework consists of four types of components: (i) a VMWatcher, which is a VM monitoring component residing in each VM monitoring and recording the behaviour of each peer VM communicating with this VM, (ii) an S-VM Detector (S-VMD), an analysis component, collecting data from VMWatchers, analysing the collected data to identify any S-VMs, (iii) a Forensic VM (FVM), a mini-VM dispatched to any identified S-VM to analyse its run-time state and (iv) a BotVM Detector (BotVMD), the component that controls the FVMs and makes decision as which S-VM is confirmed as a botVM. Components (i) and (ii) work in Stage-1 and (iii) and (iv) in Stage-2. In proposing this framework, we have also studied different parameter value settings and feature extraction techniques to increase detection accuracy, while, at the same time, minimizing overhead costs. The bVMD framework is implemented and evaluated using the Omnet++ simulation tool. The evaluation results are compared against the most relevant work in the literature. The simulation study shows that bVMD outperforms the relevant protocol in terms of the true positive and false negative detection. These enhancements make bVMD more effective in detecting a group of botVM(s). In addition, the bVMD framework minimizes the associated overhead costs, thus improving the efficiency of the system.

Declaration

No portion of this work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

First of all, I would thank my supervisor, Dr Ning Zhang, for being so patient with me throughout my PhD. I really appreciate her for keeping me motivated throughout with her constant support, guidance and valuable advice. I would also thank my co-supervisor Dr Mustafa and my advisor Professor Uli Sattler for their help and support in my research. I also gratefully acknowledge the University of Manchester for providing me the opportunity and the funding to pursue my PhD at this prestigious institute.
Secondly, I thank my lovely parents, Mulkh Raj and Suneeta, for their endless support all through my life and being there with me in all my decisions. I would also thank my husband, Ashwani, and my daughter, Myra, for being my greatest supporter throughout this journey.
Thirdly, I thank Soontorn and Tahani for being amazing friends, listening to all my nagging and supporting me over these years.

Dedication

To my family, mom, dad, hubby and my little one

Chapter 1

Introduction

1.1 Background

This section provides the background information for the work reported in this thesis, namely cloud computing and its security concerns.

1.1.1 Cloud Computing

Cloud is a model for enabling 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 [1]. It supports three service models, namely Cloud Software as a Service (SaaS), Cloud Platform as a Service (PaaS) and Cloud Infrastructure as a Service (IaaS). SaaS is a software distribution model in which a cloud provider hosts applications and makes them available to end users over the internet [2]. PaaS is a platform distribution model which provides a broad set of cloud-based application infrastructure and middleware (AIM) resources through cloud [3]. IaaS is an infrastructure distribution model which provisions the hardware resources such as storage space, computing power and network communication capabilities as on-demand services. According to recent research, the global cloud computing industry size is expected to increase at

16.3 percent Compound Annual Growth Rate (CAGR) from USD 445.3 billion in 2021 to USD 947.3 billion in 2026, indicating mass adoption of cloud computing by corporate giants to unicorn start-ups [4]. As the figures suggest, an increasing number of businesses are moving to cloud and it is expected to continue to grow with time. This is because of the feature of massive cost savings combined with increased IT agility. Despite these positives, there are some barriers holding back the companies for faster adoption of cloud computing. According to a report by The National Institute of Standards and Technology (NIST), the major barriers to broader cloud adoption are security, interoperability and portability [5]. A survey done by Cloudsecurity Insiders in 2020 specified organizations' lack of qualified staff (39%) as the biggest impediment to faster adoption of cloud computing, followed by data security issues (34%) and legal & regulatory compliance (32%) [6]. In addition, Phaphoom in [7] did a logistic regression analysis on the criticality of security concerns in cloud computing and the results show that it is responsible for an up to 26-fold increase in the non-adoption likelihood. In the next section, we discuss about the security concerns in cloud computing.

1.1.2 Security Concerns in Cloud

Using cloud computing is very advantageous but is still susceptible to attacks. Based on our literature study in the topic area of cloud security issues [8], we classify them in four categories, i.e. Data, Access, Compliance and Network. Data issues [9] refer to issues caused in handling stored data, like data leakage, redundancy, etc. Access issues [10] refer to issues raised due to inauthentic access of cloud services and data. Compliance issues [11] refer to issues raised due to lack of security standards, legal aspects etc. Network issues [12] refer to the attacks mounted by the help of cloud network, i.e. attacks like DDoS, flooding, etc.

DDoS attack is one of the major security issues in a cloud environment. A report in 2013 on the rate of DDoS attacks [13], reported that in a 12-month period, 76% of the participating companies experienced DDoS attack. It concluded that cloud services are very tempting for DDoS attackers and it stated that as more cloud services come into use, DDoS attacks on them will become more commonplace. A report [14] published by Cloud Security Alliance in 2013 identified DoS as fifth critical threat to cloud security. However, the same research group published a similar report [15] in 2020 wherein DoS was not included as one of the critical threats to cloud security, whereas it was identified as one of the technical impacts of the threats. Also, in another report [16] in 2022 by the same research group, DoS was not identified in the top ten critical threats. In addition, according to a research done by a cloud anti-DDoS vendor, Link11 reported that the frequency of DDoS attacks nearly doubled from a period of February to September 2020. These attacks were on an average 98% higher than in the same period in 2019 [17]. A 2021 report [18] also showed a massive 17% increase in the DDoS attacks from quarter 1 to quarter 3 in 2021. As the figures suggest, DDoS attacks are increasing rapidly in cloud environment. This can lead to unavailability of cloud resources and connectivity issues. These can deactivate the cloud service, which can inflict immense business and financial losses for customers [19].

Generally, DDoS attackers can be categorized in two groups [20]: 1) Outsider Attackers and 2) Insider Attackers. Outsider Attackers are those which reside outside the network and they either have little or no internal information about the network. Insider attackers are the ones who reside within the network and thus have internal network information. Today, insider DDoS attackers pose a great challenge for the cloud environment [21]. This is because insider attacker can directly communicate with other reachable benign nodes of the network as being legitimate nodes of the network [22].

In a cloud environment, a VM infected by a malware, namely botVM, is referred to as a malicious insider attacker. In addition to DDoS attacks [23], these botVMs can launch phishing attacks [24], pay-per-click fraud attacks [25], etc. Moreover, these botVMs can infect other benign VMs through spamming attacks [26] [27], VM escape attack [28] and side-channel attack [29]. Using side-channel attacks, a botVM might deduce current encrypted operation executing on the physical machine from a co-located benignVM, it might run a computer intensive MPI job or a data intensive job such as Hadoop/Spark.

These botVMs infect benign VMs with an aim of forming a group of botVMs and attacking simultaneously, thus increasing the attack strength. This group of botVMs which work simultaneously under the supervision of same attacker is called a botcloud or cloud-based botnets [30] [31]).

1.2 Research Motivation and Challenges

From the above discussions, it is clear that security against botcloud should be strengthened. Through our literature study of state-of-the-art in the topic area of botcloud detection, it was discovered that most of the existing solutions are designed to detect specific malicious activities. They are largely based on attack-evidence, i.e. detection decisions are made based on the data generated on the back of attacks. These solutions are not effective for early detection of botcloud. Furthermore, in terms of satisfying the efficiency requirements, thus making a botcloud detection solution more scalable, there is still much room for improvement. These observations have motivated the research conducted in this thesis. The research is aimed at investigating how to strengthen botcloud detection in terms of effectiveness, efficiency and scalability. We answer this question by designing and evaluating a two-stage detection framework with stage-1 focussing on network based anomaly detection to identify a set of most suspicious malicious VMs and stage-2 focussing on signature-based detection wherein a forensic analysis of the identified suspected malicious VM is conducted. In doing so, the following challenging issues are identified.

- **Trade-off between Accuracy and Overheads**: A two-stage detection framework might increase the effectiveness (in terms of detection accuracy) of the solution. However, the additional dual analysis might introduce additional complexity, thus decreasing the efficiency of the solution. Therefore, consideration should be given as how to optimize the trade-off between accuracy and overheads. This may be possible by adjusting the overheads as much as possible with as high accuracy as possible. This research aims to investigate how to facilitate botnet detections in the context such that the accuracy of the detection can be as high as possible and/or the overhead incurred can be as low as possible. At each stage of the design, we will examine alternative ways of performing the operations, including the selection of parameter values, and make the decisions accordingly.

- **Minimising Communication Overheads**: A two-stage detection framework might introduce additional communication overheads for exchanging data between the stages. Therefore, consideration should be given as how to minimize this communication overhead as much as possible.

1.3 Research Aim and Objectives

The aim of this research is to investigate how to effectively and efficiently detect members of a botcloud (botVM) in a single cloud environment. By effective, we mean that our designed solution should be able to identify botVMs as early as possible and the detection rate should be as high as possible. By efficient, we mean that the costs incurred in botVM detection should be as low as possible, and these costs are al communication cost and storage cost. This aim is supported by the following objectives.

1. To investigate and analyse the architecture and working of botcloud, its way of spreading and attacking the cloud system.
2. To investigate and critically analyse related work in the topic area of botcloud detection with the aim to identify gaps in knowledge and investigate novel measures and ideas to improve existing solutions in terms of efficiency, effectiveness and scalability
3. To analyse and specify requirements for an effective, efficient and scalable botcloud detection solution.
4. To design a two-stage botcloud detection solution to detect botVMs in a cloud environment with as high accuracy, and as low cost, as possible. In designing this system, we also investigate the effects of various parameter settings on the performance of the designed solution.
5. To evaluate the performance of botcloud detection solution. In addition, analyse the attacks that can be mounted on the solution and evaluate its performance under these attack scenarios.
6. Collect scientific evidence to demonstrate the levels of effectiveness, efficiency and scalability of the designed solution. The effectiveness can be estimated by calculating the true positive, true negative, false positive and false negative rates. The efficiency of the solution can be estimated by calculating the communication overheads and the storage overheads involved. The scalability of the solution can be estimated by analysing the performance of the solution with increase in the number of VMs. The results obtained are then compared with the most related solutions.

1.4 Research Methodology

The research methodology followed in this research comprises of three key components: literature survey and critical analysis of the related work, system design, and implementation and evaluation.

1.4.1 Literature Review

The first task carried out in this research was to do an in-depth study of the related work in the literature. We started by investigating the topic area of botcloud detection. The purpose of this study was to become familiar with the current detection techniques used for detecting botcloud. From this study, it became apparent that an effective, efficient and scalable solution for detecting botcloud is required. The next step was to critically analyse the existing solutions to identify their strengths and limitations, with the aim to build our solution on the strengths of existing solutions but overcome their limitations. The insights gained from the analysis have led us to the design of our effective, efficient and scalable botVM detection solution. In addition, literature review was carried on throughout the duration of this research. As new work was published, it was reviewed, and necessary findings were taken into consideration. Performing the literature review and critically analysing the literature satisfied our objectives 1, 2 and 3.

1.4.2 Design Work

Following the literature review, several research gaps were identified. Based on these identified gaps and the research aims, measures and ideas were formed to design the solution that addresses these gaps. The measures and ideas were repeatedly refined by considering input from the existing work and by our progressively insightful thinking towards the research problem. Careful considerations were given to reduce any additional overheads imposed in the proposed solution. At the conclusion of this stage, a novel architecture with four design components was proposed. The solution is called bVMD Framework and its components are

VMWatcher, Suspected VM Detector (S-VMD), Bot VM Detector (Bot-VMD) and Forensic VM (FVM). The design of the framework and the components satisfied our objectives 3 and 4.

1.4.3 Implementation and Evaluation

The next stage of our research was to implement and evaluate the designed solution. The performance of the framework was evaluated using simulation. The implementation of the simulation-based evaluations was carried out using Omnet++ Programming platform [32]. Before the implementation, it was first necessary to perform two tasks (i) define the evaluation metrics, and (ii) design the evaluation methods. Following these tasks, the simulations were implemented using the evaluation methods to provide accurate measures of the metrics. The simulations were carried out in three stages. In the first stage, the simulation was run under various parameter settings to evaluate the impacts of different parameter value settings on the performance of bVMD Framework. In the second stage, the simulation was carried out to evaluate the performance of the first stage of the bVMD Framework. In the third stage, the simulation was carried out to evaluate the performance of bVMD Framework and to compare it with the most relevant work to demonstrate the merits of the proposed framework over the related work. The analysis and evaluations satisfy our objectives 5 and 6. Conclusions were drawn from the evaluations of the designed solution, and directions for future research were identified.

1.5 Novel Contributions

The research work presented in this thesis has led to the following novel contributions.

Novel Contribution 1: Novel framework for detecting botVMs (bVMD Framework)
This is a novel botVM Detection Framework which aims to detect botVMs in an

effective, efficient and scalable manner. This was achieved by enabling a two-stage detection system and analysing a VM twice before accusing it as being a botVM. The performance of the bVMD Framework is evaluated in terms of detection accuracy. Storage and communication overheads are also discussed. These results are compared with the related work. The following novel contributions are carried out for supporting this novel claim.

Novel Contribution 2: The design, analysis and evaluation of a novel trust-based VM monitoring component, called VMWatcher
This is a novel monitoring component which enables normal working VMs to monitor the communication patterns of the VMs with whom they are communicating, called peerVMs. The main novel features of VMWatcher are twofold.

- It analyses and processes the monitored data locally at each working VM, and only transmits the result of analysis. This can reduce the volume of data poured into the underlying network, thus minimizing the communication cost.

The evaluation of VMWatcher in terms of the storage and communication overheads introduced on each working VM is conducted. The design and analysis are presented in detail in Chapter 4.

Novel Contribution 3: Novel algorithm for minimizing the resources used for in-depth analysis of memory log of a VM
This is a novel algorithm which determines the number of analysers (called Forensic VM, FVM) used in second stage (of bVMD framework) for in-depth analysis of the suspected malicious VMs identified from stage-1. The main novel feature of this algorithm is that it minimizes number of FVMs used, thus improving the scalability of the bVMD solution. The evaluation of the algorithm is conducted in terms of the number of FVMs required with increase in number of VMs to be analysed. The design and analysis are presented in detail in Chapter 6.

1.6 Thesis Structure

The remainder of this thesis organised as follows. **Chapter 2** presents a detailed investigative study of botcloud detection methods and techniques. It describes the related work in the area of botcloud detection solution. Based on the identified strengths and weaknesses of the existing work, the chapter identifies research gaps in this work and makes recommendations for a way forward. **Chapter 3** presents the design of the first novel contribution of this research, i.e. bVMD framework. The measures and ideas used in the design of the bVMD framework are based on the recommendations made in Chapter 2. **Chapter 4** presents the design and evaluation of the stage-1 components of the bVMD Framework, i.e. VMWatcher and S-VMD. This chapter presents the second novel contribution of this research. The stage-1 components are evaluated against the related work. **Chapter 5** presents the design of stage-2 components of bVMD framework and the evaluation of bVMD framework. This chapter presents the third novel algorithm of this thesis. **Chapter 6** presents the results of performance of bVMD Framework. **Chapter 7** concludes this thesis and suggests directions for future research. The structure of the thesis is further illustrated in Figure 1.1.

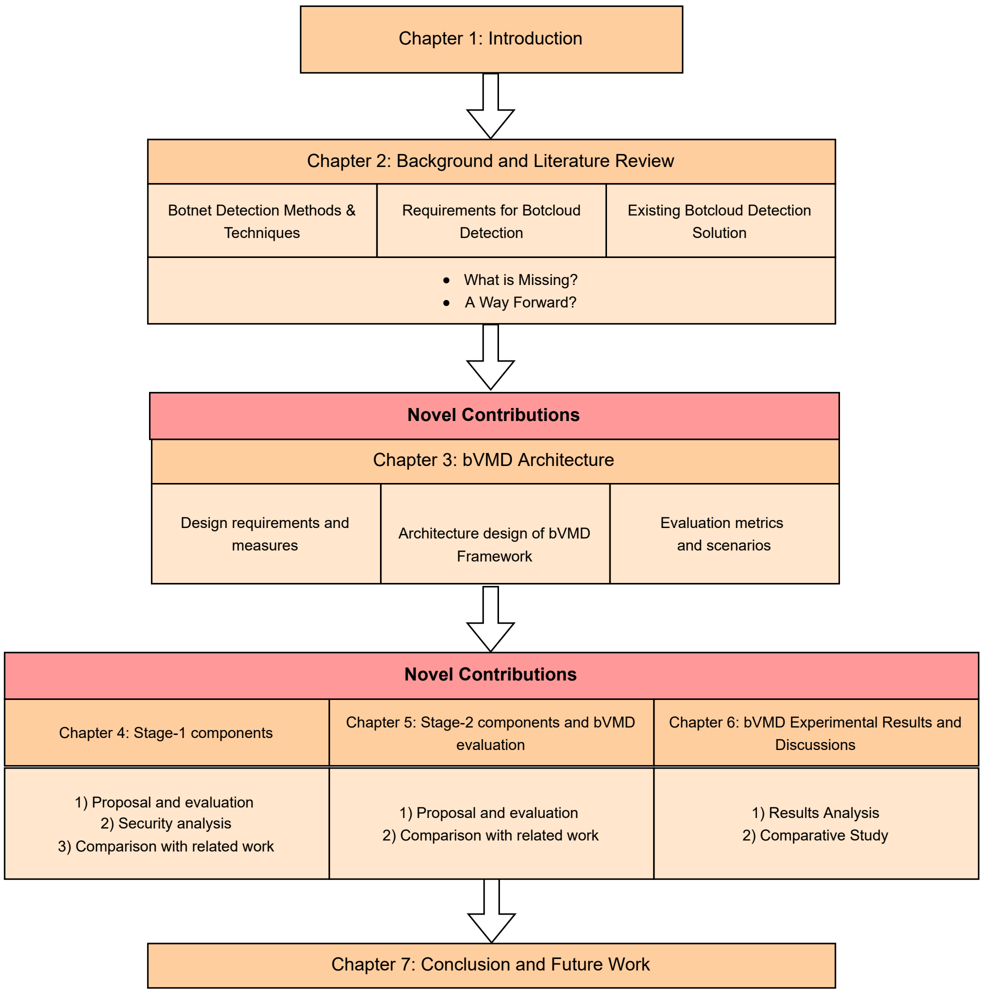

Chapter 1: Introduction
Chapter 2: Background and Literature Review
Botnet Detection Methods & Techniques
Requirements for Botcloud Detection
Existing Botcloud Detection Solution
• What is Missing?
• A Way Forward?
Novel Contributions
Chapter 3: bVMD Architecture
Design requirements and measures
Architecture design of bVMD Framework
Evaluation metrics and scenarios
Novel Contributions
Chapter 4: Stage-1 components
Chapter 5: Stage-2 components and bVMD evaluation
Chapter 6: bVMD Experimental Results and Discussions
1) Proposal and evaluation
2) Security analysis
3) Comparison with related work
1) Proposal and evaluation
2) Comparison with related work
1) Results Analysis
2) Comparative Study
Chapter 7: Conclusion and Future Work

Figure 1.1: Thesis Structure

Chapter 2

Botcloud Detection Methods: A Literature Survey

2.1 Chapter Introduction

This chapter gives a survey of the botnet detection techniques and the state-of-the-art solutions for detecting a botcloud. These techniques can be classified into Honeypots and Intrusion Detection Systems (IDS). The IDS techniques can further be classified into two groups: Signature-based and Anomaly-based IDS. This chapter critically analyses the state-of-the-art solutions to identify their strengths and limitations so that, in the design of our proposed solution, we can maintain the strengths while overcoming their weaknesses. This chapter also discusses the requirements for an effective, efficient and scalable botcloud detection solution and studies the existing solutions against these requirements. Through this study, the chapter identifies areas of improvement for an effective, efficient and scalable botcloud detection solution.

Section 2.3 gives details of a botcloud. Section 2.4 outlines the classifications of botnet detection techniques. Section 2.5 describes the requirements for an effective, efficient and scalable botcloud detection solution. Section 2.6 describes the existing botcloud detection solutions, their strengths and weaknesses. Section 2.7 outlines the gaps between the state-of-the-art solutions. Section 2.8 presents the way forward to address these gaps. Finally, Section 2.9 concludes the chapter.

2.2 Cloud Overview

A cloud is a type of parallel and distributed system consisted of a collection of inter-connected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resource(s) [33]. These virtualized computers (software or OS) are called Virtual Machines (VM(s)) [34]. They are the building blocks of a cloud and they are created & managed by a Hypervisor (Hyper-V) (also called Virtual Machine Monitor) [35]. It supports multiple VMs by virtually sharing the resources of a physical server. These includes memory, processor, network connection, hard drive, and an operating system (OS) for running programs and applications [36].

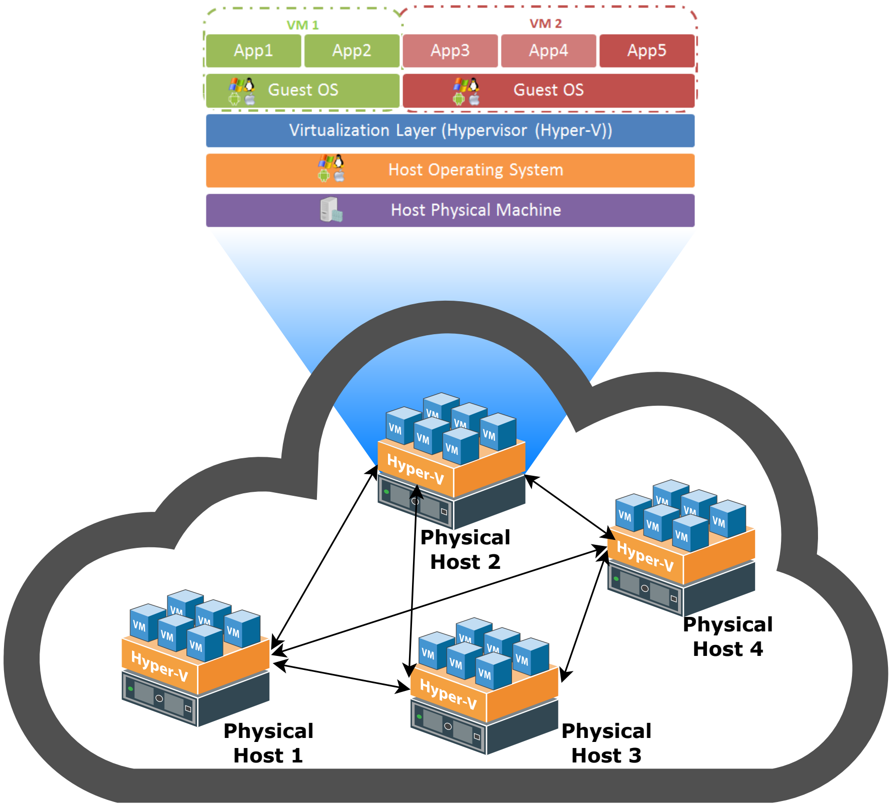


Figure 2.1: Cloud Overview

There are two main types of Virtual Machine Monitor (VMM), 'Type 1' (or 'bare metal') and 'Type 2' (or 'hosted'). A type 1 VMM acts like a lightweight operating system and runs directly on the hosts' hardware, while a type 2 VMM runs as a software layer on an operating system, like other computer programs

[37]. The most commonly deployed type of hypervisor is the type 1 or bare-metal hypervisor, where virtualization software is installed directly on the hardware where the operating system is normally installed. These VMMs provides simple primitives (start, stop, suspend, etc.) to manage VMs which are placed on a single host. However, when these VMs are spread across multiple hosts, Virtual Infrastructure Management (VIM) provides the required primitives to schedule and manage them. OpenStack and OpenVIM are the two most common VIMs used [38].

An important concept of Containerization [39] has become a major trend in software development as an alternative or companion to virtualization. It aims to encapsulate software code and all its dependencies so that it can run uniformly and consistently on any infrastructure. Containers are often referred to as 'lightweight', meaning they share the machines' operating system kernel and do not require the overhead of associating an OS within each application. Therefore, they are inherently smaller in capacity than a VM and require less start-up time, allowing far more containers to run on the same compute capacity as a single VM. The emergence of the open source Docker Engine [40] in 2013, an industry standard for containers with simple developer tools and a universal packaging approach, accelerated the adoption of this technology. In addition, Kubernetes [41] is perhaps one of the most popular container orchestration system.

2.3 Botcloud

This section first provides an overview of botcloud, followed by its communication channel. In addition, this section describes the lifecycle of a benign VM which gets infected and becomes a botVM.

2.3.1 Overview

A botcloud [42][43] is defined as a network of malicious-infected VMs (called botVMs) controlled by an attacker (called a botmaster) using Command-and-Control (C&C) communication mechanisms. These mechanisms allow botVMs to receive commands and malicious codes from the botmaster. The main objective of a botcloud is to carry out simultaneous attacks through a group of botVMs [30] [31]. These attacks could be DDoS attacks [23], phishing attacks [24], pay-per-click fraud attacks [25], etc. The strength of these botcloud attacks grows rapidly with an increase in the number of botVMs. Therefore, in addition to performing the above attacks, a botmaster also attempts to increase the number of botVMs in its botcloud network. This can be done by attacks like spamming attacks [26] [27], VM escape attacks [28] and side-channel attacks [29]. Comparing to botnet attacks which are performed in a physical environment, the botcloud attacks are easier and quicker to accomplish. This is because a botcloud can be put together in a couple of minutes just by purchasing space in the cloud, therefore making the deployment much faster as compared to traditional botnet [25] [44]. In addition, strategic placement of an attacking VM can be used by attacker to perform side-channel attacks on co-resident virtual machines [45]. According to a research done by Ristenpart [46], it was found out that just a few dollars invested in launching a VM can produce a 40% chance of placing a botVM on the same physical server as a target VM, therefore increasing the chances of side-channel attacks.

2.3.2 Communication Structures

As discussed above, botmaster [47] is a VM directly controlled by the attacker and it sends malicious commands to the botVMs [48]. To send these commands, it should be connected to the botVMs which can be done by using one of the following structures as shown in Figure 2.2.

2.3.2.1 Centralised Structure

Centralised structure [49] is the earliest of the three structures which is mostly used by a botcloud. In this structure, the botmaster is directly connected to each of the botVMs as shown in Figure 2.2 (a). It follows the traditional client and server architecture, where the botVMs work like a client and the C&C server works like a server. This C&C server receives commands from the botmaster and sends them to the botVMs. This structure is very effective in terms of the strength of attack performed. This is because the direct connection minimizes the communication time between the botmaster and botVMs, thus the commands can reach to the target quickly. In addition, as all the botVMs receive the commands at almost the same time, they can start the attack simultaneously. On the other hand, this structure has a single-point-of-failure, i.e. the C&C server. That means detection of this server can lead to all the botVMs in its network. Also, because of the direct connections, the detection of any one of the botVMs can lead to the C&C server. The major protocols used in this structure are Internet Relay Chat (IRC) [50] and HyperText Transfer Protocol (HTTP) [51].

2.3.2.2 Decentralised Structure

Decentralised structure [49] [52] follows a peer-2-peer (P2P) structure, where the botmaster is directly connected to one of the botVMs, which is, in turn, connected to one or more botVM(s) and so on, as shown in Figure 2.2(b). The commands are distributed by the botmaster to its p2p botVM and further to its p2p botVM(s) and so on. This structure is effective as compared to centralised structure as it doesn't have the disadvantage of single-point-of-failure. Also, the detection of the botmaster or any of the botVMs does not result in the failure of the entire p2p network. This is because the undetected botVMs can continue to work as usual. However, in this structure, the botVMs are slow in convergence and response. This is because of the additional time (as compared to centralised structure) taken for a command to propagate to all the botVMs. In addition, this structure is difficult to manage [53] as compared to centralised structure because it is difficult to keep a track whether all the botVMs have received the commands on time or not.

2.3.2.3 Hybrid Structure

Hybrid structure [54] [55] is the latest structure which is used by botcloud. In this structure the botmaster is connected to one or more botVMs and each botVM can work in one of the following ways: 1) as a centralised node and have a couple of botVMs as sub-nodes (centralised structure) or 2) as a p2p node and have p2p connected botVMs or 3) combination of the above ways, i.e. they act as a centralised nodes for a couple of botVMs and at the same time they have a couple of p2p botVMs. This is shown in Figure 2.1(c). Botclouds using this structure have strong inter-connected botVMs, which means that commands sent by a botmaster can reach the botVMs through different routes. If one route is seized, another route can be used to send these commands, thus making it hard to shut down the botVMs.

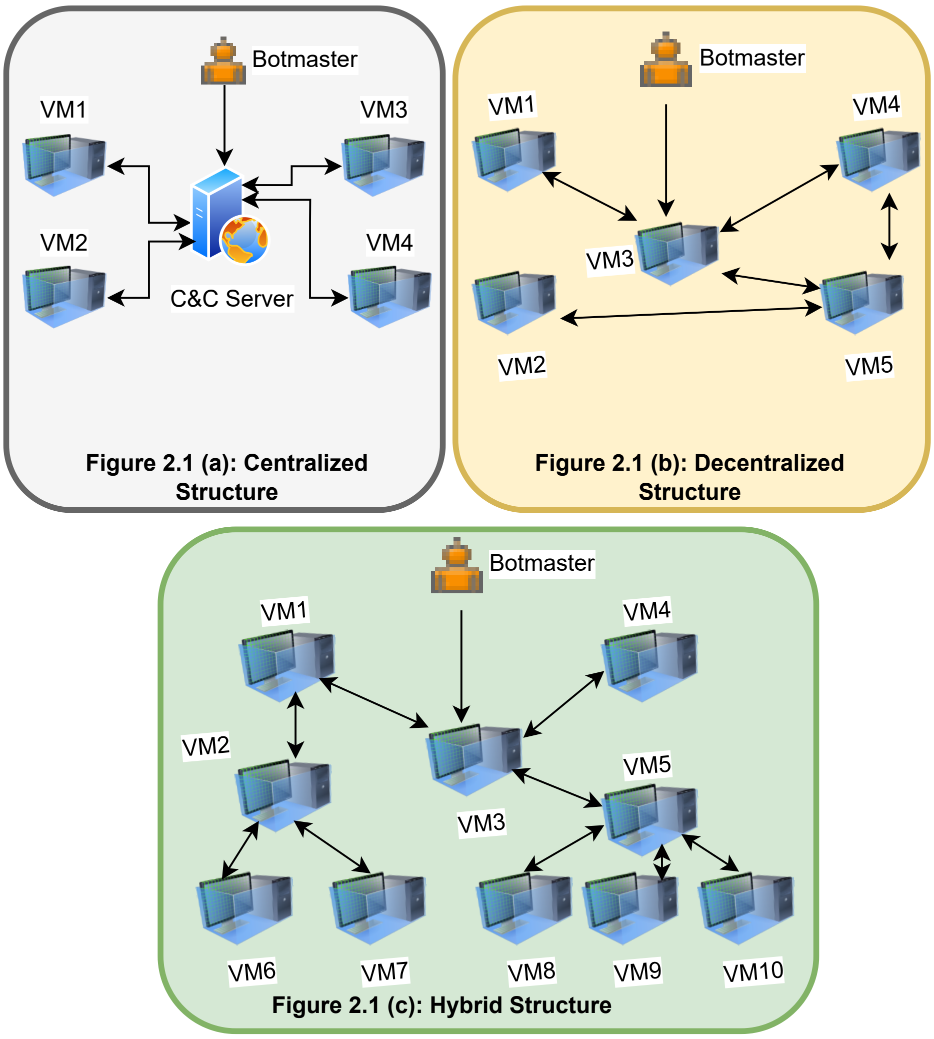

Botmaster
VM1
VM3
VM2
VM4
C&C Server
Figure 2.1 (a): Centralized Structure
Botmaster
VM1
VM4
VM3
VM2
VM5
Figure 2.1 (b): Decentralized Structure
Botmaster
VM1
VM4
VM2
VM5
VM3
VM6
VM7
VM8
VM9
VM10
Figure 2.1 (c): Hybrid Structure

Figure 2.2: Botcloud Communication Structures

The functioning of the botcloud is shown in Figure 2.3. The botmaster starts the botcloud by first deciding a communication structure to be used, the malicious attacks to be performed and the targetVM to be attacked. As per these decisions, it sends appropriate commands to the botVMs. Once the botVMs receive the commands, it circulates the commands among its p2p botVMs (if required) and then performs attacks such as DDoS, spamming attack, etc. on the targetVM(s).

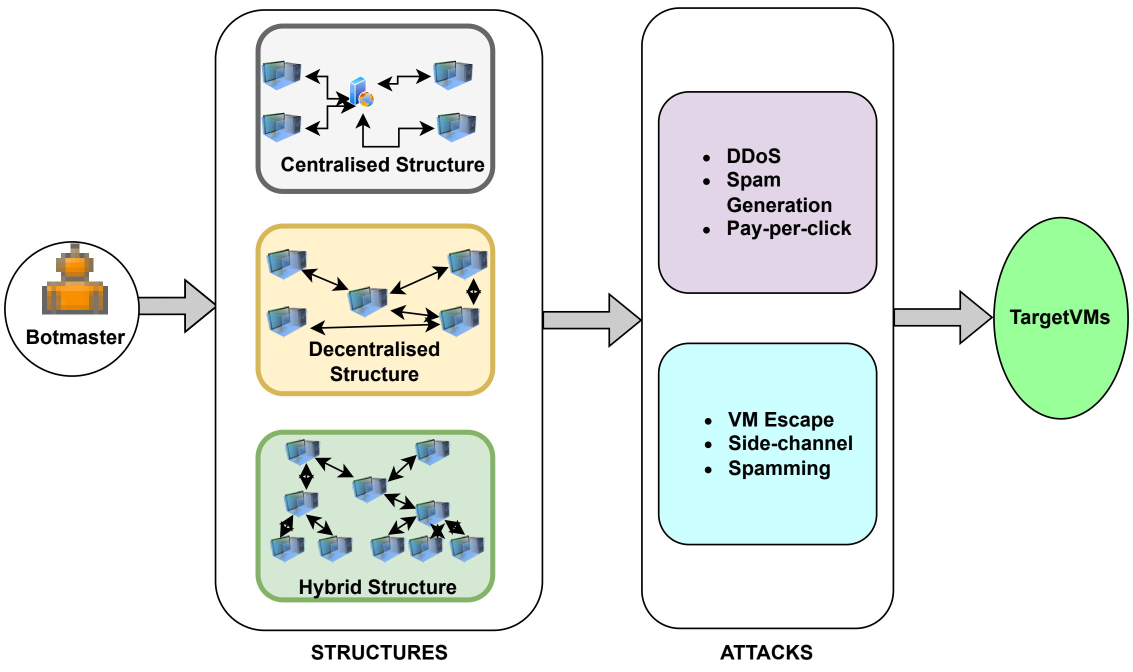


Figure 2.3: Botcloud Functioning

2.3.3 Botcloud Lifecycle

As shown in Figure 2.4, the lifecycle [56] of infecting a benign VM is divided into four stages.

*Stage*1. **Infection stage**: In this stage, a botmaster or a botVM exploits a vulnerability of a benign VM by injecting a malicious script. A benign VM might get infected and become a botVM after accessing that malicious script.

*Stage*2. **Connection stage**: In the stage, the newly formed botVM attempts to establish a connection with the C&C server (botmaster) through a variety of methods. Once it is able to establish a connection, it becomes a part of a botcloud.

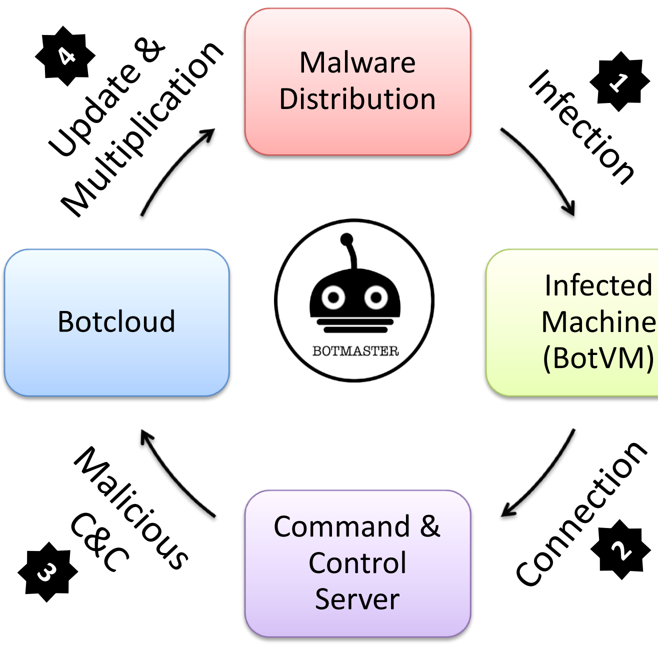


Figure 2.4: Botcloud Lifecycle

*Stage*3. **Commands from botmaster**: In this stage, the botVM request for commands from the botmaster to begin malicious activities.

*Stage*4. **Update & multiplication**: In this stage, the botVM is commanded to update its scripts, typically to defend against new attacks and improve their functionality. In addition, they are instructed to infect other benign VMs.

2.4 Botnet Detection Techniques

Botnet detection techniques can be classified into two categories [57], honeypots and Intrusion Detection System (IDS).

2.4.1 Honeypots

A honeypot [58][59] is a tool that is used with the purpose of being attacked and possibly compromised [60]. They are deliberately allowed to be compromised, and the attack is then analysed. They use special software which suspects every packet transmitted through it to collect data about the procedures and tools used by attacker. As honeypots are designed to collect network traffic passing through its own node, the data set for analysis is smaller as compared to traffic data for the whole network. This minimizes the computational overhead involved in analysing the network traffic data. However, they have limited effectiveness as they can only detect a malicious VM when it directly interacts with the honeypot. In addition, if the honeypot is compromised, it can be used for attacks on other VMs in the network.

Freiling [58] uses honeypots to detect a botnet. To do this, the solution deploys honeypot software on network resources like computers, routers, switches, etc. These resources are probed, attacked and compromised. The compromised resources regularly collect data about the system behaviour with the help of special software and facilitate automatic post-incident forensic analysis. From the results of the automatic analysis, important information necessary to observe malicious actions of a botnet is collected. This solution is implemented for an IRC-based botnet.

2.4.2 Intrusion Detection System

Intrusion in a system or network is defined as attempts to compromise the confidentiality, integrity, availability, or to bypass the security mechanisms of the system or network. Intrusion detection is the process of monitoring the events occurring in a system or a network and analysing them for signs of intrusions [61]. An Intrusion Detection System (IDS) [62] [63] is a software application or a hardware machine which aims to detect intrusions by observing and inspecting the vulnerabilities in network traffic or on the host. IDS can be classified into two types of approaches, signature-based and anomaly-based.

2.4.2.1 Signature-Based IDS

A signature-based IDS aims at recognizing unique trends and characteristics exhibited by a malware; these are known as signatures. It monitors the network data to identify a signature. This identified signature is compared with the database

of known signatures and any suspicious behaviour is flagged. This approach is effective with attacks performed by known malware (i.e. the malware whose signature is known). In addition, the database used to store the known signatures might introduce additional overheads as it needs to be regularly updated with new signatures.
Authors in [62] proposed a signature-based passive monitoring botnet detection system. It monitors network traffic to identify a set of signatures, i.e. unusual or suspicious IRC nicknames, IRC servers, and uncommon server ports. These signatures are identified and compared with the stored ones. They have proved that the system is effective in early detection of infected hosts. However, the system is limited to detect bots with known signatures, and they might not be effective in detecting non-IRC traffic or encrypted traffic.

2.4.2.2 Anomaly-Based IDS

An anomaly-based (or behaviour-based) detection approach [64] aims at identifying unusual VM behaviour patterns that do not conform to expected behaviour, called outliers. These unusual patterns could be network traffic irregularities like traffic passing through unusual ports, increased traffic volume, etc. To detect the outliers, firstly a baseline profile of normal VM activity is created and, thereafter, a deviation from normal VM activity is treated as malicious. This technique is more effective than signature-based technique as it can detect both known and unknown attacks. However, the detection results are based on the identified baseline profile. This profile may not always be effective as a botVM can hide itself by working within the identified baseline profile. Anomaly-based approaches are further divided into Host-Based and Network-Based approaches.

2.4.2.2.1 Host-Based IDS A Host-Based IDS (HB-IDS) [65] is designed to monitor individual hosts to identify any suspicious action. It can aim to either monitor the network traffic at the host level or monitor the file system on the host, i.e. logon processes, running processes, system logs, etc. This approach might be more effective than a Network-Based approach in scenarios where the network data is encrypted. However, this approach is not scalable, as each host requires a separate IDS machine. In addition, this approach might use the computing resources of the hosts they are monitoring, therefore inflicting a performance cost on the monitored systems. In a cloud environment, a botcloud detection system based on HB-IDS can either be placed at the hypervisor, or at the VM level.

IDS placed at the hypervisor In this approach, the HB-IDS is placed at the hypervisor. It is responsible to monitor network traffic or file system at the hypervisor to detect any suspicious activity. As there is always a single hypervisor in a host, so this approach requires only one IDS. This might minimize the overheads as compared to an approach where there is an IDS for each VM, however analysis of all the network data at a central server (hypervisor) might have a disadvantage of a single-point-of-failure.

IDS placed at the VM In this approach, the HB-IDS is placed at the VM level. The HB-IDS can use the following two methods: (1) Centralised Method: this involves a HB-IDS to monitor all or a group of VMs or (2) Distributed Method: this involves dedicated HB-IDS for each VM. Method-2 is more effective than method-1 because, with method-2, each VM is being monitored by a dedicated detection system. However, this option introduces higher costs as compared to method-1. In addition, method-2 involves multiple detection systems, which might increase the overall attack surface. On the other hand, as method-1 is based on a centralised approach, it might have a disadvantage of single-point-of-failure.

Authors in [65] proposed a HB-IDS, called BotSwat, which relies on command-response behaviour. It looks for possible remote bot initiation by tracking the programs that use data from unreliable network sources (tainted data). This tainted data is tracked as it propagates via dynamic library calls to other memory regions. They have specified a set of gate functions, which are system calls used in malicious bot activity. When the tainted data is passed as arguments to the specified gate functions, botnet infection is identified.
Authors in [53] proposed a HB-IDS, called BotTracer, which aims to detect three stages of a bot with the assistance of VM techniques. These stages are start-up,

preparation and attack. The BotTracer identify the following features during each stage: 1) A bot can be started automatically, therefore the BotTracer looks for VMs which work without any human interactions, especially those with networking activities, 2) after start-up, a bot C&C channel should be established with the botmaster, therefore the BotTracer captures the data from used automatic communication channels and compare them so known characteristics of bot C&C channels, and 3) a bot performs local or remote attacks sooner or later, therefore the detection system constantly monitors system-level activities and traffic patterns of the identified processes. This continuously monitoring of the vulnerabilities in the system calls for potential botnet activity might involve computational overhead.

2.4.2.2.2 Network-Based IDS A Network-based IDS (NB-IDS) [43] is designed to monitor the network packets to identify any suspicious activity. To monitor these network packets, sensors or hosts are placed on various points of a network. These network packets captured by a host can either be analysed at a centralised server or it can be analysed locally at the host. The NB-IDS is a cost-efficient approach as compared to HB-IDS, as a few well-placed monitors (sensors or hosts) can monitor a large network. However, capturing and analysing all the network packets introduces computational overheads. Also, the detection system should be able to quickly analyse the network packets to improve effectiveness of the solution.
The NB-IDS approach can be classified into two methods, active monitoring and passive monitoring methods.

Active Network Monitoring In this method, the VMs are continuously monitored in real-time. This method can be applied by injecting test packets into the network, followed by monitoring the network behaviour. This method is effective for quicker detection of malware as it benefits from real-time information. However, injection of test packets during this method can introduce additional traffic into the underlying network, thus increasing communication overhead.
Authors [66] proposed a botnet detection system based on active monitoring. It is based on the process of extracting and analysing flow characteristics, such as

bandwidth and packet timing. It classifies the traffic into groups that are likely to be a part of botnet. Then the traffic is correlated to observe common communication patterns that can lead to the detection of botnet activity. A set of 1.34 million real-time flows over a span of 4-months was examined by the authors and evidence of botnet activity was collected.

Passive Network Monitoring In this method, the live network traffic is captured and then analysed for malicious activities. This method involves three components, one for monitoring the network traffic, the second for storing the monitored data and the third for analysing the stored data. These components can be deployed in the following two ways.

- **Centralised Deployment Structure**: In this structure, the monitored data is stored in a central server, where it is further analysed. This structure has a disadvantage of a single-point-of-failure due to a centralized server. Also, with increase in the size of monitored data, the central server might get burdened, and take additional time for the detection process.
- **Distributed Deployment Structure**: In this structure, the monitored data is stored in separate local servers for further analysis. Each local server has a dedicated component to analyse the data. These servers work in parallel and could complete the detection process quicker as compared to the centralised structure. However, with an increase in the number of VMs, the numbers of servers also increase. This increase in the number of servers could result in 1) decrease in the cost-efficiency of the solution with scalability and 2) increase in the attack surface of the solution, thus making it less secure.

The passive monitoring method in NB-IDS can be applied using a number of protocols and applications such as machine leaning, data mining, IRC protocol etc. [53] Some of the research work using these protocols and applications is discussed below.

Authors in [67] proposed a detection system that focuses on detecting

P2P bots by using network traffic behaviours. A number of features are extracted from the network traffic and these are analysed using five machine learning techniques, namely Support Vector Machine (SVM), Artificial Neural Network (ANN), Nearest Neighbours Classifier (NNC), Gaussian Based Classifier (GBC), and Naive Bayesian Classifier (NBC). They specify three requirements for online botnet detection, namely adaptability, novelty detection, and early detection. The results show that none of these techniques satisfies the requirements of an online botnet detection framework.

Authors in [68] proposed a novel black-box unsupervised behaviour learning and anomaly prediction system for Infrastructure-as-a-Service (IaaS) clouds. It leverages self-organizing maps learning techniques to capture emerging system behaviours and predict unknown anomalies. It achieved high prediction accuracy with up to 98% true positive rate and 1.7% false positive rate and raise advance alarms with up to 47 seconds lead time.

Authors in [69] proposed a passive monitoring botnet detection system, called BotHunter. It is capable of detecting bots regardless of the C&C structure and network protocol. It consists of a correlation engine that aims at detecting specific stages of the malware infection process, such as inbound scanning, exploit usage, etc.

Authors in [70] proposed a network-based botnet detection system, called BotSniffer, for detecting bots using HTTP and IRC protocols. It is independent of botnet signature and C&C server address. It is based on spatial-temporal correlation, which means that bots belonging to same botnet perform identical activities. These activities include coordinated communication, propagation & attack and fraudulent activities. These spatial-temporal correlations are identified in the network traffic and statistical algorithms are used to detect botnets. The results show that BotSniffer can detect bots with high accuracy and has a low false positive rate.

Authors in [71] proposed a data mining-based botnet detection framework, called BotMiner. It is independent of the botnet C&C structure and protocol. It aims to detect groups of compromised machines that are a part of a botnet. To achieve this, the detection system clusters similar communication traffic and similar malicious traffic. Then it performs cross-cluster correlation to identify the hosts that share both similar communication patterns and similar malicious activity patterns. These hosts are assumed to be bots in the monitored network. The results show that BotMiner has a high detection rate and a low false positive rate.

Authors in [72] proposed a statistical-based passive-monitoring botnet detection system. It is designed to identify P2P botnets, even in the case when malicious activities may not be observable. To achieve this, first it identifies all hosts involve in a P2P communication. These P2P communications are used to derive statistical fingerprints, and these are used to distinguish between hosts that are part of legitimate P2P networks and P2P bots. The results confirm that the proposed system can detect stealthy P2P bots with a high detection rate and a low positive rate.

2.5 Requirement Specification for a Botcloud Detection Solution

Through literature research, the following high-level requirements are required that are necessary to detect a botcloud in an effective, efficient and scalable manner.

Req.1. **Functional (FUL)**: It should be able to detect botVMs which launch known and unknown attacks. In addition, the detection system should be independent of inter-botVM communications.

Req.2. **Effective (EFF)**: It should be able to detect a botVM as early as possible in botcloud lifecycle.

Req.3. **Efficient (EFY)**: It should be able to detect a botVM with as low overheads as possible.

Req.4. **Scalable (SCY)**: It should be efficient in protecting a large number of VMs.

It should be mentioned that the requirements introduced in this chapter are the high-level requirements for a botcloud detection solution. More specific requirements are specified in upcoming chapter. After identifying the requirements for botcloud detection solution, it is important to investigate and analyse the related work (what has been done so far) with regards to these requirements.

2.6 Critical Analysis of Existing Botcloud Detection Methods

This section gives a critical analysis of the botcloud detection solutions proposed in literature, highlighting their strengths and weaknesses with respect to the requirements specified in Section 2.5. The botcloud detection solutions can be divided into Network-based Intrusion Detection System (NB-IDS) and Host-based Intrusion Detection System (HB-IDS).

2.6.1 Network-based Botcloud Detection Solutions

These are solutions based on Network-based intrusion detection system (NB-IDS).

*Sol*1. **MAP_REDUCE**: J. Francois [73] proposed a botcloud detection solution that uses an anomaly-based detection system (i.e. one of the NB-IDS). It is a distributed computing framework that leverages a host dependency model and an adapted PageRank algorithm. The solution follows a passive monitoring scheme and captures network data from all the VMs. This captured data is stored at a central processing unit, called the Netflow Collector. The data is further analysed for determining connections between VMs, followed by forming a dependency graph. This graph is fed into a PageRank algorithm to determine densely connected VMs, as the solution assumes that densely connected VMs are botVMs.
The use of a PageRank algorithm in this solution makes it highly scalable with large number of VMs. The results show that this solution minimizes the computation overhead required for large number of network connections, i.e. for more than 1 million connections. The results also show a high level

of detection with a low false positive rate, i.e. 3%. However, the solution might have high false negative rate. This is because of their assumption about a densely connected VM being a botVM. This may not be always true as p2p nodes are also densely connected. In addition, using a central server to collect the network flow of VMs may introduce a high level of overheads affecting the scalability of the solution. The central server can also become a single-point-of-failure.

*Sol*2. **CAN'T HIDE**: Baohui [74] proposed a method for detection of DDoS attacks generated by a botcloud, called srcTrace. The system firstly identifies the flow of attacks and then traceback the malicious processes based on the attack flow address information. The system uses two components: Bot-chase and MP-Trace and relies on the analysis of traffic flows' entropy variation. The experimental results show that srcTrac can traceback malware, reducing the impacts upon cloud tenants and attack targets.

*Sol*3. **EXTRUSION**: TulasiRam [75] proposed an Extrusion detection system (EDS) based on NB-IDS. It aims to detect DDoS attacks by analysing and detecting anomalous behaviour in the communication pattern of VMs. To do so, the network packets are collected for all the VMs at a central server and a traffic dispersion graph is formed showing the network flow between the VMs. This graph is used to extract communication patterns to identify any outliers, i.e. anomalous behaviour. The solution may not be efficient in handling a huge amount of network data generated by an increasing number of VMs, thus it may have scalability issues [76]. This is because a central server may get overburdened and can act as a single-point-of-failure. In addition, the experimental evaluation of the method is incomplete.

*Sol*4. **COGNITIVE**: Kebande [77] proposed a botcloud detection approach based on NB-IDS which uses Artificial Immune System (AIS) to identify a malicious activity pattern. To do so, AIS detectors monitor network packets of all the VMs and store them at a central server, followed by analysing activity patterns. Then the VMs with similar activity patterns are recognized and are marked as botVMs, because of their observation that botVMs originating from a particular botnet are expected to exhibit similar activity patterns. The solution is based on an independent Poisson process and detection is based on the negative selection method. The experimental evaluation of this approach is incomplete.

*Sol*5. **MALICIOUS**: Bazm [48] proposed a NB-based botcloud detection approach which uses a combination of two machine learning methods, entropy and clustering methods. The traffic of all the VMs are captured and then classified. Once a dataset is obtained, the entropy value of the selected parameters for a VM is calculated. Then, the clustering method is applied on the entropy values of all the monitored VMs to detect any outliers, i.e. malicious VMs. The results proved the efficiency of the solution with hundreds of VMs. However, a centralized approach to collect the initial data might introduce additional overheads with scalability issues.

*Sol*6. **PEBBLETRACE**: Wenjie [78] presented a PebbleTrace scheme and proposed an approach to traceback to the botmaster. This study is focused on the attacks where a botmaster attempts to steal sensitive data from victim machines and the tracing pebbles can be spread along the stolen data back to the botmaster. The process first identifies cryptographic keys of the botnet communications for configuring botnet operations and then traces back to the botmaster. The approach can be applied beyond multiple clouds without deployment of monitors, router updates, or ISP support. The approach is implemented, and it shows promising results.

2.6.2 Host-based Botcloud Detection Solutions

The following botcloud detection solutions are based on Host-based intrusion detection system (HB-IDS).

*Sol*7. **PCA**: Badis [79] (PCA) proposed a botcloud detection method based on HB-IDS which relies on the previous activity of a VM and detects any new changes in its behaviour. It uses a statistical technique, the Principal Component Analysis (PCA) method [80], to detect the anomalies which are assumed to be signs of a botcloud. To do so, all the VMs are monitored and data such as CPU and memory usage, packets sent and received are collected at a central analyser. The collected data is analysed using the PCA method to detect any outlier, i.e. data which is different from the most of the data or doesn't assume the same statistical distribution model. The solution has shown a high detection accuracy in scenarios where the detection system could correctly record the usage of resources of a VM before it gets infected. However, the solution may not be able to detect botVMs which mount attacks by using legitimate resources. In addition, the use of a central analyser might become a single-point-of-failure.

*Sol*8. **EYECLOUD**: Memarian [81] proposed a botVM detection system based on HB-IDS that detects distributed botVMs using a combination of machine learning techniques, i.e. Clustering and Virtual Memory Introspection (VMI). VMI enables a VM (called Forensic VM) to inspect the memory space of another VM. The detection system uses one FVM per VM to inspect their memory space. The FVMs collect and store the system level metric values of the VMs at a central coordinator. Based on these metric values, the VMs are grouped; infected VMs are separated from benign VMs. The detection system has high detection accuracy as each VM is continuously monitored by an FVM and any change in VM metric value is reported immediately. However, this might not be cost-efficient when the number of VMs is large, as every VM is analysed by a dedicated Forensic VM (FVM). In addition, a central server might get overburdened with an increasing number of VMs and become a single-point-of-failure.

*Sol*9. **COLLABORATIVE**: Hammi [82] proposed a botcloud detection approach based on HB-IDS which uses monitoring probes (placed at the hypervisor level) to observe the system activity of all the VMs. The approach uses the PCA algorithm to calculate the factorial space describing a VM activity. The factorial space of a VM is depicted using a tree structure. Then the solution uses a signature-based approach, where the factorial space of a VM is compared with the factorial space representing DDoS flooding attack activity. Any dissimilarity is reported as a DDoS attack. The detection system uses a tree-based hierarchical architecture which allows it to apply the detection algorithm on a small set of VMs, thus minimizing the execution time and execution complexity of the detection algorithm. However, the solution may not be effective in detecting unknown attacks, due to the use of signature-based approach.

*Sol*10. **LAR_SCL**: Cogranne [83] proposed a HB-IDS solution which aims to detect botVMs with low malicious activities, e.g. botVMs which mount attacks by using legitimate resources. To do so, monitoring probes are used to capture the usage of resources consumed by each VM. These monitoring probes are located on the physical host and they use a virtualisation layer to capture metrics related to the usage of resources consumed by the VMs. Firstly, the detection algorithm estimates the cloud workload of the virtual hosts to enable the detection algorithm to get the reference of what a normal behaviour is, while the second stage compares outliers to the signature of well-known resource consumption attack patterns. The detection system has shown high efficiency and effectiveness with large number of VMs. However, the detection algorithm may not be effective against attacks other than DDoS attack and the algorithm may not be able to detect botVMs which are infected before the detection system was installed. In addition, this approach can detect botcloud in scenarios where an attacker creates VMs dedicated to the attack activity. That means all the activities of these botVMs are malicious. However, there can be scenarios where a botVM can generate a legitimate load due to the legitimate user'*s* activity as well as the malicious load due to the attacker'*s* activity. The combination of both activities modifies the behaviour of the VM, making this approach not able to detect such form of abuse.

Table 2.1: A Summary of the State-of-the-art botcloud detection solutions

Solutions	**Detection Method**	**Working Characteristics**	**Strengths**	**Limitations**
Sol1 (2011)	Anomaly-based (Machine Learning): Dependency graph & MapReduce	It leverages a host dependency model and an adapted PageRank algorithm.	Efficient (computation overhead) for large number of network connections, i.e. more than 1 million connections.	The assumption that densely interconnected VMs are botVMs can result in a high false negative rate. This is because other nodes such as p2p nodes are also highly interconnected.
Sol2 (2015)	Anomaly-based (Statistical)	Defence method leveraged by CSPs to trace back malware based on attack flows address information.	Can effectively identify hidden malware.	Can trace back the malware when an attack has already taken place and it also suffers in scalability due to the size of data collected.
Sol3 (2013)	Anomaly-based (Machine Learning): Pattern matching	It aims to examine the communication patterns of VMs to detect any anomalous behaviour.	—	This might not be cost-efficient with scalability. In addition, the experimental evaluation of the methods is incomplete.
Sol4 (2014)	Anomaly-based (Machine Learning): Artificial Immune System (AIS)	It aims to detect malicious activity patterns of botnet using AIS probability.	—	—
Sol5 (2015)	Anomaly-based: Entropy and clustering	It clusters VMs into classes that share similar behaviour based on their network parameters.	High detection accuracy.	Less efficient with large number of VMs; the central storage unit may become a bottleneck or a single-point-of-failure.
Sol6 (2012)	Anomaly-based	It identifies the cryptographic keys of botnet communication to configure botnet operation and then trace back to the botmaster.	No requirements of monitors and router updates.	can detect attacks at later stages as they tend to pebble trace once attacks are done.
Sol7 (2014)	Anomaly-based (Statistical): Principal Component Analysis (PCA)	It uses PCA-based approach to detect anomalies that can be signs of botcloud's behaviour.	Effective in cases where the VM is infected after the detection system has recorded the normal activity of the VM.	Less efficient with increase in the number of VMs. Not effective in cases where a VM was infected before the detection system is introduced.
Sol8 (2015)	Anomaly-based (Machine Learning): Virtual Machine Introspection (VMI) & Clustering method	It uses VMT technique to reduce target group under inspection and then cluster VMs based on inconsistencies detected by VMI.	High detection accuracy.	Less efficient with large number of VMs; dedicated FVM for each VM.
Sol9 (2015)	Signature & Anomaly-based (Statistical): PCA	It uses monitoring probes located at hypervisor level that monitor system activities of VMs.	Fast reaction time and low complexity.	Cannot detect unknown bots as it relies on signature based approach; less effective for detecting low footprint attacks.
Sol10 (2017)	Signature & Anomaly-based (Statistical): PCA	It uses a decentralized PCA based approach for estimating current workload of virtual hosts to highlight outliers by rejecting legitimate activity and then comparing the footprint of residual activity to known signatures.	Efficient with large number of VMs and less false-positive rate.	Cannot detect newly introduced botVMs and botVMs which are already existing in the system before the detection solution was implemented.

From the above discussions Table 2.1, we can summarise the strengths and limitations of the existing solutions. The strengths are:

*Str*1. The solution [73] uses MapReduce programming model to deal with huge volumes of data in a cloud network. MapReduce enables the monitored network data to be distributed into multiple units which are processed simultaneously and the results from the units are aggregated to get the final result. This parallel computing paradigm is well suited for a cloud network.

*Str*2. The solution [77] uses Artificial Immune System (AIS) method and shows that it could be one of the best solutions for detecting botclouds since the botVMs follow a malicious pattern.

*Str*3. The solution [81] uses Virtual Memory Introspection (VMI) method in which the memory space of a VM is inspected by another VM. This ensures a high detection rate.

*Str*4. Existing solutions [48][79][83] use source-based detection system and Principal Component Analysis (PCA) method for botcloud detection. Source-based detection enables them to study the behaviour of source of attack, i.e. VMs. PCA is a descriptive statistical method which aims at easing the analysis of high-dimensional vectors of input data, thus can be helpful for large volume of cloud data.

The limitations are:

*Lim*1. **Limited effectiveness**

(a) Existing anomaly-based solutions either leverage a network-based approach by analysing and making detection decision based on the usage of network-level resources such as bandwidth [1, 4, 5, 9] or a host-based approach by analysing and making detection decision based on the usage of host-level resources such as CPU, memory, storage resource [3, 6-8, 10]. These approaches have following limitations.

i. A network-based solution cannot detect a botcloud which only exploits host-level resources and, similarly, a host-based solution cannot detect a botcloud which only exploits network resources.

To detect both categories of botcloud attacks, solutions with both approaches need to be deployed.

ii. Solutions with these approaches may not be able to detect botclouds that consume resources within the (legitimate resource workload) limit allocated by the CSP.

iii. Solutions are designed to detect a particular attack and they cannot detect other forms of attacks.

(b) Existing solutions cannot detect botVMs that are already installed in the system but have not yet exhibited any malicious behaviour and/or botVMs that were already infected prior to the deployment of such solutions.

*Lim*2. **Post-attack detection**: The existing solutions [1, 3-10] largely focus on detecting a botcloud after it has launched an attack. Little effort has been made on examining how to detect and stop botcloud attacks as early as possible in the lifecycle of a botVM attack.

*Lim*3. **Use dedicated components for botcloud detection**: Detection of botcloud consists of two major functions: 1) VM monitoring & storage of monitored data and 2) analysis of the monitored data. In existing solutions monitored data is stored in a centralized way which may create a bottleneck in the network.

*Lim*4. **Conflict between detection accuracy and overhead imposed**: Existing network/host-based solutions require the collection of network/host related data from all the VMs in the underlying environment for a fixed period. If this period is set to a smaller value so that less data is collected thus a lower level of overhead, then the detection result may not be accurate. If this period is set to a larger value to improve the accuracy, then that will increase the storage overhead.

2.7 What is Missing?

This section outlines the findings from our survey of the existing botcloud detection solutions. These findings are depicted in Table 2.2, which summarises the botcloud detection solution requirements with the solutions surveyed above.

Table 2.2: State-of-the-art of the existing botcloud detection solutions against desired requirements

Solutions	**FUL**	**EFS**	**EFY**	**SCY**
Sol1	✗	✓*	✗	✓*
Sol2	✗	✓*	✓*	✓*
Sol3	✓	✓	✗	✗
Sol4	✗	✗	✗	✗
Sol5	✗	✗	✗	✗
Sol6	✗	✓*	✗	✗
Sol7	✗	✓	✗	✗
Sol8	✗	✓*	✗	✓*
Sol9	✗	✓	✗	✗
Sol10	✗	✓*	✓*	✓

✓: The solution has considered the corresponding requirement.
✓*: The solution has considered the corresponding requirement, however there is a scope of improvement.
✗: The solution has not considered the corresponding requirement.

From the Table 2.2, it is evident that none of the existing solutions has satisfied all the identified requirements. The observations are:

- Most of the existing solutions are designed for detecting some specific malicious activities, like DDoS. They are largely based on attack-evidence, i.e. detection decisions are made based on the data generated as the result of attacks. These solutions are not effective when being used to detect botVMs in the early stages of the attacks. In addition, owing to their dependencies on the frequency of monitored data collection, the effectiveness of the existing solutions also depends on the attack intensity. They are more effective when the attacks are high-footprint attacks.
- In addition, in terms of satisfying the efficiency requirements, thus making a botcloud detection solution more scalable, there is much room for improvement.
  - For example, in the design of solution [1], the scalability requirement has been considered and addressed by using a MapReduce model, a large-scale parallel and distributed computing paradigm. However, the solution only addresses the scalability problems as caused by processing and storage overheads. It is still subject to scalability issue as caused by potentially high communication overheads incurred in the use of a central storage server and the collection of traffic logs by the server.

  - Solutions [1-7] require the monitored data to be sent to a central analyser. This makes the resulting communication overheads increase proportional to the number of VMs being monitored' particularly, the monitored data need to be sent regularly at certain intervals. The value of this interval directly impacts the effectiveness of the botcloud detection. In other words, the smaller the interval, the higher the effectiveness in detecting botcloud, but the higher the level of the communication overheads generated in the network. This indicates that there is a trade-off between effectiveness and efficiency/scalability.

The following section highlights our vision in designing a botcloud detection solution considering the requirements specified in Section 2.5.

2.8 The Way Forward

We aim to design a solution that could overcome the limitations in the existing solutions and satisfy the requirements highlighted earlier in this chapter. More specifically, the designed solution should be able to detect a botcloud irrespective of the types of attacks it executes and as early as possible.
We would aim to design a solution that integrates the strengths of HB-IDS and NB-IDS with an aim of effectively detecting botVMs. To do so, we will investigate the idea of two-stage detection combined with a VM suspension measure upon the completion of the first stage (monitoring stage) and in-depth analysis of any VMs that have been identified as suspicious upon the completion of the second stage (analysis). This two-stage approach may allow us to detect any malicious VMs as early as possible. In addition, the solution should be efficient in that it imposes as little overhead as possible and the risks of creating a performance bottleneck in the system is as low as possible. As the process of botcloud detection consists of two stages, monitoring of VMs and analysis of monitored data, we will use a distributed approach with the idea of localised monitoring and maximising local processing of monitored data. It is anticipated that these measures may reduce traffic poured into the network, making the solution more scalable.

2.9 Chapter Summary

This chapter has given an overview of botnet detection techniques and described some related work. It then outlined a critical analysis of the existing botcloud detection methods, identifying their strengths and limitations. Based on this critical analysis, it has presented some new ideas to detect botcloud in an effective and efficient manner. The next chapter presents the building blocks used to implement the novel ideas and methodology used to evaluate the implementation.

Chapter 3

bVMD Architecture and Evaluation Methodology

3.1 Chapter Introduction

This chapter describes the architecture of a novel botVM detection framework called BotVM Detector (bVMD). The bVMD framework is designed to detect VMs infected by a botcloud in a multi-host environment in an effective, efficient and scalable manner. The main idea used in the design of bVMD framework is inspired by a local vigilance scheme known as 'Neighbourhood Watch'. This scheme actively seeks support of local people to watch each other's activity and report anything suspicious. This scheme is assumed to be cost effective because of the support of existing local people and minimizing the requirement of recruiting more police officers and stepping up patrols [84]. This same idea is used in bVMD framework whereby a local monitoring component (i.e. local people) is placed within each VM instead of having a centralized component (i.e. police), thus minimizing the chances of creating a bottleneck or a single point of failure. However this idea might introduce additional overheads due to an increase in the number of monitoring components and to minimize these additional overheads, bVMD design has made use of the following five measures: (i) hybrid approach of botcloud detection based on a network parameter and system parameters, (ii) use of in-VM monitoring component, (iii) local processing of the monitored data at VM level, (iv) collective and accountable decision made by each component at VM level, and (v) an adjustable dispatcher for system parameter analysis components. In addition, the bVMD framework introduces four architectural components: (a) VMWatcher, (b) Suspected-VM Detector (S-VMD), (c) Bot-VM Detector (Bot-VMD), and (d) Forensic VM (FVM). This chapter also describes the different parameter value settings to investigate their impact on the performance of the bVMD framework. The design and evaluation of the bVMD framework is one of our novel contributions in this thesis.

The structure of the remaining part of this chapter is as follows. Section 3.2 gives an overview of bVMD Architecture and Section 3.3 specifies the measures taken in the design of bVMD framework. Section 3.4 gives design preliminaries, the assumptions, the definitions used in the design and the design requirements for an effective, efficient and scalable botcloud detection solution. Section 3.5 describes the architecture of the framework and its components. Section 3.6 outlines the bVMD evaluation methodology and Section 3.7 presents the simulation configuration used. Section 3.8 discusses the simulation scenarios and settings, and finally Section 3.9 summarises the chapter.

3.2 bVMD Architecture Overview

This section presents an overview of bVMD architecture on a single cloud, multi-host environment. The bVMD architecture is based on an Autonomic Computing [85][86] Approach with four architectural components as shown in Figure 3.1. These components are: 1) VMWatcher, 2) Suspected VM Detector (S-VMD), 3) BotVM Detector (Bot-VMD) and 4) Forensic VM (FVM) (Δ).

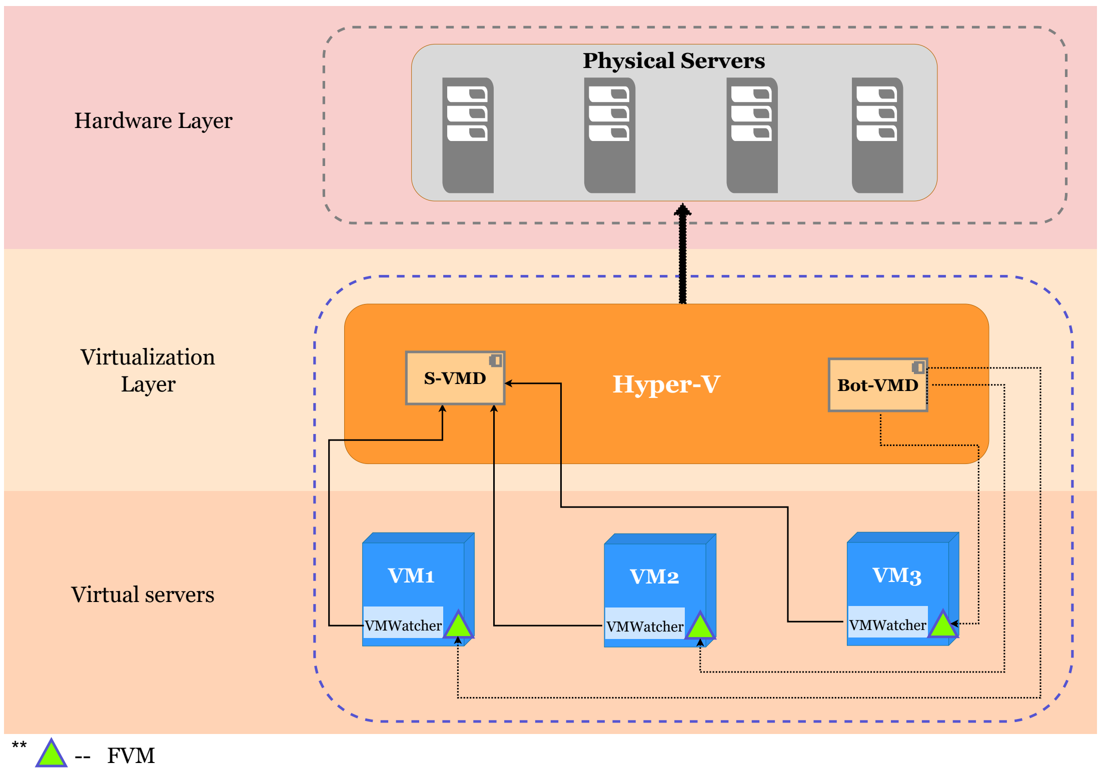


Figure 3.1: An Overview of bVMD Architecture

These components along with their functions are highlighted below and described in more depth in the upcoming chapters.

VMWatcher: This is a network monitoring component placed at the VM level. It is located in every VM and is responsible to acquire network communication data at the residing VM. The acquired data is analysed locally and is sent to its local S-VMD for further analysis.

S-VMD: This is a network analysis component placed in the hypervisor and is responsible for acquiring the individually analysed data from all the local VMs (i.e. VMs residing on the same physical host) and collates it, resulting in a set of suspected VMs (S-VMs). It then notifies the Bot-VMD of each S-VM.

Bot-VMD: This is a system analysis component placed in the hypervisor and is responsible to analyse system parameters of S-VMs. To perform the analysis, it decides the number and type of FVMs required and instructs them accordingly. In addition, it receives results from FVMs and provides the final list of botVMs.

FVM: This is a system analysis component placed at the VM level and is responsible to check the system parameters of S-VM as instructed by Bot-VMD. It reports the results back to Bot-VMD.

3.3 Design Measures

In this section, we discuss the measures used in the design of bVMD framework.

3.3.1 Hybrid approach of botcloud detection based on analysis of network parameter & system parameter

Based on our literature study, the existing botcloud detection solutions proposed are mainly based on network-based IDS or host-based IDS. Another way to design a solution is to use a hybrid approach by integrating both the methods. This means analysing each VM twice, i.e. using both these methods. On one hand this dual analysis of a VM might minimize false detections, however on the other hand it might introduce additional overheads. bVMD framework uses this hybrid approach with an aim of optimizing trade-off between accuracy and overheads. To do so, it uses a two-stage analysis process with stage-1 implemented to reduce the target group under inspection and stage-2 implemented to perform an in-depth analysis of the identified target group. Stage-1 uses a network-based anomaly detection method and stage-2 uses a host-based anomaly detection method.

3.3.2 Use of in-VM monitoring component

In stage-1 of bVMD framework, the VM network traffic volume data is monitored by a component which can either reside at hypervisor level or VM level. Residing at the hypervisor level might require only one centralized component which can monitor the network communication data of all the local VMs (i.e. VMs residing on the same physical host), thus having an economically effective solution. On the other hand, this centralised component can create a bottleneck and increase the chances of a single point of failure. In addition, this method might require all the network communication to pass through this monitoring VM (further discussed in Chapter 4), which is not feasible with an increase in number of VMs and might introduce high overheads with scalability.

Residing at the VM level, the network communication data can be monitored by (1) a centralised approach with dedicated additional VM – a single VM monitoring all the VMs or (2) a distributed approach with dedicated additional VMs – a VM monitoring each VM. The centralised approach requires only one monitoring VM, thus minimizes the overheads. However, it may have similar issues as discussed above (at the hypervisor level). These issues are overcome by using a distributed approach which requires dedicated set of VMs monitoring the VMs in parallel, thus leading to faster detection. However dedicated components might involve high maintenance cost and they might increase the attack surface of the framework thus making the solution less secure.

To minimise these overheads, bVMD introduces a distributed approach with no additional VMs, rather it uses a monitoring component which resides within each VM. This component is responsible to monitor network communication data received at its residing VM, thus reducing the overhead cost of placing and maintaining dedicated VMs & improving scalability.

3.3.3 Local processing of the monitored data at VM level

The VM traffic volume network data monitored by each component has to be analysed for further actions. As discussed, this data is monitored at VM level, and further actions are taken at hypervisor level, therefore the monitored data can either be analysed at VM or at hypervisor. In the case of analysing at the hypervisor, raw, monitored data from all VMs should be directly communicated to their hypervisor, thus introducing a high communication overhead due to the volume of raw data. In case of processing at VM locally and communicating the analysis results to the hypervisor might minimize the communication overhead as compared to the above method. Moreover, the result could be processed faster at VM level as the monitored data is analysed in parallel at all VMs, rather than a centralized component. Furthermore, in case of processing the data locally at VM level, each VM can have a local protection scheme in which it can take any immediate actions based on its locally analysed data. This protection scheme might be helpful in case of a compromised hypervisor.

3.3.4 Collective and accountable decision making by each VM

As discussed above, a monitoring and analysis (M&A) component resides in each VM; however, these VMs are vulnerable to attacks. A malicious VM might influence the M&A component residing within and forge or alter their true M&A results. For example, it might mislead by (i) showing an uninfected VM as malicious, or (ii) a malicious VM as an uninfected VM. These forgeries are addressed by taking following two measures. Firstly, the trustworthiness of each VM is calculated based on the collective feedback from its local VMs and this is fed into the decision-making process. Secondly, the final decision result is generated in a collective manner, i.e. taking into account the M&A results from all the available VMs. These measures might increase the accuracy of the system and minimize the effect of an attack done by a malicious VM. However, these might introduce additional computational and storage overheads. To study the performance of a malicious VMWatcher and effect of these measures, a security risk assessment is carried out in Section 4.7.

3.3.5 An adjustable dispatcher for system parameter analysis components

Stage-2 of bVMD Framework is inspired by the high detection rate in [81] discussed with the strengths of existing solutions (Section 2.6). In [81] the VM parameters are collected by a component, known as the Forensic Virtual Machine (FVM) and the number of FVMs depend on the number of VMs to be analysed. However, the number of FVMs could also be dependent on the number of parameters to be checked [87]. This means that it can either be (1) equal to the number of VMs, i.e. a FVM analysing all the system parameters of a VM, or (2) equal to the number of system parameters to be analysed, i.e. a FVM analysing a system parameter of all the VMs. An increase in any of these can increase the number of FVMs, which might introduce following issues: (1) an increase in computational overhead involved in maintaining the FVM, and (2) an increase in the attack surface of the solution. To address these issues a hybrid approach of determining the number of components is introduced in bVMD framework, wherein the number of components is equal to the lesser of the two, i.e. the number of VMs and the number of system parameters. This might minimise the number of FVMs required with an increase in number of VMs or system parameters.

3.4 Design Preliminaries

This section details the assumption and definitions used in the design of bVMD framework and its design requirements.

3.4.1 Assumption

To scope our design, the following assumption is used.

A1 The hypervisor is assumed to be secure, as assumed in numerous studies [88] [89] [90][91].

3.4.2 Definitions

The definitions and performance metrics used in bVMD framework are as follows.

1. Definitions:
   - *False Positive (FP)*: A benign VM wrongly detected as a botVM.
   - *True Positive (TP)*: A botVM which is correctly detected.
   - *False Negative (FN)*: A botVM which is wrongly classified as uninfected VM.
   - *True Negative (TN)*: A benign VM which is correctly detected.
   - *False Positive Rate (FPR)* [92]: Proportion of benign VMs wrongly detected as a botVM, i.e.

$$FPR = \frac{FP}{(FP + TN)} \tag{3.1}$$

   - *True Positive Rate (TPR) or Sensitivity* [93]: Proportion of botVMs correctly detected, i.e.

$$TPR = \frac{TP}{(TP + FN)} \tag{3.2}$$

   - *False Negative Rate (FNR) or Miss Rate* [94]: Proportion of botVMs wrongly classified as uninfected VM, i.e.

$$FNR = 1 - TPR \tag{3.3}$$

   - *True Negative Rate (TNR) or Specificity* [95]: Proportion of benign VMs correctly detected, i.e.

$$TNR = 1 - FPR \tag{3.4}$$

   - *False Positive Time (FPT)*: The instance of time when first benign VM was wrongly detected as a botVM.
   - *True Negative Time (TNT)*: The instance of time when all the benign VMs were correctly detected.

2. Performance metrics: To evaluate accuracy performance of a botcloud detection solution, three metrics are commonly used: (1) Receiver Operating Characteristic (ROC) Curve, (2) Accuracy and (3) Matthews' Correlation Coefficient (MCC). The definition of these metrics is given below.

   - *Accuracy*: The accuracy of a test is its ability to differentiate the infected and benign cases correctly. It is evaluated as the proportion of true positive and true negative cases among the total number of cases.

   $$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.5}$$

   - *Matthews' Correlation Coefficient (MCC)* [96] : It is correlation coefficient between groups of observed and predicted VMs, i.e.

   $$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{3.6}$$

   To evaluate the overhead performance of a botcloud detection solution, two metrics are used: (1) Communication overhead and (2) Storage overhead

   - *Communication Overhead (C-OH)*: The number of messages and the length of each message (in bits) exchanged between the architectural components.
   - *Storage Overhead (S-OH)*: The number of values recorded by each architectural component and the length of these values (in bits).

3.4.3 Design Requirements

In addressing the design challenges for an effective, efficient and scalable botcloud detection solution (identified in Section 2.5), three set of requirements (functional, performance and scalability requirements) are specified.

Functional Requirements (FUL)

(FUL1) *Detect botcloud with known and unknown signatures*: A botcloud signature is a pattern or string that corresponds to an attack by a botcloud. Botclouds with known signatures are existing botclouds or botclouds that have been identified. Botclouds with unknown signature are new botclouds or botclouds that have not been identified. The solution should be able to detect both these classes of botclouds.

(FUL2) *Detect botclouds regardless of their Command and Control (C&C) message formats, structures and protocols*: Usually, botVMs of a botcloud are commanded and controlled by their botmaster or other botVMs and this is carried out with the use of communication messages. These C&C messages may be encrypted or unencrypted.

The underlying communication structure used may be centralised (communications among botVMs are done via their botmaster), Peer-2-Peer (P2P, botVMs can communicate directly) or a hybrid set of centralised and P2P structure. The protocols used for C&C may be different too, e.g. HTTP, IRC, etc. The solution should be able to accommodate all these different message formats, C&C structures and protocols.

Performance Requirements These are divided into two categories: effectiveness and efficiency.

Effectiveness Requirements (EFS)

(EFS1) *Minimize False Positive Rate (FPR)*: The solution should be able to detect botVMs with as low FPR as possible.

(EFS2) *Minimize False Negative Rate (FNR)*: The solution should be able to detect botVMs with as low FNR as possible.

(EFS3) *Maximize True Positive Rate (TPR)*: The solution should be able to detect benign VMs with as high TPR as possible

(EFS4) *Maximize True Negative Rate (TNR)*: The solution should be able to detect botVMs with as high TNR as possible.

Efficiency Requirements (EFY)

(EFY1) *Low communication overhead*: Communication overhead (C-OH) should be as low as possible.

(EFY2) *Low storage overhead*: Storage overhead (S-OH) should be as low as possible.

Scalability Requirement (SCY)

(SCY1) *bVMD solution should be scalable*: Scalability is a characteristic of a system that describes its capability to perform under an increase or expanded workload [97]. The solution should be able to have a high Matthews' correlation coefficient (MCC) value when the number of VMs increases.

3.5 bVMD Architecture & its Components

This section describes bVMD architecture and its components. The architecture of bVMD framework is shown in Figure 3.2 and an overview of the interactions between the components is shown in Figure 3.3.

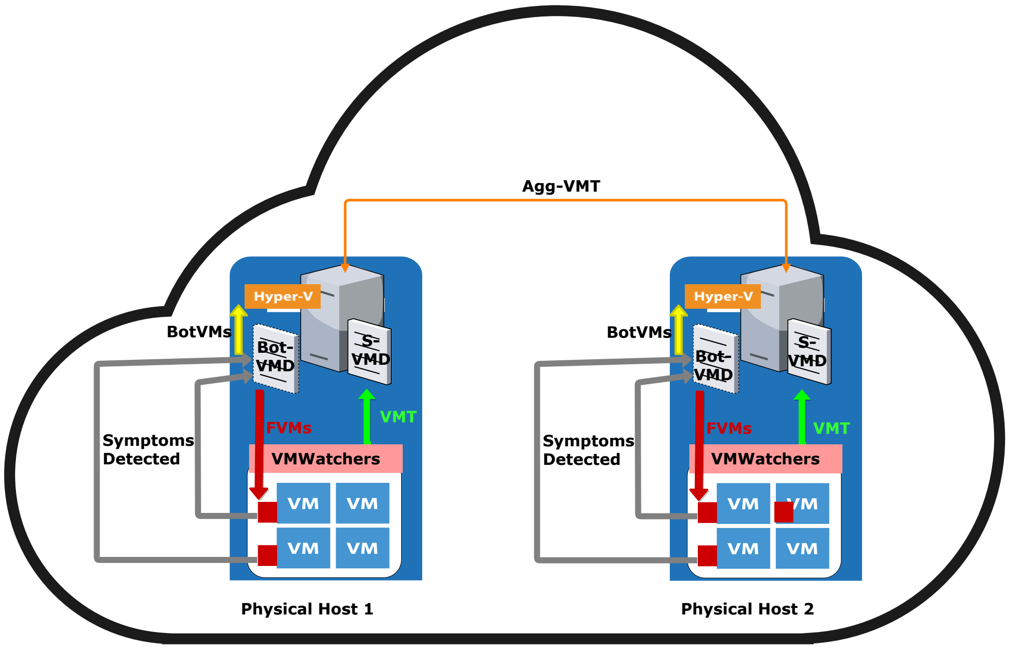


Figure 3.2: bVMD Architecture

The bVMD operations can broadly be captured in two stages: Stage-1 (Monitoring and analysis of VM network communication data) and Stage-2 (System parameter analysis). In Stage-1 the most suspected malicious VMs are identified using their network communication data, using VMWatcher and S-VMD components. In Stage-2, the identified suspected VMs are re-investigated to ensure that they are malicious VMs, using bot-VMD and FVM components.

3.5.1 VMWatcher

VMWatcher uses trust values to quantify the network communication packets received at their residing VM. These communication-based trust values are initialized and updated using the following methods: 1) VMT value Initialization and 2) VMT value Update. The Initialization method is used for initialization of a VM trust value whenever there is a first communication request from a VM. The Update method is used for updating the VMT values with every subsequent communication. All these VMT values are stored in VMT trust table (VMT-T). This component along with its methods are described in more depth in Section 4.2.

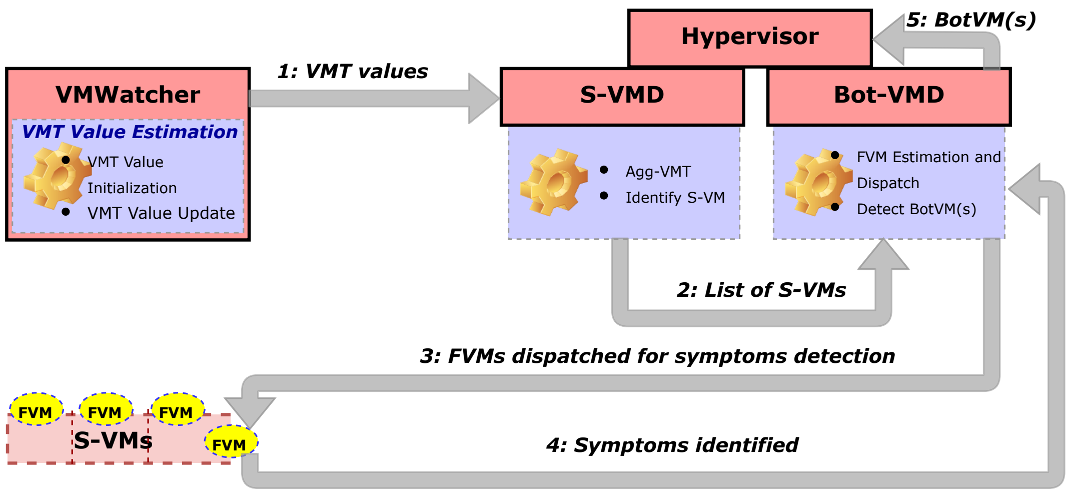


Figure 3.3: bVMD Components and their Interactions

3.5.2 S-VMD

S-VMD aggregates the VMT values received from its local VMWatchers. This is done by using the following methods: 1) Agg-VMT and 2) Identify S-VM. The Aggregated-VMT (Agg-VMT) method is used to aggregate the VMT values received from all the local VMWatchers and calculate an Agg-VMT value for each VMWatcher. Based on the Agg-VMT value of each VMWatcher, the S-VMD method identifies a set of S-VMs and notifies their respective bot-VMDs. This component along with its methods are described in more depth in Section 4.3

3.5.3 Bot-VMD

Bot-VMD receives notification from S-VMD and perform the following two methods: 1) FVM Estimation and Dispatch and 2) Detect botVM(s). The FVM Estimation and Dispatch method is used to estimate the number and types of FVMs required for analysis and then dispatch them to identified S-VM. The Detect botVM method is used to receive the recorded results from FVMs, analyse them and identify botVMs. It then notifies the hypervisor about the result. This component along with its methods are described in more depth in Section 5.5.

As discussed above, the components S-VMD and bot-VMD are positioned in the hypervisor. This idea of placing the components in the hypervisor is inspired by hypervisor-based IDS [98] [99]. Hypervisor-based IDS use the hypervisor layer to secure the monitoring system. In theory, since hypervisors operate at a lower level than the monitored system, they too are isolated and become more secure [100]. This means that although placing the components in the hypervisor might introduce additional overheads on the hypervisor, but at the same time they can be more secure. Also, it can be seen from the literature that several solutions are [101] [102] based on this approach.

3.5.4 FVM

The FVMs check the system parameter values as per the instructions of Bot-VMD. They are only active for the time of checking system parameters and reporting back. After completing their work, they are destroyed to minimize the attack surface.

3.6 Evaluation Methodology

This section discusses the possible evaluation methods that may be used to evaluate bVMD Framework, and this discussion serves as our justification for selection of the methodology used in the study presented in this thesis. According to literature [103], there are three evaluation methodologies, real system experiments, mathematical modelling and simulation.

3.6.1 Real System Experiments

The real system experiment evaluation method evaluates the performance of a solution on a large-scale physical network. Here the network refers to a hardware infrastructure system and a software framework managing it. This is done using testbeds built on a physical network. Using these testbeds to evaluate a solution is advantageous as the results are based on realistic conditions. These can be helpful for understanding the performance of a solution before deploying it on a large-scale network [104].

Based on the above discussed reasons, we have chosen this method as the evaluation methodology for the Stage-2 of bVMD Framework reported in this thesis.

3.6.2 Mathematical Modelling

The mathematical modelling method builds an abstract model that uses mathematical language to evaluate the behaviour and performance of a framework. Although this method is more cost effective than the method discussed above, it might not generate valid results in bVMD framework. This is because bVMD framework is designed for a complex cloud network operating in a dynamic environment. To model complex operations, many assumptions have to be made to keep the analysis traceable, and simplifying these assumptions may limit the usefulness or validity of the finding.

3.6.3 Simulation

A network simulator method uses a software program to model the behaviour and performance of a solution. For a cloud network, this method is getting increasingly popular due to three main benefits [105][106]. Firstly, the simulation method is cheaper as compared to the real environment. Secondly, it is scalable, i.e. it can be used to simulate a small or large network. Thirdly, it is less time consuming and repeatable.

Based on the above discussed reasons, we have chosen this method as the evaluation methodology for the Stage-1 of bVMD Framework reported in this thesis.

To determine a simulator for evaluation of bVMD Framework, we identified the simulators used by the related work. Authors in [79] and [82] used R tool [107] to build their own simulator whereas authors in [48] used CloudSim [108] simulator. In addition, we identified that NS-2 [109] and Omnet++ [110] was used couple of solutions as network simulations.

Cloudsim is a generalised and extensible Java-based simulation tool kit [111]. It is used by researchers to model and simulate cloud infrastructure and analyse the performance of the application service in a controlled environment [112]. It can view the availability, power consumption, and a network traffic of services on a cloud environment; however it cannot support the Quality of Service [113]. NS-2 is an object-oriented, discrete event simulator for networking research. It is written in C++ and the command and configuration interface is built using Object Tool Command Language (OTcl). The advantage of this programming approach is that it allows for fast regeneration of large scenarios. However, one of its limitation is that modifying and extending the simulator requires programming and debugging in both languages simultaneously [114]. In addition, it has another limitation of poor graphical support, i.e. no Graphical User Interface (GUI). Omnet++ is an extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators. Although it is not a network simulator itself, it has gained widespread popularity as a network simulation platform for building up a large user community in the scientific as well as industrial settings [115]. Unlike NS-2, Omnet++ provides a powerful GUI making the tracing and debugging much easier. However, there is a problem of compatibility issue, i.e. combining of models developed separately is difficult to combine and it may result in bugs [116].

Omnet++ has been chosen for simulation evaluation of stage-1 of bVMD Framework. As discussed, it has been used to write many frameworks in diverse areas: peer-to-peer networks, mesh network, wireless sensor network, cloud computing and more. Omnet++ uses INET Framework [117] as one of the major frameworks for modelling communication networks. The version of INET 4.2.0 framework is available at [118]. It is compatible for Omnet 5.5.1 or later. Omnet++ 5.6 [119] with INET 4.2.0 is used as an evaluation tool for the work reported in this thesis. In the INET framework, an INetworkNode is used to build each node (i.e. each VMWatcher) and standardhost is used to build the S-VMD component. This INetworknode maintains regular communication as per the scenario with a rate of 60 packets/minute and perform an attack with a rate of 120 packets/minute.

3.7 Simulation Configuration

This section explains network configuration, network modelling and performance metrics used for the evaluation of bVMD framework. In addition, this section explains how some of the parameter values were chosen during our simulation study.

3.7.1 Nodes

The VMWatchers are also known as nodes. These are divided into three categories: 1) target nodes (targetVM), 2) bot nodes (botVM), and 3) standard nodes (standardVM). A targetVM is a node which is attacked by a botcloud and botVM is a node which is a member of a botcloud. StandardVM is a node which is neither a target nor an attacker. All the nodes have similar rate of communication.

3.7.2 Peer-to-peer Model

The peer-to-peer model defines the number of peer-VMs of a VMWatcher. In other words, it defines the number of VMWatchers communicating with a VMWatcher. This model is designed to study the effect of change in the number of peer-VMs on the performance of bVMD framework. This is done by evaluating the performance of the framework with minimum, average and maximum number of peerVMs. Here, 'minimum' means 1% of the total number of VMWatchers in the system, with an exception of at-least one peer-VM. 'Average' and 'maximum' means 50% and 100% of the total number of VMWatchers. For consistency, the number of peer-VMs is assumed to be same for all VMWatchers in bVMD evaluation.

3.7.3 Performance Metric

Six performance metrics are used in our investigation of bVMD framework. They are FPR, FNR, TPR, TNR, FPT and TNT. These were discussed in detail in Section 3.4.

3.8 Simulation Scenarios and Settings

The simulation is based on an empirical model [120] wherein the number of experimental variable, such as the number of each type of VM, the time number of peerVMs, and, in particular, to find experimentally the optimal combination of conditions that provides the optimal results are identified. To the best of our knowledge, the experimental variables and scenarios discussed below are unknown. Also, as stated in [121], empirical models might be very useful in such circumstances where the mathematical relationship is unknown.

The simulation experiment scenarios are designed with an objective of analysing the performance of bVMD framework during the different stages of a botcloud. The botcloud stages and the ways of incorporating them into the simulation model are as follows.

Stage 1 (Initial stage) The botcloud has infected one VMWatcher.

Stage 2 (Intermediate stage) The botcloud has infected a larger group of VMWatchers as compared to initial stage, however there are still some uninfected VMs.

Stage 3 (Final stage) The botcloud has infected all the VMWatchers, except one.

To analyse the above cases, the following factors are considered:

F1 Number of botVMs (numBot): This describes the number of VMWatchers which are infected by botcloud.

F2 Number of targetVMs (numTarget): This describes the number of VMWatchers which are being attacked by botcloud.

The performance of bVMD framework is evaluated with minimum, average and maximum range of these factors. Here minimum refers to 1%, average refers to 50% and maximum refers to 99% of the number of botVMs and targetVMs. In addition, we make sure that there is at-least one botVM and one targetVM in every scenario. Based on these factors, six scenarios are formed as described in Table 3.1.

To conclude, the performance of bVMD is analysed for the following.

1 The effect of a large group of botVMs attacking a small group of VMWatchers.

2 The effect of a small group of botVMs attacking a large group of VMWatchers.

3 The effect of a small group of botVMs attacking a small group of VMWatchers.

4 The effect of an attack where the number of botVMs and targetVMs are equally distributed.

Table 3.1: Scenarios based on the distribution of three types of nodes

Scenario	Distribution of nodes
Scenario_A	• 1% of the NumBot • 1% of the NumTarget • 98% of the StandardVM
Scenario_B	• 1% of the NumBot • 50% of the NumTarget • 49% of the StandardVM
Scenario_C	• 1% of the NumBot • 99% of the NumTarget • 0% of the StandardVM
Scenario_D	• 50% of the NumBot • 1% of the NumTarget • 49% of the StandardVM
Scenario_E	• 50% of the NumBot • 50% of the NumTarget • 0% of the StandardVM
Scenario_F	• 99% of the NumBot • 1% of the NumTarget • 0% of the StandardVM

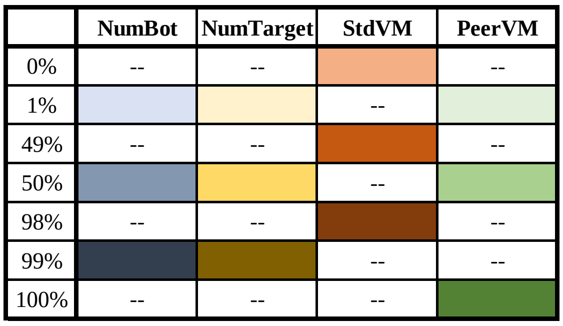
3.8.1 Settings

The peer-to-peer model described in Section 3.7 is also taken into account in the simulation experiment evaluation scenarios. This is to analyse the effect of change in performance of bVMD with the change in number of peerVMs. The different setting values considered for the number of peerVMs are summarized in Table 3.2

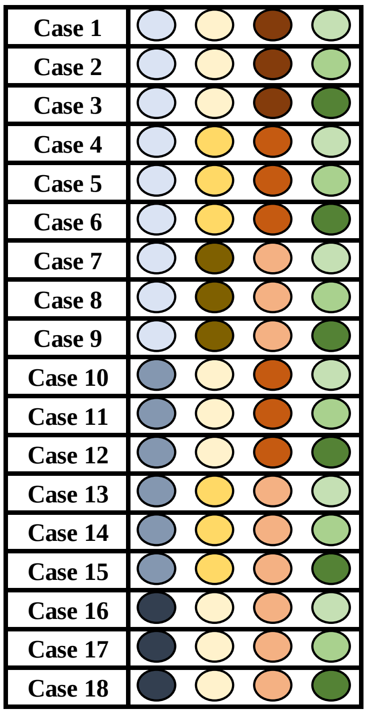
Table 3.2: Settings based on the percentage of peerVMs

Setting (SE)	Percentage of the number of peerVMs
SE1	1%
SE2	50%
SE3	100%

To conclude, Figure 3.4 depicts the 18 cases derived from the above mentioned scenarios and settings:



	NumBot	NumTarget	StdVM	PeerVM
0%	--	--	🟧	--
1%	🟦	🟨	--	🟩
49%	--	--	🟧	--
50%	🟦	🟨	--	🟩
98%	--	--	🟫	--
99%	⬛	🟫	--	--
100%	--	--	--	🟩



Case 1	🔵	🟡	🟤	🟢
Case 2	🔵	🟡	🟤	🟢
Case 3	🔵	🟡	🟤	🟢
Case 4	🔵	🟡	🟠	🟢
Case 5	🔵	🟡	🟠	🟢
Case 6	🔵	🟡	🟠	🟢
Case 7	🔵	🟤	🟠	🟢
Case 8	🔵	🟤	🟠	🟢
Case 9	🔵	🟤	🟠	🟢
Case 10	🔵	🟡	🟠	🟢
Case 11	🔵	🟡	🟠	🟢
Case 12	🔵	🟡	🟠	🟢
Case 13	🔵	🟡	🟠	🟢
Case 14	🔵	🟡	🟠	🟢
Case 15	🔵	🟡	🟠	🟢
Case 16	⚫	🟡	🟠	🟢
Case 17	⚫	🟡	🟠	🟢
Case 18	⚫	🟡	🟠	🟢

Figure 3.4: The settings (percentage of the number of peerVM) and the set of Cases derived from the scenarios (indicating the distributions of different types of nodes)

The first figure in Figure 3.4 shows the colour code assigned to each type of VM and the percentage of these VMs. Here the x-axis shows the types of VMs and the y-axis shows the percentage of these VMs. The right figure in Figure 3.4 shows the type and percentage of VMs in each case. For example, in Case 1, numBot is 1%, numtarget is 1%, stdVM is 100% and peerVM is 1%.

3.9 Chapter Summary

This chapter specifies a set of five measures and four set of requirements for the design of an effective, efficient and scalable botcloud detection solution. The measures are: (i) Hybrid approach of botcloud detection based on a network parameters and system parameters, (ii) Use of in-VM monitoring component, (iii) Local processing of the monitored data at VM level,

(iv) Collective and accountable decision made by each component at VM level, and (v) An adjustable dispatcher for system parameter analysis components. The requirements are FUL1) Detect botVMs with known and unknown signatures, FUL2) Detect botVMs regardless of their Command and Control (C&C) message formats, structures and protocols; EFS1) Minimise false positive rate, EFS2) Minimise false negative rate, EFS3) Maximise true positive rate, EFS4) Maximise true negative rate; EFY1) Low communication overhead, EFY3) Low storage overhead; SCY1) Solution should be scalable. To realise these measures and support the identified requirements, we present an overview of bVMD framework design. The bVMD framework consists of a two-stage novel architecture with the first stage centred on a novel network-based anomaly detection method and the second stage based on an improved host-based anomaly detection method. This two-stage architecture might improve the detection accuracy by reducing chances of false-positive detection and making the solution more effective. However, the two stages add a level of complexity due to dual-analysis. Efforts have been made to reduce the associated communicational overhead. The methods of evaluation are discussed further, and simulation is selected as the investigation methodology, and Omnet++ as a simulator. The configuration of Omnet++ is described in detail and a set of scenarios with related settings used in the simulation were discussed.

Chapter 4

bVMD Stage-1: VMWatcher & S-VMD

4.1 Chapter Introduction

This chapter presents the design and evaluation of VMWatcher and S-VMD; two components used in the Stage-1 of the bVMD architecture. As mentioned in Chapter 3, bVMD uses a peerVM mutual monitoring-based approach to identify suspected VMs in Stage-1 before carrying out detailed forensic analysis in Stage-2. This peerVM mutual monitoring approach, which is implemented by using VMWatcher and S-VMD, is aimed at reducing the number of VMs that should be further analysed in Stage-2. This approach brings a number of benefits. Firstly, it is designed to monitor the frequency of communications, which is independent of botVM signatures, communication structure and protocols. Therefore, the detection solution can be applied for detecting polymorphic malware botVMs and in a communication environment with varying communication topologies and protocols. Secondly, each VM is monitored by a number of peerVMs, so if a VM has more peerVMs, then a false reporting by a single peerVM will have less impact on the overall trust value, thus making the detection result more reliable. In other words, the more peerVMs a VM has, the harder it is for a single VM to abuse the rating operation, and thus a more accurate result is expected. However, this approach introduces additional overheads on the VMs for monitoring and analysing their peerVMs, efforts have been made to minimize these overheads.
A notable feature of a botcloud is that it tries to infect as many VMs as possible, thus attacking many peerVMs, so this peerVM mutual monitoring approach may produce some interesting results which will be evaluated. Also, the monitored data each VMWatcher transmits to the decision-making component, i.e. S-VMD, is a trust value, rather than the raw data, and the

former is much smaller in volume than the latter, so the communication overhead can be reduced. In addition, as the function of VMWatcher is embedded in each working VM, rather than as a dedicated component, the additional workload is spread across multiple components, and this can improve the scalability of the system. Lastly, S-VMD component is also embedded in each hypervisor which spreads the workload across multiple hypervisors, and this can also improve the scalability of the system.
The structure of the remaining part of this chapter is as follows. Section 4.2 presents the details of the VMWatcher component. Section 4.3 describes the S-VMD component. Section 4.4 presents the operations of bVMD stage-1 components and Section 4.5 discusses the experimental evaluation. The evaluation results and discussions are described in Section 4.6. Section 4.7 discusses the attack model, which identifies possible security threats and attacks that may be performed against Stage-1 of bVMD Framework and it also describes the results obtained in these attack scenarios. Section 4.8 presents the key findings of the experimental results, and finally Section 4.9 summarises the chapter.

4.2 VMWatcher

This section describes the novel VMWatcher component, the methods used to implement it and set of actions that it can take in response to different attack scenarios.

4.2.1 VM Communication Paths

VMWatcher component analyses the flow of network packets to identify any suspicious activity, so it should be placed in a location where it can access all the network packets. To determine such a location, we study the communication paths followed by the VM packets. There is a need for the VMs to communicate with each other to access set of services offered by each other and this is done using a communication channel. This channel can use two types of networks, i.e. physical network and/or virtual network.
A physical network refers to a network of physical hosts that communicate using physical devices like cables, modem, etc. As illustrated in Figure 4.1, a physical network consists of two main components: Physical Switches [122] and Physical Network Interface Cards [123].

- *Physical Switch (pSwitch)*: It links physical devices together by relaying network packets between them. Each pSwitch has multiple ports with each port connecting a physical device or another pSwitch.
- *Physical Network Interface Card (pNIC)*: It links a physical host to physical network (i.e. to a pSwitch port) and it has a MAC address to guide packets. In a cloud environment, a pNIC connects the hypervisor to the physical network. It serves as a hardware interface bridge between the physical and virtual network.

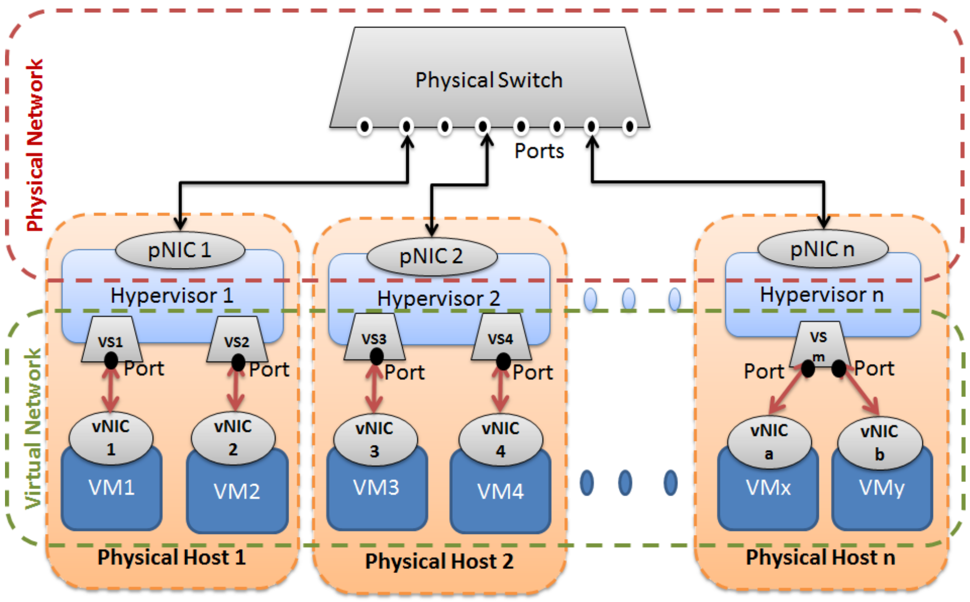


Figure 4.1: VM Communication Networks (Physical & Virtual) with their components

Virtual network refers to a network of VMs that communicate using virtual devices. As illustrated in Figure 4.1, the virtual network consists of two main components: Virtual Switch and Virtual Network Interface Cards.

- *Virtual Switch (vSwitch)*: A vSwitch connects a VM to a physical network. Each vSwitch has multiple ports, called vPorts, with each vPort connecting one or more VM(s). The vPorts can be configured to form port groups based on various setting like security, traffic shaping, etc.
- *Virtual Network Interface Card (vNIC)*: The vNIC connects a VM to a virtual network (i.e. to a vPort). Similar to pNIC, it also has a unique MAC address through which packets are guided.

During a communication between a pair of VMs, the communication path followed by the network packets depends on the location of the communicating VMs. These VMs can either reside on the same physical host, i.e. intra-host VM communication or on different physical hosts, i.e. inter-host VM communication. The communication paths [9, 10] followed by the packets (say, from VM1 to VM2) during an intra-host and inter-host communication is discussed in the following schemes.

Scheme 1 **(Same-host; same-VS; same-PG)**: As shown in Figure 4.2, VM1 (with vNIC1) and VM2 (with vNIC2) reside on same physical host and are connected to the same vSwitch (VS1) and same port group (PG A).
Flow: The network packets flow from vNIC1 to VS1 and from there they travel to vNIC2.
Network(s) used: Virtual

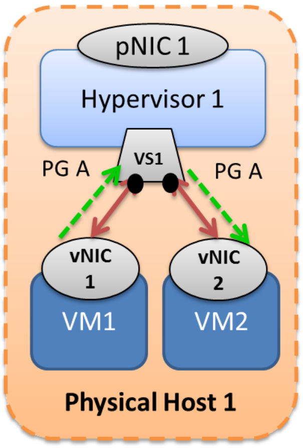


Figure 4.2: Scheme 1: Same-host; same-VS; same-PG

Scheme 2 **(Same-host; same-VS; diff-PG)**: As shown in Figure 4.3, VM1 (with vNIC1) and VM2 (with vNIC2) reside on same physical host and connected to same vSwitch (VS1). However, they are connected to different port groups, VM 1 to PG A and VM 2 to PG B.
Flow: The network packets flow from vNIC1 to VS1, and from VS1 the packets enter pSwitch using pNIC. From pNIC the packets travel back to VS1 and then to vNIC2.
Network(s) used: Virtual and physical

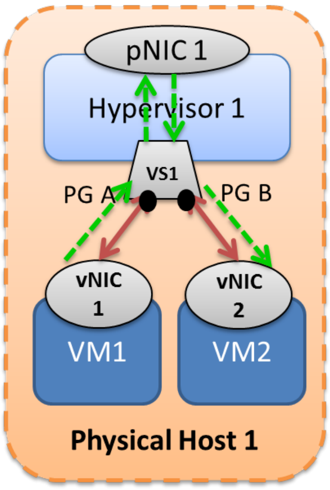


Figure 4.3: Scheme 2: Same-host; same-VS; diff-PG

Scheme 3 **(Same-host; diff-VS)**: As shown in Figure 4.4, VM1 (with vNIC1) and VM2 (with vNIC2) reside on same physical host; however they are connected to different vSwitches, VS1 for VM1 and VS2 for VM2. As they are connected to different vSwitches, the port group doesn't affect their communication path.
Flow: The network packets flow from vNIC1 to VS1, and from VS1 the packets enter pSwitch using pNIC. From pNIC the packets travel to VS2 and then to vNIC2.
Network(s) used: Virtual and physical

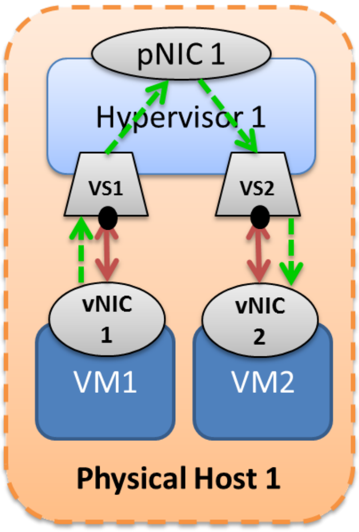


Figure 4.4: Scheme 3: Same-host; diff-VS

Scheme 4 **(Diff-host)**: As shown in Figure 4.5, VM1 (with vNIC1) and VM4 (with vNIC4) reside on different physical hosts; therefore they have different vSwitches, VS1 for VM1 and VS4 for VM4 and different pNIC, pNIC1 to VM1 and pNIC2 to VM4.
Flow: The network packets flow from vNIC1 to VS1, and from VS1 the packets enter pSwitch using pNIC1. Through pSwitch, the packets reach pNIC2. From here they travel to VS4, and then reach VM4 through vNIC4.
Network(s) used: Virtual and physical

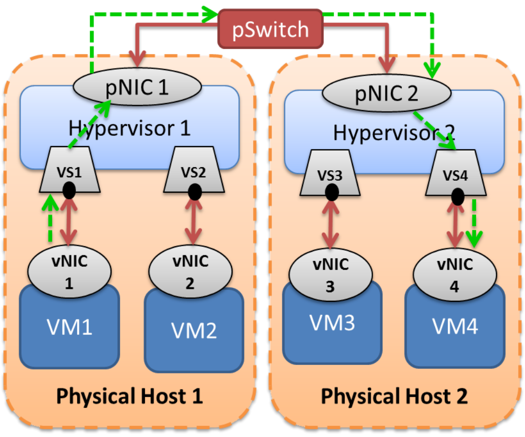


Figure 4.5: Scheme 4: Diff-host

From Figure 4.2 to Figure 4.5, we observe that the network packets can use virtual and/or physical networks to communicate with each other. In one of the schemes (i.e. Scheme 1 – Same-host; same-VS; same-PG), the network packets only use a virtual network and do not enter the hypervisor level. Thus, VMWatcher placed at the hypervisor might not be able to analyse some network packets, thus affecting the detection accuracy. To overcome this limitation, we have decided to place the VMWatcher inside each VM. It monitors the network traffic data at the VM level rather than at the hypervisor level. In addition, placing VMWatchers at the VM level distributes the monitoring workload among multiple VMs, instead of imposing the workload on a single hypervisor, enhancing the scalability of the solution.

4.2.2 VM Communication Patterns

This section describes the communication patterns which are used by bVMD framework to identify a suspicious activity. A communication pattern is associated to the communication flows between a set of VMs. It can be associated with a number of attributes such as: (1) the number of packets received from a peerVM within a time window, (2) the average time taken by a peerVM to send an acknowledgement to VMWatcher, (3) the number of packets unacknowledged by the peerVM in a time window, etc. A change in any of these attributes depicts a change in the communication pattern among the set of VMs. Communication patterns can be classified into two types, benign and malicious communication patterns.

- **Benign Communication Patterns**: A communication pattern is said to be benign if the measured attribute values are within certain thresholds over a period. That means VMs communicate with each other as per their expectations and there is no unexpected change (in the above mentioned attribute(s)) in an ongoing communication. In other words, if the communication pattern shows consistencies, the communicating VMs would trust each other and believe that they are not infected.
- **Malicious Communication Patterns**: A communication pattern is suspected to be malicious if communication flows exhibit some unexpected change (above a threshold value) in the above mentioned attribute(s) during an ongoing communication. An unexpected change in a communication pattern indicates that there are unsolicited communication packets, and this could be an indicator of a malware attack.
  As discussed in Chapter 3, bVMD Framework aim to identifying botcloud which perform DDoS attacks and the first attribute, i.e. the number of packets received within a time window, hereafter called Packet Arrival Rate from PeerVM, PARfPVM, is used for detecting malicious behaviour. This is because the DDoS attack is meant to send large number of packets to the targetVM and this would reflect in PARfPVM attribute. So, a sudden change in this attribute would trigger further actions in the Framework. This PARfPVM is calculated over a time window. The PARfPVM which is calculated in the current time window is called PARfPVM$_{\mathrm{curr}}/_{\mathrm{prev}}$. This is compared with the PARfPVM calculated in the previous time window, called, PARfPVM$_{\mathrm{curr}}/_{\mathrm{prev}}$ stored in VMT-T.

  $$if(PARfPVM_{curr} > PARf\Delta \times PARfPVM_{prev})$$

  is true, then it is classified as malicious pattern, otherwise it is classified as a benign pattern.

- If it is classified as benign pattern, then the VMT-T is updated with a new PARfPVM using the following equation.

$$PARfPVM_{curr} = \frac{PARfPVM_{prev} + PARfPVM_{curr}}{2} \tag{4.1}$$

  This $\mathrm{PARfPVM_{curr}}$ is stored in the VMT-T as $\mathrm{PARfPVM_{prev}}$

- If it is classified as a malicious pattern, then the $\mathrm{PARfPVM_{prev}}$ value stored in the VMT-T is not replaced by $\mathrm{PARfPVM_{curr}}$. This is to make sure that the system doesn't classify a malicious pattern as a benign pattern over a period.

Two major parameters which play a major role here are:

1 ΔPAR – This is the difference in the rate of a benign communication pattern and a malicious communication pattern. A low value can lead to high false positive detections, whereas a high value can lead to low detection rate. In the bVMD implementation, the Δ PAR value is assumed as 2.

2 Window size (WPAR) – This refers to a window size after which the PARfPVM is calculated and then compared with the one calculated in the previous window. In bVMD, we propose a flexible window size, $\mathrm{W_{PAR}}$, for each peerVM. This means that every pair of VM has different $\mathrm{W_{PAR}}$. This is because every pair of VMs has independent communication patterns, and thus a separate PARfPVM. A high PARfPVM might require a small $\mathrm{W_{PAR}}$, whereas a low PARfPVM might require a high $\mathrm{W_{PAR}}$. Therefore, we propose to use (Flexi -$\mathrm{W_{PAR}}$) for each peerVM. This might improve the efficiency of a VM as the computations to be performed for each VM at the end of each window is at different timings. In addition, it might help to reduce the burden on the VM at a given point in time.

When an unknown VM requests a communication, then its corresponding window size is calculated using the following ways.

- Along with the recommendation VMT value, the window size of peerVM is also communicated. This value is stored in VMT-T. If there is more than 1 peerVM which has recommended VMT for the unknown VM, then the final window size is calculated by taking an average of the recommended VMT value.
  For example, $\mathrm{VM_i}$ has two peerVMs, $\mathrm{VM_a}$ and $\mathrm{VM_b}$, and it requests them for PARfPVM for $\mathrm{VM_j}$. If it receives $\mathrm{PARfPVM_a{}^j}$ and $\mathrm{PARfPVM_b{}^j}$ from $\mathrm{VM_a}$ and $\mathrm{VM_b}$ respectively. It calculates $\mathrm{PARfPVM_i{}^j}$ using the following equation.

$$PARfPVM_{\mathrm{i}}{}^{\mathrm{j}} = \frac{\sum_{i=0}^{k}(PARfPVM_{\mathrm{a}}{}^{\mathrm{j}} + PARfPVM_{\mathrm{b}}{}^{\mathrm{j}})}{k} \tag{4.2}$$

- If there is no recommendation trust, the window size is calculated by taking an average of all the window sizes of its peerVMs.

Using the above example, if $\mathrm{VM_a}$ and $\mathrm{VM_b}$ doesn't have PARfPVM for $\mathrm{VM_j}$, then $\mathrm{VM_i}$ calculates $\mathrm{PARfPVM_i{}^j}$ using the following equation.

$$PARfPVM_{\mathrm{i}}{}^{\mathrm{j}} = \frac{\sum_{i=0}^{k}(PARfPVM_{\mathrm{i}}{}^{\mathrm{a}} + PARfPVM_{\mathrm{i}}{}^{\mathrm{b}})}{k} \tag{4.3}$$

When a known VM sends a communication, then the window size is updated at the end of each window. In bVMD Framework evaluation, the window size is kept constant.

4.2.3 Malicious Communication Pattern Quantification

This section describes the metric and the methods used for the quantification of malicious communication patterns in bVMD Framework.

- **Trust as a Quantification Metric**: A VMWatcher quantifies communication patterns of its peerVM using a generic parameter 'trust'. Here trust is defined as a level of confidence that a VM has on its peerVM that they have a benign communication pattern. A benign communication pattern increases the VMT value of the peerVM with every subsequent communication and on the other hand, a malicious communication pattern decreases its VMT value with every subsequent communication. A VM (which is regularly communicating) which has a higher VMT value means that it is more trustworthy, or less likely under an attack, whereas a VM with a lower VMT value means that it is less trustworthy, or more likely under an attack.
- **Direct and Recommended Trust Values**: The bVMD Framework uses two types of trust, direct and recommended trust [124]. Here direct trust is evaluated based on communication patterns between two known VMs (i.e. VMs which are already communicating) whereas recommended trust is evaluated when two unknown VMs (i.e. VMs which don't have a communication history) communicate. Recommendation trust involves a third party, which is a known VM to both unknown VMs. For example, if there are two unknown VMs, $\mathrm{VM_i}$ & $\mathrm{VM_j}$ and $\mathrm{VM_j}$ sends network packets to $\mathrm{VM_i}$, then $\mathrm{VM_i}$ receives a recommendation (in the form of VMT value) of $\mathrm{VM_j}$ from peerVM of $\mathrm{VM_i}$. Based on the recommended VMT values received, the VMT value of $\mathrm{VMT_i{}^j}$ is calculated as given follows [125].

$$VMT_{\mathrm{i}}{}^{\mathrm{j}} = \frac{\sum_{i=0}^{k}(Cred_{\mathrm{VM_k}} \times VMT_{\mathrm{k}}{}^{\mathrm{j}})}{k} \tag{4.4}$$

where, $\mathrm{VMT_i{}^j}$ is the VMT value of $\mathrm{VM_i}$ on $\mathrm{VM_j}$,
k is the number of peerVMs of $\mathrm{VM_i}$ which are peerVMs of $\mathrm{VM_j}$, and
$\mathrm{Cred_{VM_k}}$ is the credibility value of $\mathrm{VM_k}$. The credibility value is defined in Section 4.3.

- **VMWatcher Trust (VMT-T)**: Each VMWatcher maintains a VMT table storing the information related to each of its peerVMs. As shown in Table 4.1, the VMT table is captured using a number of attributes, namely, a VM unique IP address; its trust value, VMTi, number of packets received by the peerVM in previous time window (W_{curr-1}) and current time window (W_{curr}). The trust value, VMT_i, and frequency of communication, for each peer-VM are estimated by using the VM Trust Value Estimation (VMT-VE) method described in the next section.

Table 4.1: VMT Table stored by each VMWatcher

VM IP Address	VMT Value	Previous Packet Count (W_{curr-1})	Current Packet Count (W_{curr})	Window Size

4.2.4 VM Trust Value Estimation (VMT-VE) Method

This section presents details of VMT Value Estimation (VMT-VE) method used by the VMWatcher component. As mentioned earlier, one of the tasks each VMWatcher ought to perform in bVMD is to estimate a VMT value for each of its peerVMs. In this method, VMWatcher estimates a VMT value for each of its peerVMs based on observing their communication patterns. At each VMWatcher, VM_i, the VMT value assigned to a peerVM, VM_j, denoted by $VMT_k{}^j$, reflects the confidence of VMWatcher VM_i on VM_j that it is not a malicious VM. The higher the VMT value of a peerVM, the higher the confidence the VMWatcher has on the peerVM. A VMT value is measured in a given range [VMT_{min}, VMT_{max}], where $VMT_i{}^j{}_{min}$ is the minimum, and $VMTVMT_i{}^j{}_{max}$ is the maximum, VMT value.

The VMT-VE consists of three algorithms, VMT-Initialization (VMT-I) method, VMT-Update (VMT-U) method and VMT-Exchange (VMT-Exc) method.

- **Virtual Machine Trust Initialisation (VMT-I)**: The VMT-I algorithm is used to initialise the VMT value when a VM initiates a connection with the VMWatcher. In other words, the VMT-I algorithm is used when a VMWatcher receives a packet from an unknown VM. In this case, if there are any peerVMs, then the VMWatcher, VM_i, calculates $VMT_i{}^j$ uses the recommendation trust value given in Equation (4.4).

  However, if there are no peerVMs, then the VMWatcher, VM_i, calculates $VMT_i{}^j$ uses the following Equation.

$$VMT_{\mathrm{VM_i}^{\mathrm{j}}} = \frac{VMT_{\mathrm{min}} + VMT_{\mathrm{max}}}{2} \tag{4.5}$$

This neutral value (calculated by Equation (4.5)) indicates that VMWatcher, $\mathrm{VM_i}$ cannot yet determine whether this new peerVM is trustworthy or not. With subsequent communication, the value of $\mathrm{VMT_i}^{\mathrm{j}}$ is updated by $\mathrm{VM_i}$. The bVMD is based on Marsh's trust model [126], thus $\mathrm{VMT_{max}}$ value is set to 1 and the $\mathrm{VMT_{min}}$ value is set to -1 [127][128][129][130]. This algorithm is called whenever there is a packet received from an unknown VM.

- **Virtual Machine Trust Update (VMT-U)** : The VMT-U method is used to update VMT values of known VMs (or peerVMs). The update is based on the subsequent communication patterns of peerVMs. As discussed in Section 4.2, the communication pattern of a VM can be classified into benign or malicious pattern. A benign communication pattern increases the VMT value of the VMWatcher by a trust factor of $\mathrm{VMT}\Delta_1$, i.e.

$$VMT_{\mathrm{i}}{}^{\mathrm{j}}{}_{(\mathrm{t=t_0})} = VMT_{\mathrm{i}}{}^{\mathrm{j}}{}_{(\mathrm{t=(t_0\text{-}1)})} + \Delta_1 \qquad if, k > 0 \tag{4.6}$$

And a malicious communication pattern decrements the VMT value of the VMWatcher by a trust factor of $\mathrm{VMT}\Delta_2$, i.e.

$$VMT_{\mathrm{i}}{}^{\mathrm{j}}{}_{(\mathrm{t=t_0})} = VMT_{\mathrm{i}}{}^{\mathrm{j}}{}_{(\mathrm{t=t_0\text{-}1})} - \Delta_2, \qquad if, k > 0 \tag{4.7}$$

Where, $\mathrm{VMT}\Delta_1$ and $\mathrm{VMT}\Delta_2$ are the trust factors. The VMT algorithm along with the VMT-I and VMT-U algorithms are given below.

Algorithm 1 The VMT-I Algorithm

```
Input: $\mathrm{Cred_{VM_k}}$ and $\mathrm{VMT_k}^{\mathrm{j}}$ for each peer-VM, $\mathrm{VM_k}$
Output: $\mathrm{VMT_i}^{\mathrm{j}}$

1: if $(k > 0)$ then
2:     $\mathrm{VMT_i}^{\mathrm{j}} = \frac{\sum_{i=0}^{k}(Cred_{\mathrm{VM_k}} \times VMT_{\mathrm{k}}{}^{\mathrm{j}})}{k}$
3: else
4:     $VMT_{\mathrm{i}}{}^{\mathrm{j}} = \frac{VMT_{\mathrm{min}} + VMT_{\mathrm{max}}}{2}$
5: end if
```

Algorithm 2 The VMT-U Algorithm

```
Input: $PARfPVM_{\text{curr}}, PARfPVM_{(\text{curr-1})}, PARfPVM\Delta, VMT\Delta_1, VMT\Delta_2, VMT_{\text{i}}^{\text{j}}{}_{(\text{t=t}_0\text{-1})}$
Output: $VMT_{\text{i}}^{\text{j}}{}_{(\text{t=t}_0)}$

1: if ($PARfPVM_{\text{curr-1}} >= PARfPVM\Delta \times PARfPVM_{\text{curr-1}}$) then
2:     $VMT_{\text{i}}^{\text{j}}{}_{(\text{t=t}_0)} = VMT_{\text{i}}^{\text{j}}{}_{(\text{t=t}_0\text{-1})} - VMT\Delta_2$
3:     if $VMT_{\text{i}}^{\text{j}}{}_{(\text{t=t}_0)} <= (-1)$ then $VMT_{\text{i}}^{\text{j}}{}_{(\text{t=t}_0)} = -1$
4:     end if
5: else
6:     $VMT_{\text{i}}^{\text{j}}{}_{(\text{t=t}_0)} = VMT_{\text{i}}^{\text{j}}{}_{(\text{t=t}_0\text{-1})} + VMT\Delta_1$
7:     if $VMT_{\text{i}}^{\text{j}}{}_{(\text{t=t}_0)} >= (+1)$ then $VMT_{\text{i}}^{\text{j}}{}_{(\text{t=t}_0)} = +1$
8:     end if
9: end if
```

- **Virtual Machine Trust Exchange (VMT-Exc) Method**: As discussed above, recommendation trust is used by VMWatcher's to exchange VMT values. This is useful in cases where two unknown VMs start communication and then a known peerVM can act as a third party and recommend VMT values. This process can be done by two methods.

  - *Synchronous method* – This is a method in which a VMWatcher sends a request to its peerVM asking for a recommendation VMT value about an unknown VM. The peerVM sends a response back with either a VMT value (if it has one) or no VMT value (if it doesn't have one). This method might minimise the storage overhead as a VMWatcher only stores a VMT value if it needs it, otherwise not. However, this involves an additional time for a VMWatcher to send a request and receive a response. This might be equal to ($2 \times T_{\text{comm}} + T_{\text{procc}}$), where T_{comm} is the communication time between the VMWatcher and its peerVM and T_{procc} is the processing time taken by peerVM to process the request and send a response. For an effective detection, this time should be less than the average time a botcloud attack lasts. However, this could be very unpredictable as attacks could last from a few minutes to months. Therefore, this method would not be effective in these scenarios.
  - *Asynchronous method* – This is a method in which VMWatchers exchange their VMT values regularly. This means that when a VMWatcher initiates or updates

a VMT value of a peerVM, it sends this value to other peerVMs. The receiving peerVMs, use recommendation trust discussed in Equation (4.4) to calculate a final VMT value. For example, as shown in Figure 4.6 there are four VMs in a cloud environment, namely $\mathrm{VM_a}$, $\mathrm{VM_b}$, $\mathrm{VM_c}$ and $\mathrm{VM_d}$. If $\mathrm{VM_a}$ has two peerVMs, namely $\mathrm{VM_c}$ and $\mathrm{VM_d}$ and an unknown VM, $\mathrm{VM_b}$ requests for a communication, then the below mentioned process is followed.

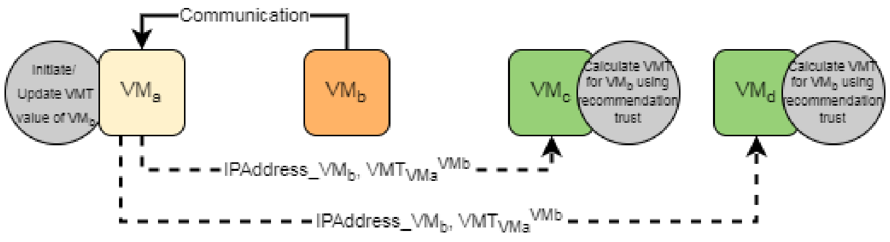

Communication
Initiate/ Update VMT value of VM_b
VM_a
VM_b
VM_c
Calculate VMT for VM_b using recommendation trust
VM_d
Calculate VMT for VM_b using recommendation trust
IPAddress_VM_b, VMT_VMa^VMb
IPAddress_VM_b, VMT_VMa^VMb

Figure 4.6: Exchange of VMT values

Firstly, $\mathrm{VM_a}$ calculates VMT value for $\mathrm{VM_b}$, i.e. $\mathrm{VMT_a}^\mathrm{b}$, using Algorithm 1. Secondly, it communicates this value to $\mathrm{VM_c}$ and $\mathrm{VM_d}$. Then they both check their VMT-T for $\mathrm{VMT_c}^\mathrm{b}$ and $\mathrm{VMT_d}^\mathrm{b}$ respectively.

- Say, if $\mathrm{VM_c}$ found a $\mathrm{VMT_c}^\mathrm{b}{}_{\mathrm{(curr)}}$ in its VMT-T, then it updates the VMT value using the following equation.

$$VMT_\mathrm{c}{}^\mathrm{b} = \frac{VMT_\mathrm{c}{}^\mathrm{b}{}_{\mathrm{(prev)}} + \frac{(Cred_{\mathrm{VM_a}} \times VMT_\mathrm{a}{}^\mathrm{b})}{2}}{2} \tag{4.8}$$

- Say, if $\mathrm{VM_d}$ didn't find $\mathrm{VMT_d}^\mathrm{b}$ in its VMT-T, then it calculates VMT value using the following equation.

$$VMT_\mathrm{d}{}^\mathrm{b} = Cred_{\mathrm{VM_a}} \times VMT_\mathrm{a}{}^\mathrm{b} \tag{4.9}$$

This method might have the following weaknesses. Firstly, regular sharing of the VMT values might increase the communication overheads. Secondly, it would increase the storage overhead at each VM, as the VMWatcher has to store VMT-values of unknown VMs. In addition, these overheads might increase with increase in number of VMWatchers. However, this method minimises the waiting time of a VMWatcher to get a recommendation value about another VM, thus minimising the overall detection time of the solution. In bMMD Framework, the VMWatcher uses this asynchronous communication approach.

4.2.5 VMWatcher Operations

The VMWatcher component can be placed on three types of VMs, namely target VM (targetVM), attacking VM (botVM) and standard working VM (stdVM). These are described below.

- VMWatcher-on-TargetVM: A target VM is a VM which is attacked by a botVM. VMWatcher on targetVM is responsible to rate its peerVMs and share these values with S-VMD.
- VMWatcher-on-BotVM: An attacking VM or botVM is a VM which is a member of the botcloud. It is responsible for attacking target VM(s) by sending high number of packets. In addition, VMWatcher on botVM is responsible to rate its peerVMs and share these rating values with S-VMD.
- VMWatcher-on-StdVM: Standard working VMs are the VMs which are neither attacked, nor an attacker. VMWatcher on stdVM rate their peerVMs and share these rating values with S-VMD.

In addition, the VMWatcher communicate to share VMT values. Figure 4.7 shows the packet which is exchanged between VMWatchers.

PeerVM IP Address	VMT value
	VMT_i^j
...	...
	VMT_i^n

Figure 4.7: V-to-V Packet

The V-to-V packet contains the IP address of the peerVMs whose value has to be sent and their respective VMT value. This packet is exchanged among VMWatchers after a fixed window size. In bVMD this window size was kept as a minute.

4.2.6 Actions Taken

Firstly, each VMWatcher identifies the peerVM which has negative VMT. The details of this identified peerVM is shared with the local S-VMD. Figure 4.8 gives the V-to-S packet which is used to share the details with S-VMD.

PeerVM IP Address	VMT value
	VMT_i^j
…	…
	VMT_i^m

Figure 4.8: V-to-S Packet

This packet contains the address of the VMWatcher which has a low VMT value, i.e. a negative VMT value. Say, if VM_i is sending this packet, then it contains the details of peerVMs of VM_i which have negative VMT values in VM_i VMT-T. This packet is sent to VMWatcher as per the Risk Level discussed below.
In a scenario where majority of bVMD components are compromised, the VMWatcher has a feature of locally protecting itself by taking special actions against suspicious VMs. These set of actions are decided by a factor called, Risk Level (RL). According to [131], the risk concept is broken down into two main criteria: (a) the probability, which is the possibility of an undesirable occurrence, such as a cost overrun, and (b) the impact, which is the degree of seriousness and the scale of the impact on other activities if the undesirable thing happens. Using a mathematical description, a risk is described as follows.

$$R = P * I \tag{4.10}$$

where R is the degree of risk, P is the probability of the risk occurring and I is the degree of impact of the risk. Here risk level is defined as a measure of malicious behaviour of a particular VMWatcher experienced by its peerVM. It is divided into three categories as described below.

- **High RL**: The scenario is categorised into high risk if all the following are true.
  - The VMWatcher's VMT value in peerVM's VMT-T is at its lowest, i.e. equal to (-1 in bVMD Framework).
  - The S-VMD has already been notified about this low VMT value, however it has not yet responded.
  - The VMWatcher is still attacking the peerVM and the peerVM cannot reduce the VMWatcher's VMT value anymore (because it is already at its lowest level).

  Actions Taken: The VMWatcher is suspended by the peerVM, i.e. all the communication to and from the VMWatcher are immediately stopped and a high alert notification is sent to the S-VMD. This action of suspending communication with VMWatcher is temporary and it is revoked after S-VMD responds.

- **Medium RL**: The scenario is categorised into a medium risk level if all the following are true.
  - The VMWatcher's VMT value in peerVM's VMT-T is below the average VMT value of the system; however, it should not be equal to minimum VMT value, i.e. VMT_{min}. In bVMD Framework it should be between (0, -1).

  Actions Taken: With every change in the VMT value of VMWatcher in VMT-T of peerVM, the S-VMD is notified.
- **Low RL**: The scenario is categorised into low risk level if all the following are true.
  - The VMWatcher's VMT value in peerVM's VMT-T should be above average VMT value of the system. In bVMD Framework it should be between [0, 1].
  - The VMWatcher is still attacking the peerVM. As a result, its VMT value in peerVM's VMT-T is reducing, compared to the previous time window; however, it is still above average VMT value.

  Actions Taken: Unlike other risk levels, the peerVM doesn't explicitly notify about this risk level to the S-VMD. It shares its VMT-T after the window size and continues to communicate with the VMWatcher.

This idea of Risk Level at VM level can be effective in the following scenarios. (i) When a botVM is only attacking one of its peerVM (targetVM) and it maintains a benign communication with other peerVMs (stdVMs). The stdVMs might have more influence on the credibility of botVM as compared to a single targetVM, thus resulting in a high credit value of botVM in the system. In such a scenario, this idea of risk level gives a free hand to the targetVM to notify S-VMD and protect itself locally and (ii) a scenario where S-VMD is compromised and it doesn't respond to a high risk scenario, the VMWatcher can locally protect itself by suspending all the communication from the botVM. This approach of using Risk Level at VMWatcher level can be seen as a VM-local level trust value approach.

4.3 Suspected VM Detector (S-VMD)

S-VMD is a bVMD component placed on each hypervisor. The details of S-VMD components are described in this section.

4.3.1 VMT Value Acquisition

Each S-VMD receives VMT values from all the VMWatchers on the same physical host, called local VMWatchers. These received VMT values are stored in a table, called S-VMT-T (Suspected-VMT-Table), shown in Table 4.3.

		IP Address of VM who is assigning the VMT value				**Credit Value**
		A	B	…	Z	
IP Address of VM who is receiving the VMT value	A		VMT_b^a		VMT_z^a	$Cred_{VMa}$
	B	VMT_{VMa}^{VMb}			VMT_z^b	$Cred_{VMb}$
	…					
	Z	VMT_a^z	VMT_b^z			$Cred_{VMz}$

Table 4.3: S-VMD Trust Table (S-VMT-T)

As shown in Table 4.3, VMT values received from VMWatchers and the calculated $\mathrm{Cred}_{\mathrm{VM_x}}$ is stored in this table. The $\mathrm{CredV_x}$ is calculated using Aggregated VMT Value Estimation method described in the next section.

4.3.2 Aggregated VMT (Agg-VMT) Value Estimation

As mentioned earlier, one of the tasks a S-VMD ought to perform in the bVMD framework is to estimate credibility for each of its local VMWatchers. This is done by using Credit Value ($\mathrm{CredV_i}$) estimation method. The CredV of a VMWatcher reflects the reliability of the VMWatcher in the Framework. The higher the CredV of a VMWatcher, the more reliable it is perceived in the system. The CredV of a VMWatcher is based on: 1) the VMT values given by its peerVMs and 2) the CredV of those peerVMs. These CredV are used for detecting suspected malicious VMs, i.e. the VMs with negative credibility. The credit value of a VMWatcher $\mathrm{VM_i}$, denoted by $\mathrm{Cred}_{\mathrm{VM_i}}$ is estimated as follows.

Initially, when $\mathrm{VM_i}$ newly joins the system, i.e. it has no communication history, a neutral value is assigned to $\mathrm{Cred}_{\mathrm{VM_i}}$, which is calculated using Equation (4.11).

$$Cred_{\mathrm{VM_x}} = \frac{Cred_{\mathrm{min}} + Cred_{\mathrm{max}}}{2} \tag{4.11}$$

Where $\mathrm{Cred}_{\mathrm{VM_{min}}}$ is minimum credit value and $\mathrm{Cred}_{\mathrm{VM_{max}}}$ is the maximum credit value. In the bVMD Framework, $\mathrm{Cred}_{\mathrm{VM_{min}}}$ is set to -1 and $\mathrm{Cred}_{\mathrm{VM_{max}}}$ is set to 1 [126] [130]. This neutral value (calculated in Equation (4.11)) indicates that the system cannot yet determine whether VMWatcher $\mathrm{VM_i}$ is trustworthy or not. Then, from this point, the value of $\mathrm{Cred}_{\mathrm{VM_i}}$ is updated whenever S-VMD receives a VMT value from a peerVM. If $\mathrm{VM_i}$ has ‘n’ peerVMs and each peerVM has a $\mathrm{Cred}_{\mathrm{VM_i}}$, then S-VMD calculates the credit value of $\mathrm{VM_i}$ using the following equations.

$$Cred_{\mathrm{VM_x}} = \frac{\sum_{i=1}^{n}(Cred_{\mathrm{VM_a}} \times VMT_{\mathrm{VM_x}}{}^{\mathrm{VM_a}})}{n} \tag{4.12}$$

if $n > 0$, i.e. the VM has at least one peerVM.

If VMWatcher doesn't have any peerVM, then its CredV is calculated using Equation (4.11). This is because a VM with no peerVM means that the VM is not communicating and thus its CredV is assigned as a neutral value.

4.3.3 S-VM Detection and Actions

After receiving the VMT values from peerVMs, the S-VMD checks the Risk Level (RL) specified by the peerVM. If a peerVM has flagged a high RL, then S-VMD calculates the CredV of VMWatcher immediately, otherwise if it has flagged a medium RL, S-VMD calculates CredV of all the VMWatchers immediately and in case of low RL, S-VMD calculates credit value after a fixed time window. The size of this fixed window might affect the time of S-VM detection. A big window size might delay the detection process if there are frequent attacks. On the other hand, a small window size might introduce a high computational cost. Therefore, to overcome this issue, we used the above mentioned idea of RL. Frequent attacks mean a high RL and therefore a minimal window size. Less frequent attacks mean a lower RL, and therefore a bigger window size. In the evaluation, it is assumed the window size equal to the communication time, i.e. every time there is a communication (and this value is kept constant).

After determining the CredV, a VM is classified as a suspected malicious VM (S-VM) if it has CredV lesser than an average Cred value. In other words, a VM with negative VMT value is classified as S-VM. Once the S-VMs are identified, then their local bot-VMDs are notified. This is done by sending a packet with the IP Address of the S-VM along with its CredV. In addition, the details of a sub-set of its peerVMs are also sent. This sub-set contains the peerVMs which satisfy either one of the following: 1) High VMT (close to 1) on the identified S-VM, 2) identified S-VM has high VMT (close to 1) on VM, and 3) the peerVM has a negative CredV.

To improve the efficiency of bVMD Framework, S-VMs have a feature of grouping VMWatchers based on their credit values. This means that VMs with high credit value (most likely group of benign VM) are grouped together and those with low credit value (most likely group of botVM) are grouped separately. As botVMs are designed to work in groups, this idea of grouping at S-VMD might be very useful. Even if S-VMD incorrectly classified the groups, the groups can be classified correctly after stage-2.

Grouping at S-VMD is based on the MCC values (described in Chapter 3) of VMWatchers. The MCC values range from -1 to +1 and their meanings are described below in Table 4.4.

MCC value	**botVM**	**targetVM**	**Meaning**
Positive	Negative	Positive	Correct detection
Negative	Positive	Negative	Wrong detection
Zero	Negative	Negative	No detection
	Positive	Positive	

Table 4.4: MCC value analysis for grouping VMWatchers

- Positive MCC value: A positive MCC is calculated if the botVM have negative CredV and targetVM has positive CredV. In this case, the groups of botVM and targetVM are correctly classified and the correct group is detected.
- Negative MCC value: A negative MCC is calculated if the botVM has positive CredV and targetVM has negative CredV. In this case, the groups of botVM and targetVM are correctly classified, however they are incorrectly identified. That means botVM group is detected as malicious and targetVM group is detected as botVMs.
- Zero MCC value: A zero MCC is calculated if both the groups of botVM and targetVM either have positive CredV or negative CredV. In this case, S-VMD cannot classify them into groups. However, S-VMD can identify the cases where all the VMWatchers have negative CredV and marks it as an alarming scenario. Thereafter, it analyses the current VMT values of all the VMWatchers and try to find a pattern for grouping them. If it doesn't find a pattern, then it triggers an alarm to stage-2 of bVMD Framework.
  The scenario with all the positive CredV cannot be detected by S-VMD as this resembles a normal scenario with no botVMs.

4.4 Operations of bVMD stage-1 Components

This section describes the operations of the Stage-1 components, i.e. VMWatcher and S-VMD, of the bVMD architecture.

Figure 4.9 illustrates a scenario with two physical hosts, X and Y. Each physical host hosts an S-VMD, S-VMD 1 and S-VMD 2, placed in the hypervisor. Host X has two VMWatchers, namely VM_a and VM_b and host Y has one VMWatcher, namely VM_c. VMa has two peerVMs, VM_b and VM_c, VM_b has VM_a as its peerVM and VM_c has VM_a as its peerVM.

Each VMWatcher has the following components: (1) VMT-Initialize (VMT-I) (2) VMT-Update (VMT-U), (3) VMT-Exchange (VMT-Exc), and (4) VMT-table (VMT-T). Each S-VMD has the following components: (1) CredV Estimation method, (2) S-VM Detection method, and (4) S-VM trust table (S-VMT-T).
In this scenario, there is a regular communication between the pair (VM_a, VM_b) and the pair (VM_a, VM_c). Using VMT-Exc method, VM_a sends ${VMT_{VM_a}}^{VM_c}$ to VM_b and ${VMT_{VM_a}}^{VM_b}$ to VM_c. These VMT values received by VM_b and VM_c are used to calculate recommendation VMT values using Equation 4.1.

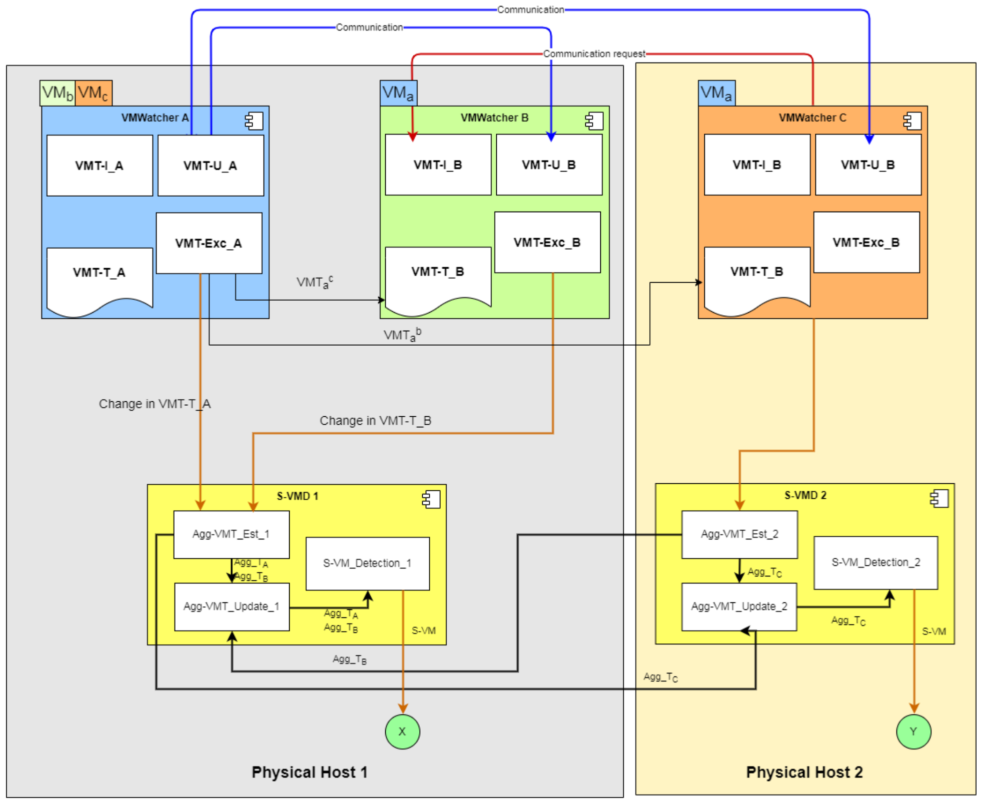


Figure 4.9: Working of Stage-1 components of bVMD Architecture

As VM_c sends a communication request to VM_b, this request is received by VMT-I at VM_b and the following steps are followed.

Algorithm 3 Working of VMWatcher component

```
Input: Communication packet from $VM_c$ to $VM_b$
Output: $VMT_c^b$

1: VMT-I at $VM_b$
2: if $VMT_b^c$ != 0 then
3:     goto Step 11
4: else
5:     if $VMT_a^c$ != 0 then
6:         $VMT_b^c = Cred_{VM_a} \times VMT_a^c$
7:     else
8:         $VMT_b^c = \frac{(VMT_{min} + VMT_{max})}{2}$
9:     end if
10: end if goto Step 24

11: VMT-U
12: if $PARfPVM_{curr-1} >= PARfPVM\Delta \times PARfPVM_{curr-1}$ then
13:         $VMT_b^c{}_{(t=t_0)} = VMT_b^c{}_{(t=t_0-1)} - VMT\Delta_2$
14:     if $VMT_b^c{}_{(t=t_0)} <= (-1)$ then
15:         $VMT_b^c{}_{(t=t_0)} = -1$
16:     end if
17: else
18:     $VMT_b^c{}_{(t=t_0)} = VMT_b^c{}_{(t=t_0-1)} + VMT\Delta_1$
19:     if $VMT_b^c{}_{(t=t_0)} >= (+1)$ then
20:         $VMT_b^c{}_{(t=t_0)} = +1$
21:     end if
22: end if

23: VMT-Exc
24:     Send $VMT_b^c$ from $VM_b$ to S-VMD1
25:     Send $VMT_b^c$ from $VM_b$ to $VM_a$

26:     $VM_a$ sends its VMT-T to S-VMD1
27:     $VM_b$ sends its VMT-T to S-VMD1
28:     $VM_c$ sends its VMT-T to S-VMD2
```

Algorithm 4 Working of S-VMD component

Input: VMT-T Packets at S-VMD1 and S-VMD2

Output: IP Address of S-VMs

```
1: CredV Est
2: i = 3
3: while (i!=0) do
4:     if (i=3) then
5:         x=a
6:     else if (i=2) then
7:         x=b
8:     else
9:         x=c
10:    end if
11:    if (VM_x doesn't have peerVMs) then
12:        $Cred_{\mathrm{VM_x}} = \frac{Cred_{\min}+Cred_{\max}}{2}$
13:    else
14:        $Cred_{\mathrm{VM_x}} = \frac{\sum_{i=1}^{n}(Cred_{\mathrm{VM_a}} \times VMT_{\mathrm{VM_x}}{}^{\mathrm{VM_a}})}{n}$
15:    end if
16:    i = i - 1
17: end while
18: if ($CredV_{\mathrm{VM_a}} < 0$) then
19:    Send IP Address of $\mathrm{VM_a}$ to botVMD
20: end if
21: if ($CredV_{\mathrm{VM_b}} < 0$) then
22:    Send IP Address of $\mathrm{VM_b}$ to botVMD
23: end if
24: if ($CredV_{\mathrm{VM_c}} < 0$) then
25:    Send IP Address of $\mathrm{VM_c}$ to botVMD
26: end if
```

One S-VMD received the VMT values from VMWatchers, it calculates their CredV using Equation (4.12) and identifies the S-VMs with negative CredV. These identified S-VMs are notified to their respective bot-VMD.

4.5 Experimental Evaluation

This section reports the experiments carried out to evaluate the performance of Stage-1 components of the bVMD Framework. It first describes the environment under which the experiments were carried out and then discusses the experimental results and findings.

4.5.1 Experimental Environment

We have used Omnet++ [32] [110] [115] to evaluate Stage-1 components of bVMD Framework. The details of Omnet++ are already discussed in Chapter 3. Experiments are carried out to determine: 1) Delta Values, 2) Number of VMs and 3) Time interval of each simulation run. In these experiments, performance of bVMD Framework in terms of Matthews Correlation Coefficient (MCC) curve (described in Chapter 3) is taken into account. The percentage of each VM type, i.e. botVM, targetVM, stdVM, and the number of peerVM of each VM is chosen from the cases discussed in Chapter 3. The six scenarios, A-F (Table 3.1) with different parameter settings are given in Appendix A.

4.5.1.1 Determining Delta Values

As discussed above, delta values (VMTΔ_1 and VMTΔ_2) are used in bVMD Framework to increase and decrease the VMT values. Delta values are measured in a given range VMTΔ_{min}, VMTΔ_{max}, where VMTΔ_{min} and VMTΔ_{max} is respectively the minimum and maximum VMT delta value. Similar to VMT value range, VMTΔ_{min} value is set to -1 and the VMTΔ_{max} value is set to 1. However, VMTΔ_2 is aimed to decrease a VMT value, therefore the range of delta values below zero is not functional. The final range of delta values is [0, 1].

For the evaluation of determining the delta values, cases 3, 6 and 12 out of 1-18 are chosen. The reason for selecting these cases is described next.

Cases with 100% peerVM mean that every VM affects every other VM in the network. This might lead to high dependency on the performance of each other. On the other hand, 50% and 10% peerVM will have less dependent VM nodes. Therefore to ensure that our results include the effect of every VM, we have chosen cases 3, 6, 9, 12, 15 and 18. In addition, to further ensure that the results include the effect of each VM type, we have chosen the cases which have at least one VM with all the three VM types. Hence, cases 3, 6 and 12 are chosen for this evaluation study. These experiments are carried out only once. The reason for this decision is explained in Section 4.5.1.4.

The performance of the bVMD Framework is analysed with 30 sets of delta values. These pair of delta value VMTΔ_1, VMTΔ_2 is formed using the following steps.

Step 1 *VMTΔ_1 value* – As discussed above, VMTΔ_1 ranges between 0 and 1. However a zero delta value might not be useful in the evaluation, so the minimum value is adjusted to 0.01 instead of 0. Then starting from 0.01 (close to 0), it is increased by VMTΔ till VMTΔ_{max}, i.e. 1. This VMTΔ is set at 0.1 in the bVMD Framework.

Step 2 *VMTΔ_2 value* – VMTΔ_2 is calculated using a minimum, average and maximum value of VMTΔ range. These three values are considered so that the performance can be studied at boundary values and at a central value.

$$[VMT\Delta 1 < VMT\Delta 2][132] \tag{4.13}$$

Therefore, VMTΔ_2 is calculated using the following equations.

- (Minimum) VMTΔ_2 is equal to VMTΔ_1
- (Average) VMTΔ_2 = floor((VMTΔ_2+VMTΔ_{max})/2, 0.1)
- (Maximum) VMTΔ_2 = VMTΔ_{max}

Table 4.5: Delta Cases

Delta_Case_Number (DCN)	**DCN Group**	**VMTΔ1**	**VMTΔ2**		**Delta_Case_Number (DCN)**	**DCN Group**	**VMTΔ1**	**VMTΔ2**
1		0.01	0.01		16		0.50	0.50
2	DCN_I	0.01	0.50		17	DCN_VI	0.50	0.70
3		0.01	1.00		18		0.50	1.00
4		0.10	0.10		19		0.60	0.60
5	DCN_II	0.10	0.50		20	DCN_VII	0.60	0.80
6		0.10	1.00		21		0.60	1.00
7		0.20	0.20		22		0.70	0.70
8	DCN_III	0.20	0.60		23	DCN_VIII	0.70	0.80
9		0.20	1.00		24		0.70	1.00
10		0.30	0.30		25		0.80	0.80
11	DCN_IV	0.30	0.60		26	DCN_IX	0.80	0.90
12		0.30	1.00		27		0.80	1.00
13		0.40	0.40		28	DCN_X	0.90	0.90
14	DCN_V	0.40	0.70		29		0.90	1.00
15		0.40	1.00		30	DCN_XI	1.00	1.00

The final values of VMTΔ_1 and VMTΔ_2 are given in Table 4.5.

The number of VMs in the evaluation ranges from 3 to 50. A minimum of 3 VMs is chosen so that there is at least one VM type in each evaluation. This is to make sure that the results are based on diverse node types. The maximum number of 50 VMs.

The time of each simulation run was chosen as 3600 seconds. This is because in one of the papers [133], the authors reported that in the year 2012-13 an average internal DDoS attack occurred for 3060 seconds. In addition, another report stated that most of the DDoS attacks in 2018-2022 lasted for 1800 seconds to 3600 seconds. Therefore, we have chosen the maximum of these values.

A total of 144 experiments were carried out for each DCN. This means that for each DCN and for each set of VMs, ranging from 3 to 50, three cases (3, 6 and 12) were implemented for a duration of 3600 seconds. Figure 4.10 depicts the average MCC value calculated. Every point of blue (case 3), red (case 6) and green (case 12) curves in the graph represents an average MCC value of 48 set of VMs for 3600 seconds, for a particular case. The purple curve (Average) depicts the average of all the three cases. In the graph, the x-axis shows the DCN and y-axis shows the average MCC values. As discussed, a high MCC value (closer to 1) shows better detection.

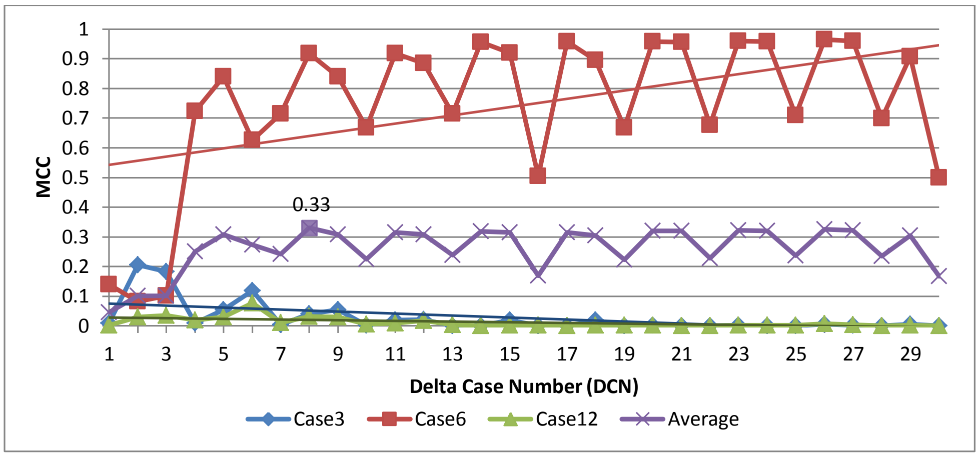


Figure 4.10: Average MCC value of all the delta cases

From Figure 4.10, we make the following observations.

Obs 1 The results show that majority (90.91%) of the cases with unequal delta values show better results than the other cases in their group (DCN Group), i.e. cases with equal delta values. This means that DCN 2 and 3 show better results than DCN 1, 5 and 6 show better results than DCN 4 and so on. In addition, the above results also depict that out of the two cases with unequal delta values in a group, the case with an average VMTΔ2 value depicts better results. This means that DCN 5, 8, 11, 14, 17, 20, 23, and 26 gives best results in their respective DCN groups. These results are in line with our thoughts that 1) delta values should not be equal to each other, and 2) they should not be equal to a boundary value (here 'boundary value' means VMT_{max} and VMT_{min}. The reasons for these two thoughts are given below.

(a) *Equal delta values*: Cases with equal delta values might not be effective in scenarios where a set of botVMs attack a set of targetVMs and at the same time they maintain a regular communication with a set of stdVMs. In this scenario, if the number of stdVMs equals or exceeds the number of targetVMs, then they (stdVMs) will be able to overpower CredV estimation of botVMs. This is because the rate with which the VMT value of a botVM is increased (by targetVMs) is equal to the rate with which its VMT value is increased by stdVMs.

(b) *Extreme delta values*: Cases using extreme delta values (VMTΔ_{min} & VMTΔ_{max}) might introduce high false detection rates. The use of these extreme delta values means that instead of a steady increase/drop in the VMT value with every communication, there will be a steep increase/drop change. Therefore, even a slight variation in the communication pattern might be wrongly classified as a botVM/benign VM.

Obs 2 Figure 4.10 shows that case 6 gives better results than cases 3 and 12. This is because of the obvious reason that case 6 has more targetVMs than botVMs. In addition, we observe that Case 6 gives better results with higher than average VMTΔ value, whereas cases 3 and 12 gives better results with lower than average number of VMTΔ values. This is because of the same reason as discussed above, i.e. case 6 have more targetVMs which are correctly identified, thus leading to high number of true positives as compared to cases 3 and 12.

Obs 3 The results show that DCN 8 has the highest average MCC value of 0.33. Therefore, in this thesis evaluation, we use DCN 8 delta range value. The values are: VMTΔ_1=0.2 & VMTΔ_2=0.6.

4.5.1.2 Determining the Number of VMs

To determine the number of VMs for collecting performance results for the bVMD Framework, we calculate the MCC values for 48 VMs (3-50) over a time of 60 minutes. The aim of plotting these graphs is to identify a point (number of VMs) from where the performance results are constant. For this evaluation, the 18 cases (mentioned in Table 3.5) are considered. These cases can be differentiated based on the number of peerVMs, 10%, 50% and 100% peerVMs. Out of these cases, the ones with 100% peerVMs (Cases 3, 6, 9, 12, 15 and 18) are chosen for this evaluation. This is to ensure that the performance results are affected by all the VMs in the system. For the evaluation, delta values are kept constant at VMTΔ_1=0.2 and VMTΔ_2=0.6 (as determined in Section 4.3.1). The MCC values are depicted below in Figure 4.11-4.16. Every point in these figures shows the MCC value (y-axis) of a particular case for 'n' number of VMs. This is recorded for 60 minutes (x-axis).

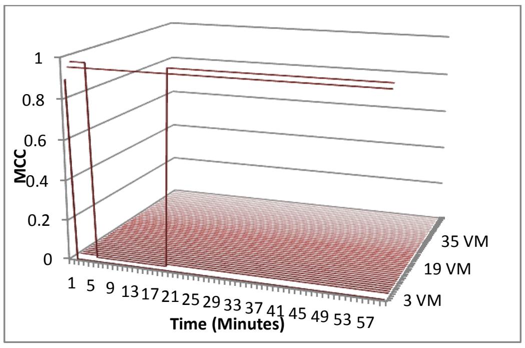


Figure 4.11: Average MCC value for Case 3

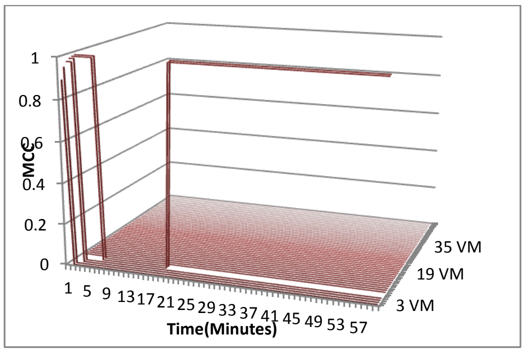


Figure 4.14: Average MCC value for Case 12

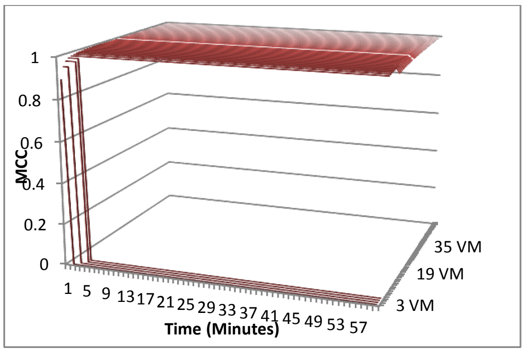


Figure 4.12: Average MCC value for Case 6

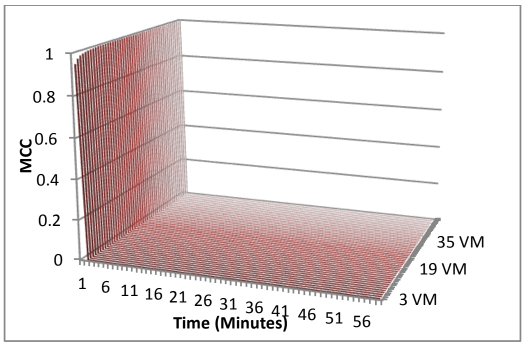


Figure 4.15: Average MCC value for Case 15

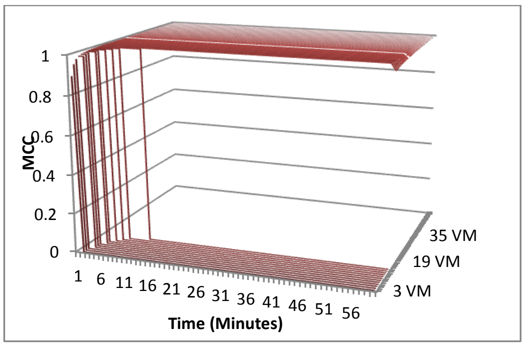


Figure 4.13: Average MCC value for Case 9

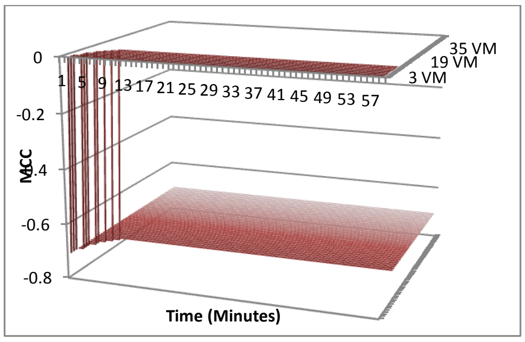


Figure 4.16: Average MCC value for Case 18

From the results in Figure 4.11 –4.16, we can see that the MCC value is constant after a certain number of VMs and a certain time. This is because with a regular communication, the VMT values of the VMs have stabilized to $\mathrm{VMT}_{\mathrm{max}}$ or $\mathrm{VMT}_{\mathrm{min}}$. A further increase in the amount of communication (in terms of increasing the number of VMs or time) cannot further increase or decrease the VMT value. This is due to a regular communication pattern used for our evaluation and any change in the communication pattern changes the results. From Figure 4.11, we can see that the MCC value is constant from 6 numbers of VMs (i.e. VM 6). Similarly in Figure 4.12 – 4.16, the results are constant from VM 8, 15, 9, 3 and 15. Out of these values, we chose the highest number of VMs so that all the cases have constant results from that point. The highest number is 15. Therefore, for all the computations in this thesis we have considered number of VMs as 15.

4.5.1.3 Determining the time interval for each simulation

To determine the duration of each simulation run, a similar approach is applied. We analyse the results of cases 3, 6, 9, 12, 15 and 18 and record the time from when the results are constant.

From the results in Figure 4.11, we can see that the MCC values are constant from 3 minutes. Similarly, in Figure 4.12 – 4.16, the results are constant from 19, 7, and 10 minutes respectively. The maximum time after which the results are constant is 19 minutes, therefore for all the computations in this this work, we have considered the time window of 19 minutes.

4.5.1.4 Determining Number of Simulation Runs

A simulation result is produced by taking an average of the data collected from ‘n’ independent simulation runs. For a reliable simulation, the value of ‘n’ should be sufficiently large. We have used 10 simulation runs for each experiment. However, for experiments determining the delta values, the number of VMs and the time duration of each simulation run, every experiment is simulated only once. This is because with every simulation run the position of botVM, targetVM and stdVM changes. The change in position of these VMs changes the type of peerVMs in every simulation run. This means that a targetVM might have botVM as a peerVM in one simulation run, while it might not have it in the next simulation run. This change would not affect cases 3, 6 and 12 (cases used for the above mentioned experiments) as they have 100% peerVM, therefore a single simulation run is used.

4.6 Evaluation of Results and Discussion

This section describes the experiments carried out to evaluate the performance of Stage-1 of the bVMD Framework. The results show MCC values for all the 18 cases evaluated with VMTΔ_1 and VMTΔ_2 values as 0.2 and 0.6 respectively. Each simulation has 13 sets of VMs (3-15 VMs described in Section 4.3.2) and it runs for 19 minutes . The x-axis depicts the time (in minutes) and y-axis depicts the average MCC value. Each point in the graph shows an MCC value (y-axis) of 'x' number of VMs at time 't' (x-axis). Prior to discussing the results of each case, we discuss a phenomenon observed in these cases. This is called the Dual Effect of VMT value (DEofVMT) and it is explained below.

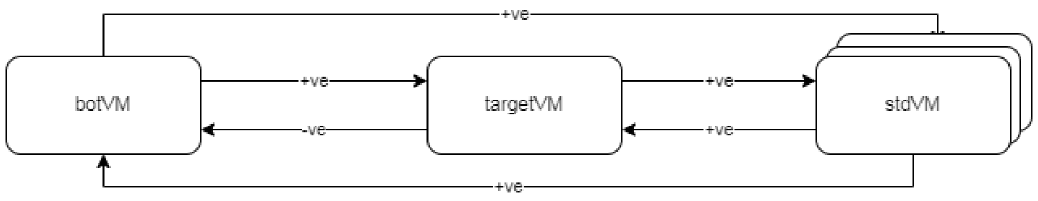


Figure 4.17: Dual Effect of a VMT value

Figure 4.17 shows a scenario with botVM(s), targetVM(s) and stdVM(s). In the figure, '+ve' refers a positive VMT value and '-ve' refers to a negative VMT value. In the scenario, the botVM attacks the targetVM and it maintains a regular (benign) communication pattern with the stdVMs. This means that the botVM receives positive VMT value to from stdVMs (peerVMs) and negative VMT value from targetVM. With time, this negative VMT has the following effects.

1 It directly affects the CredV of the botVM

2 It indirect affects the CredV of the peerVMs of the botVM's. This is because their (peerVMs of the botVM) CredV is dependent on the CredV of botVM. This indirect effect propagates through peerVMs of the botVM and then their peerVM and so on. That means the more peerVMs, the larger number of VMs are affected by this value.
Similarly, a positive VMT value also propagates in a similar manner and affects the CredV of peerVMs. However due to Equation (4.13), the Dual Effect of Negative VMT (DEofNVMT) is larger than Dual Effect of a Positive VMT (DEofPVMT).

The results of each case are described next. In these discussions, a high MCC value refers to value close to 1 (i.e. between 0.91 to 1), a medium MCC value refers to value close to zero and a low MCC value refers to value close to -1 (i.e. between -0.91 to -1). These MCC values are calculated for 13 sets of VMs (3-15 VMs) for a duration of 19 minutes.

Scenario A: With 1% botVMs, 1% targetVM and 98% standardVM: It consists of cases 1, 2 and 3.

- *Case 1 (Scenario A& SE1): With 1% botVMs, 1% targetVM and 98% standardVM and 1% peerVM*

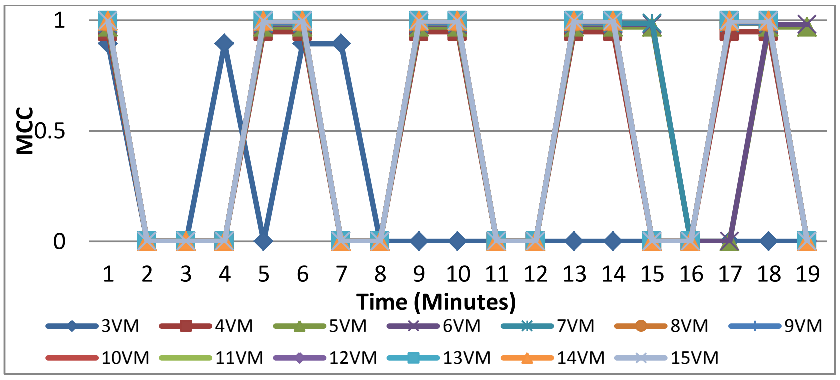


Figure 4.18: MCC values for Case 1

From Figure 4.18, we observe that the MCC value fluctuates between a medium and a high value. A high MCC value is achieved as each botVM receives a negative VMT from targetVM (1%) and a positive VMT value from the same number of peerVMs, therefore due to direct DEofVMT, the CredV of botVM is positive. On the other hand, targetVM receives a positive VMT from peerVM (1%), therefore its CredV is positive. A negative CredV of botVM and a positive CredV of targetVM results in positive MCC value.

A zero MCC value is achieved as the CredV of both targetVM and botVM decreases to a negative value, This is because with time the DEofNVMT propagates and the CredV of targetVM reduces, however it is still a positive value. On the other hand, DEofPVMT propagates and the CredV of botVM increases.

With time, the CredV of targetVM is always positive, however the CredV of botVM fluctuates between a positive and a negative VMT value. This fluctuation is because of DEofVMT. From Figure 4.19 and 4.20, we observe that the CredV value of cases with set of 3 VMs and set of 15 VMs. It can be seen that with increase in the number of VMs, the fluctuations in CredV increases. This is because, with increase in the number of VMs, we can observe the following.

1 The indirect DEofNVMT is spread across a larger section, therefore the CredV of targetVM and stdVM is always positive as the number of VMs increase.

2 The botVM receives direct DEofNVMT and indirect DEofPVMT with time, targetVM has VMT fluctuates between a medium and high VMT value and this puts

a direct DEofNVMT on botVM, thus the CredV of botVM fluctuated between high and low VMT value.

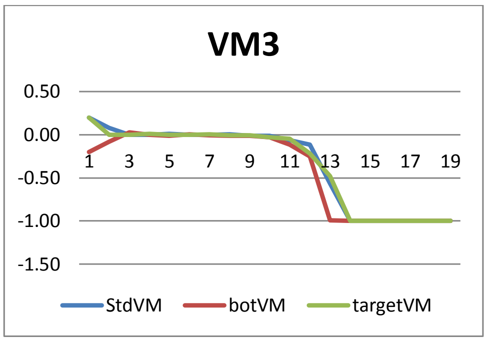


Figure 4.19: CredV value of all VM nodes with VM3 in Case 1

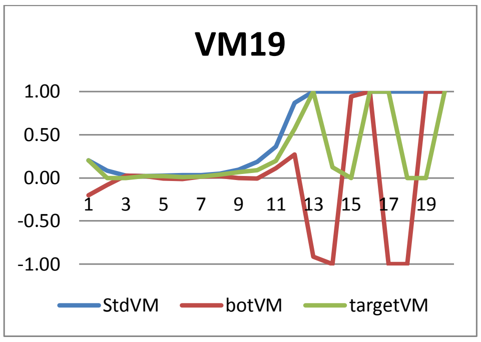


Figure 4.20: CredV values of all VM nodes with VM19 in case 1

The results show that the MCC value of all the VMs is always greater than or equal to zero. As the MCC values are the cases which can be correctly detected, bVMD can detect 46.55% sets.

- *Case 2 (Scenario A& SE2): With 1% botVMs, 1% targetVM and 98% standardVM and 50% peerVM*

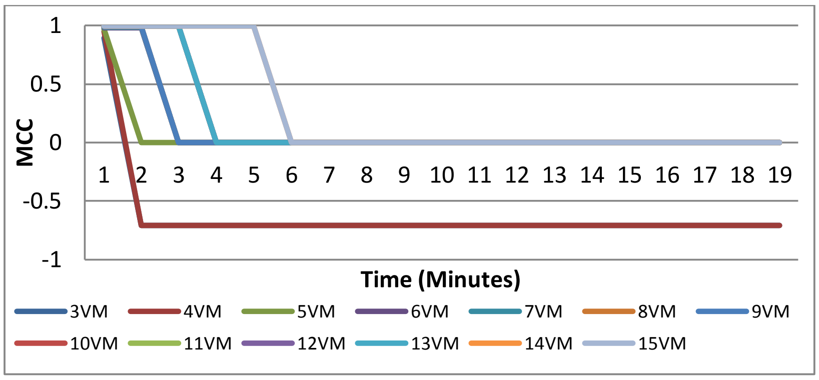


Figure 4.21: MCC values for Case 2

Figure 4.21 shows the MCC value of Case 2. From Figure 4.21, we make the following observations.

1 With an increase in number of VMs (greater than 4VMs), the MCC value is high for a maximum of 6 minutes and then stabilizes either at zero or a negative value. As compared to case 1, the values stabilize with time because of the increase in number of peerVMs. This results in wider propagation of DEofVMT values and thus the CredV stabilizes either at -1 or +1.

2 The sets with number of VMs between 4 to 7, have high MCC for a minute, sets with number of VMs between 8-13, have high MCC for 2 minutes and sets with number of VMs between 14 and 15 have high MCC for 3 minutes. This suggests that with an increase in the number of VMs, the MCC value is high for a longer duration. This is because with an increase in number of VMs, the indirect DEofVMT spreads across a larger group of VMs, therefore its effect is visible after a longer duration.

3 The sets with 3 set of VMs have 1 botVM, 1 targetVM and 1 stdVM. Here the DEofVMT propagates quickly due to a fewer number of VMs. Initially the targetVM gives negative VMT to botVM, so the MCC value is high. As the negative VMT value propagates, the CredV of targetVM is mostly affected in all the simulation runs (due to a smaller number of VMs). This reduces the CredV of targetVM to a negative value. Next time the targetVM gives a negative VMT to botVM, it turns out to be a positive CredV (due to negative CredV of targetVM). Therefore, the MCC value stabilizes at a negative value.

4 The bVMD can detect 100% of sets of VMs within a minute.

- *Case 3 (Scenario A& SE2): With 1% botVMs, 1% targetVM and 98% standardVM and 100% peerVM*

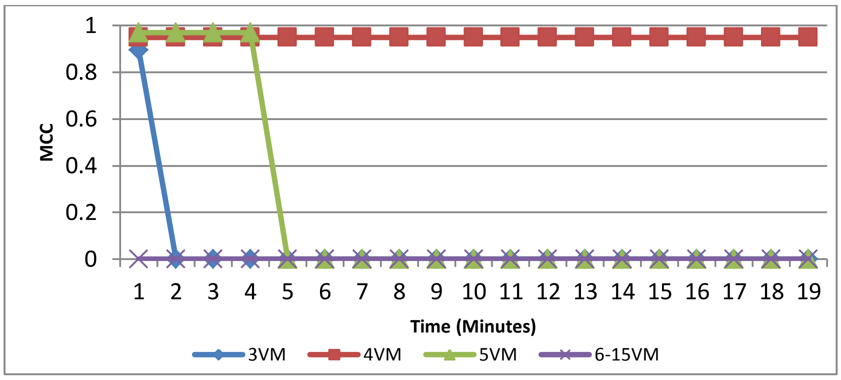


Figure 4.22: MCC values for Case 3

Figure 4.12 shows the MCC value of Case 3. The following differences that we observe are the following.

1 The time before the MCC value stabilizes at zero is less compared to that in Figure 4.21. In this case, the time is between 1-4 minutes as compared to 1-6 minutes in case 2. This reduction is because of the increase in the number of peerVMs, thus the indirect-DEofNVMT value propagates quickly. Although indirect-DEofPVMT also propagates at the same rate, due to Equation 2.9, DEofNVMT is more than DEofPVMT.

2 In this case, the sets with 3 and 4 number of VMs have a high MCC value. In case 3, the botVM receives negative VMT from targetVM (1%) and positive VMT from peerVM (98%, stdVM in this case). In case 3 and 4, there are 1 botVM, 1 targetVM and 1 or 2 stdVMs respectively. With low (1 or 2) number of stdVMs, the DEofNVMT overpower the DEofPVMT and therefore the CredV of botVM is negative. On the other hand, targetVM receives positive VMT from botVM and stdVM (all are peerVM). Its CredV is positive. With a positive CredV of botVM and a positive CredV of targetVM, the MCC value is positive.

3 In a set of 5 VMs, the stdVM is further increased by 1 stdVM as compared to Case 4. We can observe that the MCC is high for 4 minutes and then stabilizes at zero. This is because the positive VMT given by the additional stdVM (as compared to Case 3) increases the CredV of targetVM to a positive value. Therefore, it takes longer time for CredV to drop to a negative value.

4 The bVMD can detect 20% sets within first 6 minutes.

To further analyse the sets with zero MCC value, we study the CredV values of botVM and targetVM for 6VMs (minimum) and 15VMs (maximum). We have chosen 6VMs as this is the number from which the MCC value is initially a positive value and then stabilizes at zero.

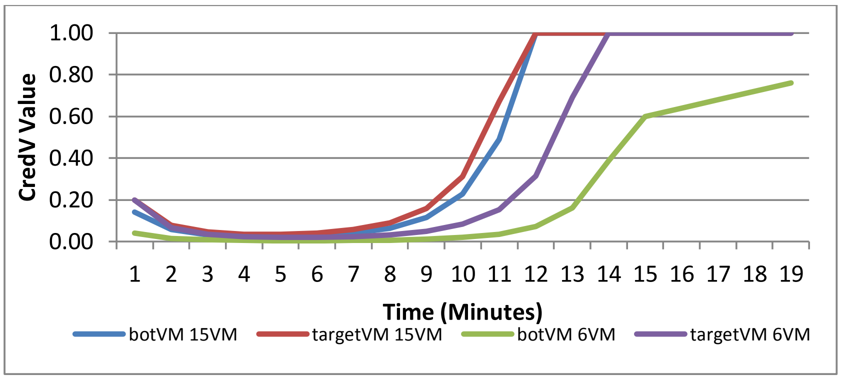


Figure 4.23: Detailed analysis of Case 3

From Figure 4.23 we can observe that with increase in number of VMs, the difference in VMT value of botVM and targetVM decreases, although the VMT value of targetVM is always more than the VMT value of botVM till they reach VMTmax. This is because with 100% peerVM, both botVM and targetVM receive similar positive VMT from all stdVMs. However, the negative VMT received by botVM from targetVM reduces its CredV. As the number of VMs increase, the higher is the VMT value of a targetVM and the direct-DEofNVMT is lower.

Scenario B: With 1% botVMs, 50% targetVM and 49% standardVM: It consists of Cases 4, 5 and 6.

- *Case 4 (Scenario-B & SE1): With 1% botVMs, 50% targetVM, 49% standardVM and 10% peerVM*

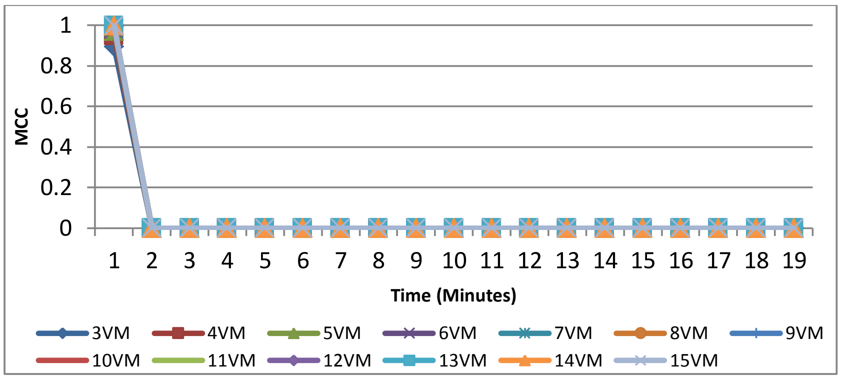


Figure 4.24: Detailed analysis of Case 4

Figure 4.24 shows the MCC value for case 4. From the figure, we can make the following observations.

1 For all the sets of VMs, the MCC value is high for a minute and then stabilizes at zero. This is because there are greater number of targetVMs than the number of botVMs. This means that a larger group of targetVMs give negative VMT to a smaller group of botVMs. This results in negative CredV of botVM and positive CredV of targetVM, thus a high MCC value.

2 The S-VMD can detect 100% sets within the first 1 minute.

- *Case 5 (Scenario-B & SE2): With 1% botVMs, 50% targetVM, 49% standardVM and 50% peerVM*

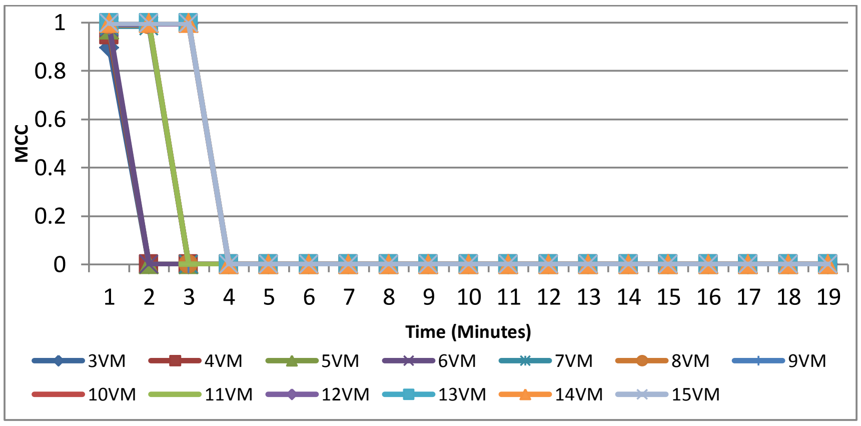


Figure 4.25: Detailed analysis of Case 5

Figure 4.25 (Case 5) shows similar results as compared to Figure 4.24 (Case 4). The differences we can observe are.

1 The time before the MCC value stabilizes at zero is between 1-4 minutes, whereas it is 1 minute in case 4. The increase in the duration is due to the increase in the number of peerVMs. That means botVM receive negative VMT from targetVM (50%) and positive VMT from similar number of peerVMs. Due to Equation 2.9, the CredV of botVM is negative.
On the other hand, targetVM receives positive VMT from peerVM (50%), thus a positive CredV. A negative CredV of botVM and a positive CredV of targetVM results in a positive MCC value.

2 With time, the DEofNVMT propagates through peerVMs (50%) and decreases the CredV of targetVM to a negative value. Therefore, the MCC value stabilize at zero.

3 The time taken by MCC value to stabilize at zero increases with an increase in the number of VMs. This is because with larger group of VMs, the probability of targetVM experiencing the indirect-DEofVMT is less.

4 The bVMD can detect 100 % sets within 4 minutes.

- *Case 6 (Scenario-B & SE3): With 1% botVMs, 50% targetVM, 49% standardVM and 100% peerVM*

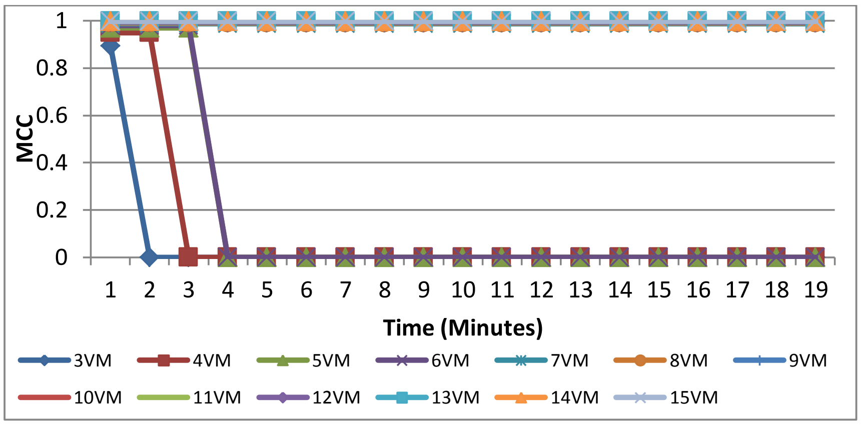


Figure 4.26: Detailed analysis of Case 6

From Figure 4.26 (Case 6) shows similar results as to Figure 4.25 (Case 5). The differences we can observe are.

1 Sets of VMs greater than 6VMs have high MCC value as compared to average MCC value in case 5. The reason behind this is increase in the number of peerVMs. With larger number of peerVMs, the CredV of targetVM doesn't drop to a negative value, as in case 5. Due to DEofNVMT on stdVMs, the CredV of botVM is negative

and this value continues to reduce as the number of stdVMs increase. This results in a positive MCC value.

2 The bVMD can detect 100% sets within 4 minutes.

Scenario C: With 1% botVMs, 99% targetVM and 0% standardVM: It consists of Cases 7, 8 and 9.

- *Case 7 (Scenario-C & SE1): With 1% botVMs, 99% targetVM, 0% standardVM and 1% peerVM*

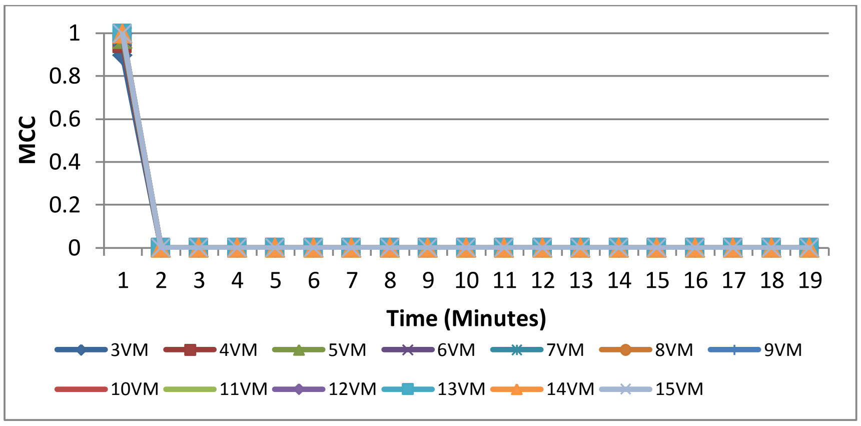


Figure 4.27: Detailed analysis of Case 7

Figure 4.27 shows similar results as to Figure 4.24 (case 4). Although the number of targetVMs increases from 50% to 99% in case 4 to case 7 and number of stdVMs decrease from 49% to 0%, the MCC values are similar. This is because of the following reasons.

1 A larger group of targetVMs gives a negative VMT value in case 7. This decreases the CredV of botVM to a negative value. Although this CredV is less than the corresponding CredV in case 4, but as both CredV values are negative, the MCC value is similar. Similarly, targetVM in case 7 has less CredV than case, but as both CredV are positive, the MCC value is same.

2 The S-VMD can detect 100% sets within the first 1 minute.

- *Case 8 (Scenario-C & SE2): With 1% botVMs, 99% targetVM, 0% standardVM and 50% peerVM*

From Figure 4.28 (Case 8) shows similar results as to Figure 4.25 (case 5). The differences we can observe are.

1 The duration after which the MCC values stabilize at zero differs in the sets with 14 and 15 number of VMs. In case 8, they stabilize after 4 minutes, whereas in

case 5 they stabilize after 3 minutes. This is because there are no stdVMs in case 8, propagation of DEofVMT is slower.

2 The S-VMD can detect 100% sets within the 4 minutes.

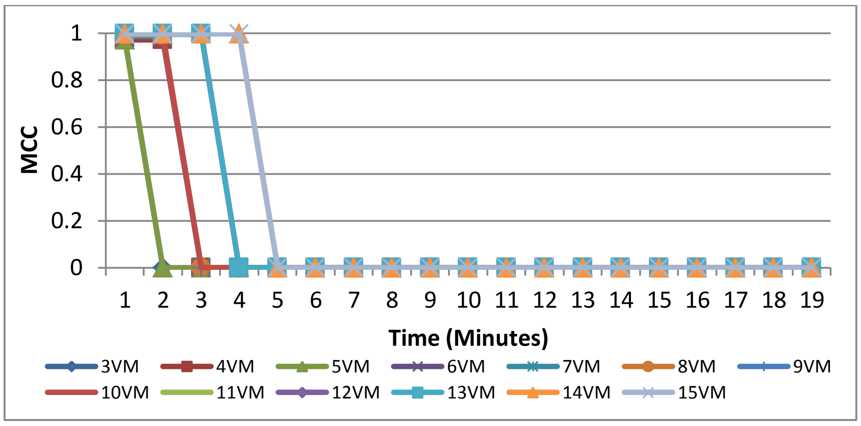


Figure 4.28: Detailed analysis of Case 8

- *Case 9 (Scenario-C & SE3): With 1% botVMs, 99% targetVM, 0% standardVM and 100% peerVM*

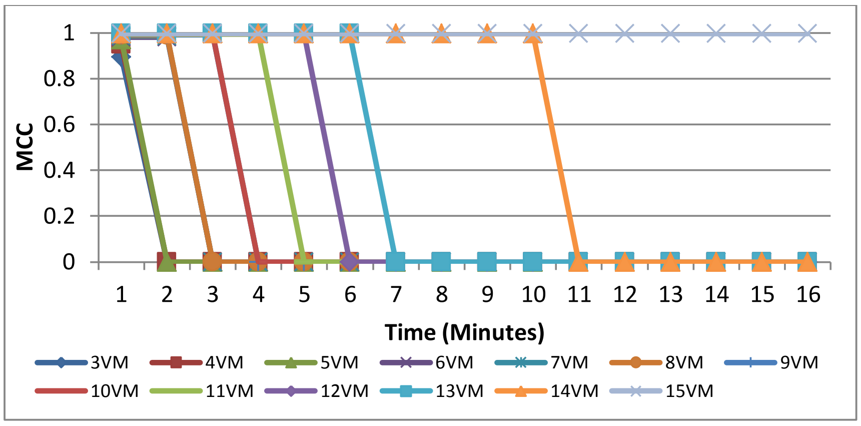


Figure 4.29: Detailed analysis of Case 9

From Figure 4.29 (Case 9) shows similar results as to Figure 4.26 (case 6).The difference is that the number of VMs from which the MCC value stabilizes closer to one is higher (15VMs) than Case 6 (7 VMs). This is because of high number of peerVMs and high (99%) targetVMs, the negative VMT value given to botVM have an indirect effect on all the targetVMs, thus reducing their CredV. As the number of VMs increase, the positive effect of VMT value they give to each other (as they are peerVMs), overcomes the negative VMT value effect and the MCC value stabilizes near one.

Scenario D: With 50% botVMs, 1% targetVM and 49% standardVM: It consists of Cases 10, 11 and 12.

- *Case 10 (Scenario-D & SE1): With 50% botVMs, 1% targetVM, 49% standardVM and 1% peerVM*

  Case 10 gives similar results to Case 4 (Figure 4.24), wherein we observe that the MCC value is high for a minute and then stabilizes at zero. The reasons behind are (i) botVMs receive negative VMT from targetVMs (1%) and they receive positive VMT from peerVMs (1%). Both these reasons combined with Equation (4.13) results a negative MCC value for botVMs. On the other hand, the targetVM receives positive VMT from peerVM (1%), thus increases its CredV.
  With an increase in time, DEofNVMT propagates and with larger number of botVMs (50%) the CredV of targetVM falls to a negative value. This results in a zero MCC value.
  Case 4 has low (1%) botVM and high targetVM (50%) as compared to Case 10, which has high (50%) botVM and low (1%) targetVM, they show similar results. The reason behind this is discussed below.
  As shown in Figure 4.30, Case 4 shows many to one relationship, where a set of targetVMs give negative VMT to a botVM, which shows a drop in the CredV of botVM to a negative value. This gives an initial high MCC value. However, with time DEofNVMT propagates and it reduces the CredV of targetVMs. On the other hand, case 10 shows one to many relationships where a targetVM gives negative to set of botVMs, which reduces their CredV to a negative value. This gives an initial high MCC value. However, with time the DEofNVMT propagates and it reduces the CredV of targetVMs to a negative value. Therefore, both cases show similar results.

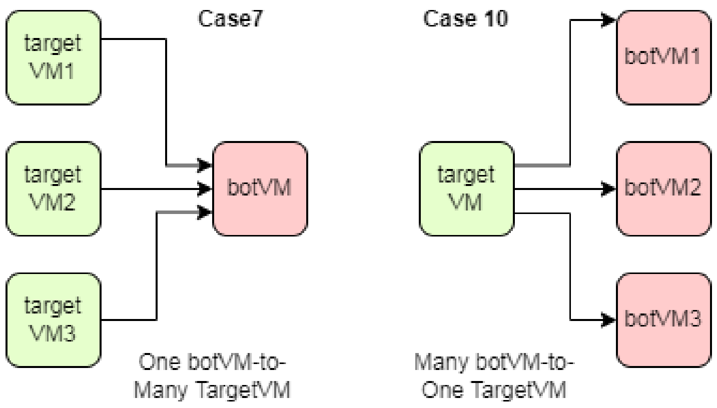


Figure 4.30: BotVM & TargetVM Relationship in Case 7 and Case 10

- *Case 11 (Scenario-D & SE2): With 50% botVMs, 1% targetVM, 49% standardVM and 50% peerVM*

Case 11 shows similar results to Case 10. The MCC value is high for a minute and then stabilizes at zero. Although we observe that the CredV of botVM is lower in case 11 compared to case 10 and the CredV of targetVM is greater in case 11 compared to case 10, still the MCC values are similar. This is because with increase in the number of peerVMs, the DEofVMT propagates to a larger section. At the same time, the difference in values is not large enough to be reflected in the final results, i.e. MCC values.

- *Case 12 (Scenario-D & SE3): With 50% botVMs, 1% targetVM, 49% standardVM and 100% peerVM*

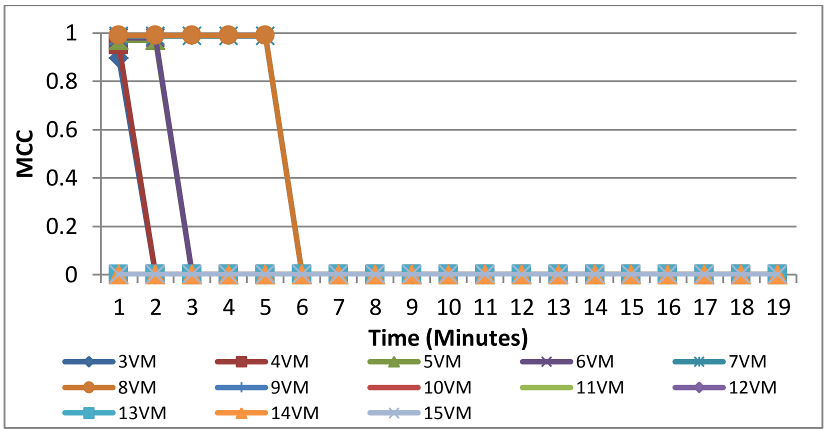


Figure 4.31: Detailed analysis of Case 12

Figure 4.31 gives the MCC value of case 12. From Figure 4.31, we can observe that case 12 has initial high MCC value for set of VMs ranging from 3 to 8. However, in sets with greater than 8 VMs, the MCC values is stable at zero. Initially a botVM receives negative VMT from a targetVM and positive VMT from peerVMs (50%). With sets less than 9 VMs, the CredV of botVM is negative as DEofNVMT overpowers DEofPVMT. However, with sets greater than 9 VMs, the CredV of botVM is positive as DEofPVMT overpowers DEofNVMT. On the other hand, the targetVM receives positive VMT from peerVM (100%), thus a positive CredV. A negative CredV of targetVM (for a maximum of 5 minutes) gives a positive MCC value and a positive CredV of targetVM (after 5 minutes) gives a zero MCC value.

Scenario E: With 50% botVMs, 50% targetVM and 0% standardVM: It includes cases 13, 14 and 15. Cases in this scenario are implemented with even numbers of VMs, i.e. 4, 6, 8, 10 and 12. This is because the aim of these cases to check the performance for equal number of botVMs and targetVMs.

- *Case 10 (Scenario-E & SE1): With 50% botVMs, 50% targetVM, 0% standardVM and 1% peerVM*

Cases 13 shows similar results as case 10. The MCC value is initially high for a minute and then stabilizes at zero. Initially the botVMs receive negative VMT from targetVMs (50%) and a positive VMT from peerVMs (10%). As the percentage of targetVMs is more than the number of peerVMs, the CredV of botVM decreases to a negative value. On the other hand, targetVMs receive positive VMT from peerVMs (10%), so it has a positive CredV. Finally, a negative CredV of botVM and a positive CredV of targetVM gives a positive MCC. With time, the DEofVMT propagates and, due to Equation (4.13), the CredV of targetVM decreases to a negative value. This stabilizes the result at zero MCC.
Cases 14 and 15 give similar results as case 13. This is because with an increase in number of peerVM, a peerVM could be picked from group of targetVM or botVM (as there are no stdVMs). As targetVMs are already giving negative VMT to botVM, so it doesn't act as its peerVM. In addition, botVMs have negative CredV initially, so a positive VMT from a co-botVM results in a negative VMT value. This results in zero MCC and doesn't change the results as compared to case 13.

Scenario F: With 99% botVMs, 1% targetVM and 0% standardVM: This includes scenarios 16, 17 and 18.

- *Case 16 (Scenario-F & SE1): With 99% botVMs, 1% targetVM, 0% standardVM and 1% peerVM*

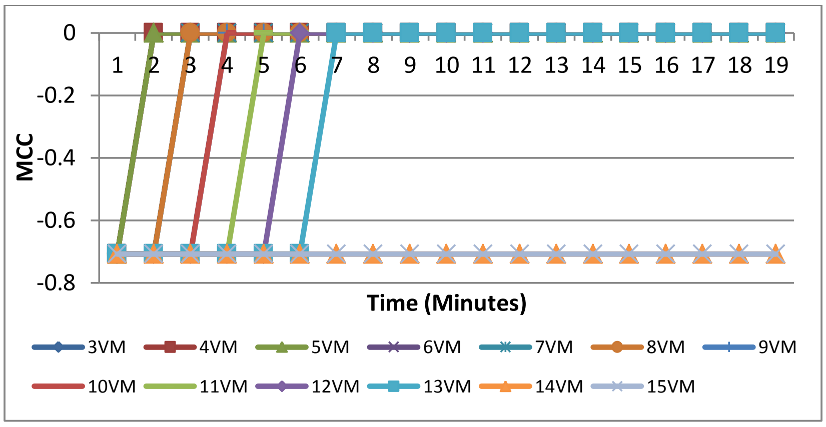


Figure 4.32: Detailed analysis of Case 16

Figure 4.32 gives MCC value for case 16. From the figure we can observe that the MCC value is negative for a maximum duration of 10 minutes and then stabilizes at zero. This negative VMT value stabilizes to zero till 14 VMs. For set of VM with greater than 14 VMs, the MCC value stabilizes at a negative MCC value. This negative MCC value is due to the high (99%) number of botVMs as compared to the number of targetVM (1%). This ensures that targetVM experiences indirect-DEofNVMT and its CredV is negative.

With a smaller number of VMs, this DEofNVMT is overcome by DEofPVMT with time and the MCC value stabilizes at zero. However, with larger group of VMs (>14), the DEofNVMT could not be overpowered within 19 minutes.

- *Case 17 (Scenario-F & SE2): With 99% botVMs, 1% targetVM, 0% standardVM and 50% peerVM*

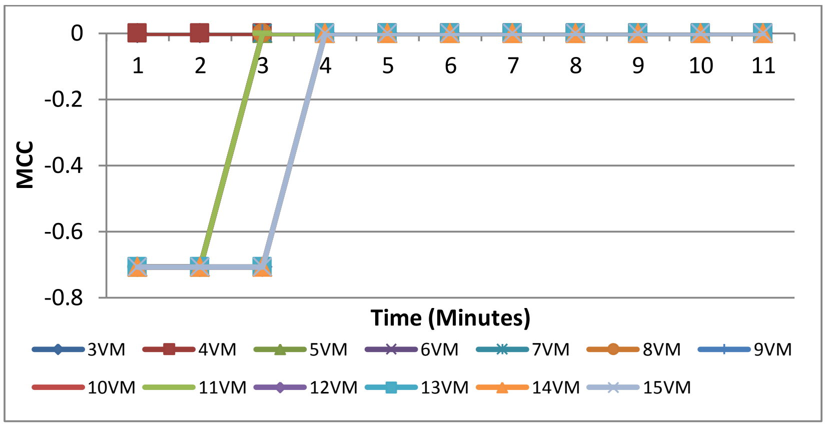


Figure 4.33: Detailed analysis of Case 17

Figure 4.33 shows the MCC value for case 17. From the figure we can observe that it gives similar results as case 15, however in the set of VMs with greater than 13 VMs, the MCC value stabilizes (at zero) quickly compared to case 16. This is because with increase in the number of peerVMs, the DEofVMT propagates to a wider range of VMs and thus affects the results faster.

- *Case 18 (Scenario-F & SE2): With 99% botVMs, 1% targetVM, 0% standardVM and 100% peerVM*

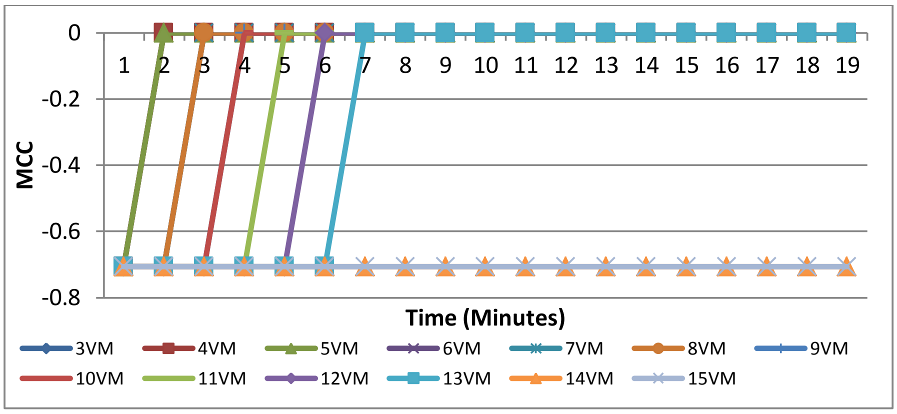


Figure 4.34: Detailed analysis of Case 18

Figure 4.34 shows the MCC value for case 18. From the figure we can observe that it gives similar results to case 14. In this case, 100% peerVMs means that VMs give positive VMT to each other, however as 1% of them is targetVM, so 99% botVMs give positive VMT to each other. At the same time, they experience indirect-DEofNVMT from each other. This results in a decrease in VMT value of botVMs. This should have been reflected in the MCC results, but this decrease is very small, so it is shows similar MCC values.

4.7 Security Analysis

In this section, we analyse the performance of Stage-1 components of bVMD Framework against the identified security attacks. These Stage-1 components can be compromised and can attack the system from within. These attacks are classified as internal attacks (also called insider attacks). These insider attacks can either be done from a compromised VM or a compromised S-VMD. Attacks initiated by a compromised VM are classified into three categories[134][135].

1 Bad Mouthing Attack: The botVM aims to degrade the credibility of benign peerVMs. To do this, it incorrectly rates its benign peerVMs with negative VMT value.

2 Ballot stuffing attack: The botVM aims to increase the credibility of benign botVMs. To do this, it incorrectly rates its bot peerVMs with positive VMT value.

3 Combined attack: The botVM aims to perform bad mouthing and ballot stuffing attack simultaneously. For this it incorrectly rates its benign peerVMs with negative VMT value and bot peerVMs with positive VMT value.

To study the effect of these attacks on the performance of bVMD Framework, we implement these attacks under all 18 cases. The results are discussed below.

4.7.1 Bad Mouthing (BM) Attack

The performance of all the 18 cases under bad mouthing attack is discussed below.

1 Case 1 :1% botVMs, 1% targetVM and 98% standardVM and 10% peerVM
Figure 4.35 shows the results of Badmouthing attack on case 1. From the results we observe that the MCC value is initially high and then stabilizes at zero within the first minutes. BotVM receives negative VMT from targetVM(1%) and positive VMT from peerVM (1%). Because of Equation (4.13), the CredV of botVM is negative. On the other hand, targetVM receives negative VMT from botVM (1%) and positive VMT from peerVM (1%). Therefore,

its CredV is negative. These values of CredV of botVM and targetVM results in a zero MCC value.

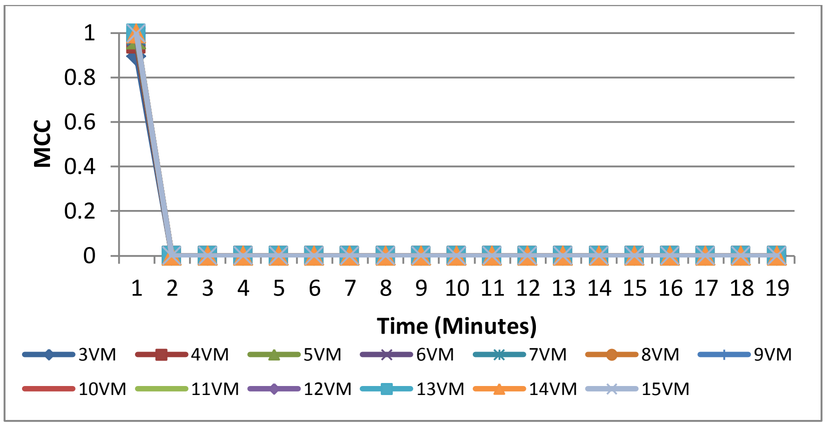


Figure 4.35: Bad mouthing attack Case 1

2 Case 2: 1% botVMs, 1% targetVM and 98% standardVM and 50% peerVM

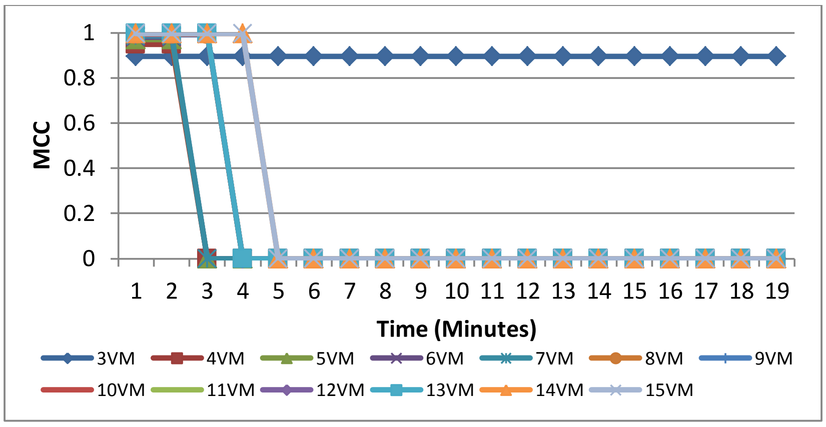


Figure 4.36: Bad mouthing attack Case 2

From the results in Figure 4.36, we observe that for sets with greater than 3 VMs, the MCC value is high before stabilizing at zero. As compared to BMCase1, the number of peerVMs increased which means a larger group (as compared to Case 1) of peerVMs giving positive VMT value to botVM and/or targetVM. BotVM receives negative VMT from targetVM (1%) and positive from peerVM (50%) and similar as the with targetVM. Therefore, both their CredV is low because the VMT value of botVM is negative and that of targetVM is zero (till 10 decimal place), thus giving a positive MCC value. These zero values of a targetVM might be different with more than 10 decimal precision.

3 Case 3: 1% botVMs, 1% targetVM and 98% standardVM and 100% peerVM
Figure 4.37 gives MCC results of case 3 in a badmouthing attack. From the figure we can observe that the MCC value is zero as both the botVM and targetVM have same CredV. This is because they give negative VMT to each other and positive VMT to stdVMs. Also, they receive positive VMT from stdVMs. As they have the same experience, their CredV is always same and thus the MCC value is zero.

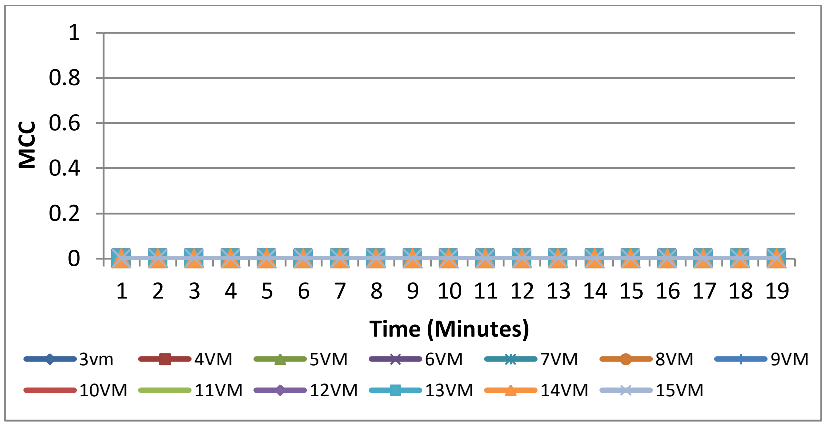


Figure 4.37: Bad mouthing attack Case 3

4 Case 4: 1% botVMs, 50% targetVM and 49% standardVM and 1% peerVM

This case gives similar results to Figure 4.37, i.e. a zero MCC value. BotVMs receive negative VMT from targetVM (50%) and positive VMT from peerVM (1%). Therefore, its CredV is negative. On the other hand, targetVM receives negative VMT from botVM(1%) and positive from 1% peerVM. Due to Equation (4.13), its CredV is negative. This results in zero MCC.

5 Case 5: 1% botVMs, 50% targetVM and 49% standardVM and 50% peerVM

Figure 4.38 shows the MCC value of Case 5 with badmouthing attack. We can observe that with increase in number of peerVMs as compared to Case 5, we can observe that the MCC value fluctuates before stabilizing at zero.

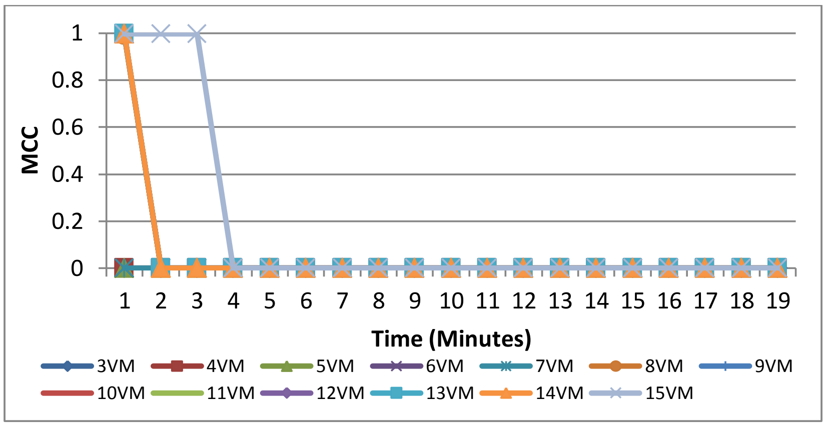


Figure 4.38: Bad mouthing attack Case 5

From Figure 4.38 we can make the following observations.

- The sets with VMs ranging from 3 to 7 show similar results as case 4. This is because with smaller number of VMs, the difference in peerVMs (50% and 100%) is not big enough to give visible results on the MCC value.
- For set with VMs ranging from 8 to 15, the botVM receives VMT from targetVM (50%) and positive VMT from peerVM (50%). Due to Equation (4.13), its CredV is negative. On the other hand, the targetVM receives negative VMT from botVM (1%) and positive VMT from stdVM (49%). As percentage of botVM is less than that of stdVM, the CredV of targetVM is positive.
- With time, the DEofVMT propagates and reduces the CredV of targetVM to a negative value and thus the MCC value decreases to zero. However, in set of 15 VMs, the time duration for CredV to stabilize at zero is higher. This is because the DEofNVMT takes more time to show visible effects on CredV of targetVM.

6 Case 6: 1% botVMs, 50% targetVM and 49% standardVM and 100% peerVM

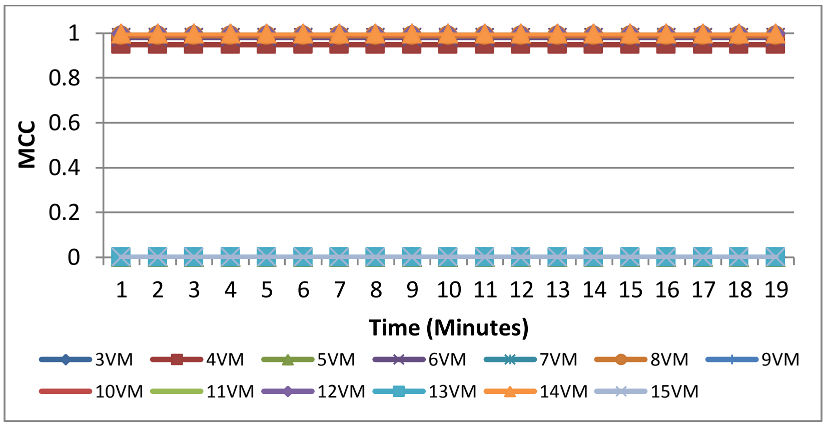


Figure 4.39: Bad mouthing attack Case 6

Figure 4.39 shows the MCC value of case 6 under badmouthing attack. From the results we observe that for sets with an even number of VMs, the MCC value is high and for sets with an odd number of VMs, the MCC value is low. This is because of the difference in the number of stdVMs. In even number of VMs, the number of targetVMs is greater than the number of stdVMs and in odd number of VMs, the number of targetVMs. The additional targetVM (in sets with even number) introduces additional negative VMT and as it propagates, the MCC value is high.

7 Case 7: 1% botVMs, 99% targetVM and 0% standardVM and 1% peerVM

Case 7 shows similar results to Figure 4.37. In this case, botVM receives negative VMT from targetVM, so it has a negative CredV. On the other hand, targetVM receives negative VMT from botVM and positive VMT from peerVM. Due to Equation (4.13), the CredV of targetVM is negative. Due to negative CredV of botVM and targetVM, the MCC value is zero.

8 Case 8: 1% botVMs, 99% targetVM and 0% standardVM and 50% peerVM

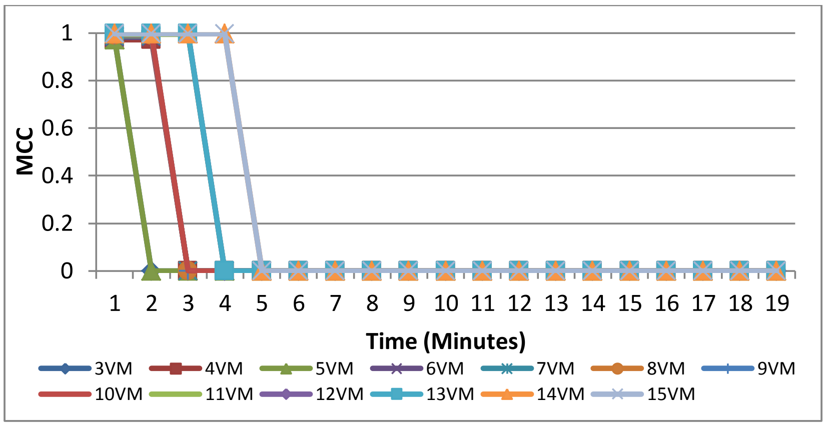


Figure 4.40: Bad mouthing attack Case 8

Figure 4.40 shows the MCC value for Case 8 under badmouthing attack. From the results we observe that the results are similar when there was no attack. This is because of smaller number of botVMs, therefore the effect of badmouthing, is not enough so that it creates a visible effect on the results (MCC value).

9 Case 9: 1% botVMs, 99% targetVM and 0% standardVM and 90% peerVM

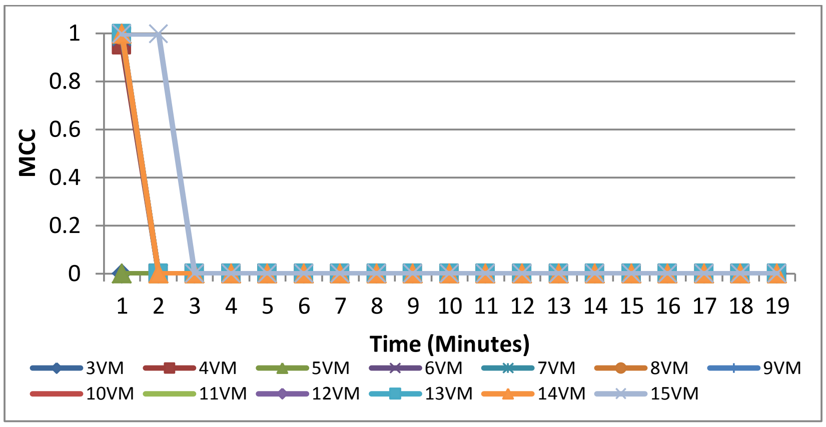


Figure 4.41: Bad mouthing attack Case 9

Figure 4.41 shows the MCC value for case 9 under badmouthing attack. From the results we observe that it gives similar results as case 8. However, in case 9, the MCC value reduces from a high value within 2 minutes, whereas it takes 4 minutes in case 8. This is because of 100% peerVMs due to which the indirect-DEofNVMT propagates among the targetVMs. This reduces the MCC value faster compared to case 8.

10 Case 10: 50% botVMs, 1% targetVM and 49% standardVM and 1% peerVM

Case 10 shows similar results to Figure 4.37. In this case, the botVM receives negative VMT from targetVM (1%) and positive VMT from peerVM (1%). Due to Equation (4.13), its CredV is negative. On the other hand, targetVM receives negative VMT from botVM (50%) and positive VMT from peerVM (1%), thus its CredV is negative. Due to a negative CredV of targetVM and botVM, the MCC value is zero.

11 Case 11: 50% botVMs, 1% targetVM and 49% standardVM and 50% peerVM

Case 11 shows similar results to case 10. As compared to case 10, the number of peerVMs increase in case 11. Due to this increase, the botVM receives positive VMT from a larger group of peerVMs (50%). However due to Equation (4.13), the CredV of botVM is still negative. On the other hand, the targetVM also receives positive VMT from a larger group of peerVMs, therefore its CredV is also negative. Due to negative CredV of botVM and targetVM, the MCC value is zero.

12 Case 12: 50% botVMs, 1% targetVM and 49% standardVM and 100% peerVM

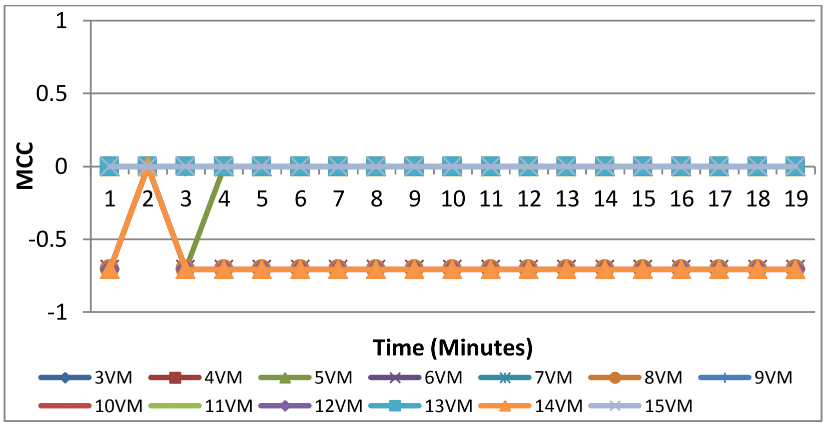


Figure 4.42: Bad mouthing attack Case 12

Figure 4.42 shows the results of MCC value for case 12 under a badmouthing attack. In this case, we observe that set with an even number of VMs, the MCC value stabilizes at a negative value, whereas, with an odd number of VMs it stabilizes at a positive value. This is because, with an even number of VMs, the number of botVMs is more than stdVMs, therefore the DEofNVMT overpowers DEofPVMT on targetVM and the MCC value reduces to a negative value. In sets with an odd number of VMs, due to the additional stdVM, the DEofPVMT overpowers and the MCC value stabilizes at zero

13 Case 13: 50% botVMs, 50% targetVM and 0% standardVM and 1% peerVM

Cases 13-15 under badmouthing attack have similar results Figure 4.37. In these cases, the botVM receives negative VMT from targetVM and positive VMT from peerVM. As there are no stdVM, the CredV of botVM is negative. On the other hand, the targetVM receives negative VMT from botVM and positive VMT from peerVM. Again, due to no stdVMs, the CredV of targetVM is negative. A negative CredV of targetVM and botVM gives a zero MCC value.

14 Case 16: 99% botVMs, 1% targetVM and 0% standardVM and 1% peerVM

Case 16 under badmouthing attack gives similar MCC results Figure 4.42. In this case, the botVM receive negative VMT form targetVM (1%) and positive VMT from peerVM (1%). Due to Equation (4.13), CredV of botVM is negative. On the other hand, targetVM receives a negative VMT from botVM (99%) and a positive VMT from peerVM (1%). So, its CredV is negative. As both the botVM and targetVM has negative CredV, the MCC value is zero.

15 Case 17: 99% botVMs, 1% targetVM and 0% standardVM and 50% peerVM

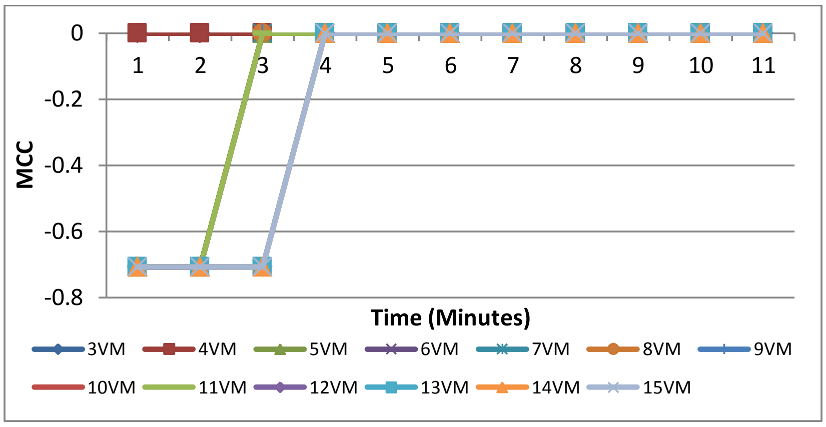


Figure 4.43: Bad mouthing attack Case 17

Figure 4.43 shows the MCC results for case 17 under badmouthing attack. In this case, the botVM receive negative VMT form targetVM (1%) and positive VMT from peerVM (50%). So, the CredV of botVM is positive. On the other hand, targetVM receives a negative VMT from botVM (99%) and a positive VMT from peerVM (50%). So, its CredV is negative. As botVM has a positive VMT and targetVM has a negative VMT, the MCC value is negative. With time, due to DEofVMT, the CredV of botVM and targetVM reduces to a negative value and thus the MCC value stabilized at zero.

16 Case 18: 99% botVMs, 1% targetVM and 0% standardVM and 100% peerVM

Case 18 under badmouthing attack gives similar results to case 17 under badmouthing attack (Figure 4.43). Although in case 19, the botVMs rate each other, so the DEofNVMT propagates to wider range of VMs and thus affects the overall result (MCC value) quicker as compared to case 17BM. However, the CredV is calculated after a fixed duration, i.e. 1 minute in bVMD, so the results are similar.

4.7.2 Ballotstuffing (Bs) Attack

As discussed, a ballot stuffing attack aims to give false positive VMT value to peer-bot. Out of the 18 cases used for evaluation, cases 1-9 only have single botVM, therefore they are not considered in this evaluation. Similarly, the cases with 100% peerVMs, i.e. 12, 15 and 18 already give positive VMT to each other, so they are not considered in the evaluation. The performance of 8 cases, i.e. 10, 11, 13, 14, 16 and 17 under ballot stuffing attack is discussed below.

1 Case 10: With 50% botVMs, 1% targetVM, 49% standardVM and 1% peerVM It gives similar results as Figure 4.37, i.e. the MCC value is stable at zero. In this case, the botVM receives negative VMT from targetVM (1%), positive VMT from peerVM (1%) and positive

VMT from peer-botVMs. Therefore, the CredV of botVM is positive. On the other hand, targetVM receives positive VMT from peerVMs (1%), so its CredV is positive. As positive CredV of botVM and targetVM, the MCC value is zero.

2 Case 11: With 50% botVMs, 1% targetVM, 49% standardVM and 50% peerVM

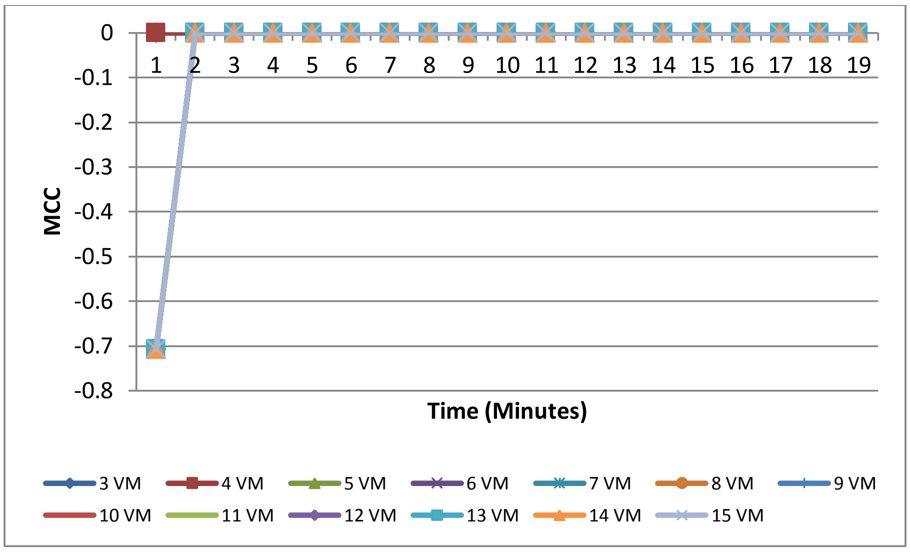


Figure 4.44: Ballotstuffing Attack on case 11

Figure 4.44 gives the MCC value of case 11 in ballot stuffing attack. In this case, initially the botVM receives negative VMT from targetVM, positive VMT from botVMs (50%) and positive VMT from its peerVMs (50%). This results in a positive CredV of botVM. On the other hand, targetVM receives negative VMT from the botVMs (50%) and positive VMT from peerVMs (50%). Due to Equation (4.13), the CredV of targetVM goes down to a negative value. A positive CredV of botVM and a negative CredV of targetVM results in a negative MCC. With time, the DEofNVMT propagates and the CredV of botVMs reduces to a negative value. This results in a zero MCC value.

3 Case 13: With 50% botVMs, 50% targetVM, 0% standardVM and 1% peerVM Case 13 under ballot stuffing attack shows similar results to case 12 under ballot stuffing attacks. In this case, the botVM receives negative VMT from targetVM (50%) and positive VMT from co-botVMs and positive VMT from peerVM (1%). This results in a negative CredV for botVM. On the other hand, targetVM receives positive VMT from peerVMs (1%) and negative VMT from botVMs (50%), so it has a negative CredV. A positive CredV of botVM and a negative CredV of targetVM results in a negative MCC. With time the DEofNVMT propagates and the CredV of botVMs decrease to a negative value. This results in a zero MCC value.

4 Case 14: With 50% botVMs, 50% targetVM, 0% standardVM and 50% peerVM Case 14 under ballot stuffing attack shows similar results as case 13 under ballot stuffing attacks.

This is because, as the number of peerVMs increase, the CredV of botVM and targetVM are not affected.

5 Case 16: With 99% botVMs, 1% targetVM, 0% standardVM and 1% peerVM

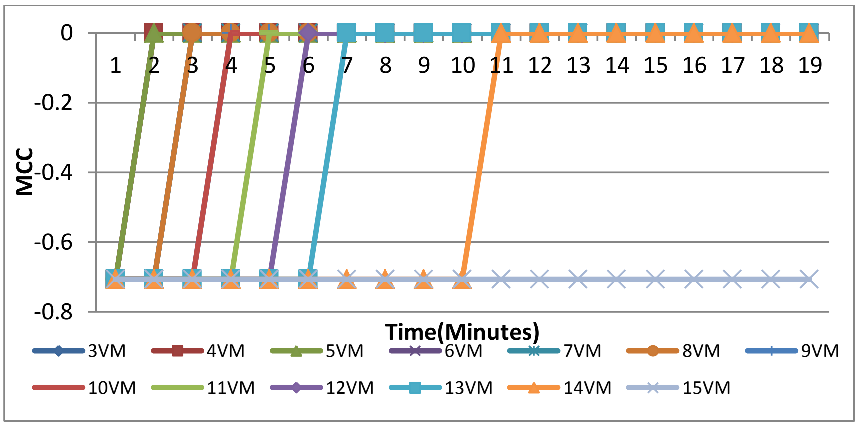


Figure 4.45: Ballotstuffing Attack on case 16

From the results in Figure 4.45, we observe that the MCC value is initially a negative value and then stabilizes at zero. The negative value is stable for duration of 11 minutes. The botVM receives negative VMT from targetVM (1%) and positive VMT from peerVM (1%) and co-botVMs (99%). This leads to a positive CredV of botVM. On the other hand, targetVM receives positive VMT value from peerVMs (1%) leading to a positive CredV. A positive CredV of botVM and targetVM results in zero MCC value. With time the DEofNVMT propagates and the CredV of targetVM and botVMs are affected, leading to zero MCC.

6 Case 17: With 99% botVMs, 1% targetVM, 0% standardVM and 50% peerVM

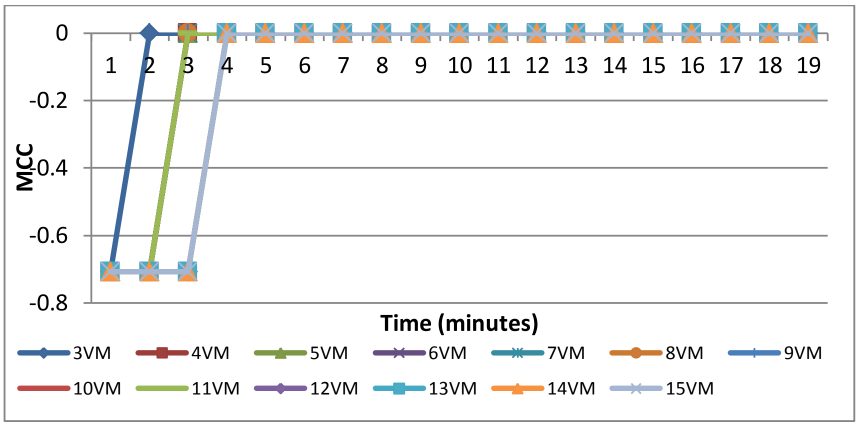


Figure 4.46: Ballotstuffing Attack on case 17

This case shows similar results as case 16. The botVM receives negative VMT from targetVM, positive VMT from co-botVMs and positive VMT from peerVM (50%). On the other hand, targetVM receives negative VMT from botVMs (99%) and positive VMT from peerVMs (50%). This results in a negative CredV. A positive CredV of botVM and a negative CredV of targetVM results in negative MCC value. With time the DEofNVMT propagates and the MCC stabilizes at zero.

4.7.3 Combined (Com) Attack: Badmouthing and Ballot-stuffing Attack

Similar to Attack 2, the performance of 8 cases, i.e. cases 10, 11, 13, 14, 16 and 17 are discussed below.

1 Case 10: With 50% botVMs, 1% targetVM, 49% standardVM and 1% peerVM
It has similar results to case 10 under ballot stuffing attack The CredV of botVM is similar to case 10 under ballot stuffing, however in this case the targetVM receives negative VMT from botVM (50%), positive VMT from peerVM (1%). So, the CredV of targetVM is negative. A positive CredV of botVM and negative CredV of targetVM gives a negative MCC value. With time, the MCC value stabilizes at zero.

2 Case 11: With 50% botVMs, 1% targetVM, 49% standardVM and 50% peerVM
In this case, botVM receives similar CredV (positive) as in case 11 under ballot stuffing attack. On the other hand, targetVM receives negative VMT from botVMs (50%) and positive VMT from peerVMs (50%). Due to Equation (4.13), the MCC value is negative. With time, the MCC value stabilizes at zero.

3 Case 13: With 50% botVMs, 50% targetVM, 0% standardVM and 1% peerVM In this case, botVM receives similar value as in Ballot stuffing case 13. On the other hand, targetVM receives positive VMT from targetVMs (1%) and negative VMT from botVMs. This results in negative VMT. Therefore, a negative CredV of botVM and targetVM gives zero MCC.

4 Case 14: With 50% botVMs, 50% targetVM, 0% standardVM and 50% peerVM In this case, botVM receives similar CredV as Ballot stuffing case 14. On the other hand, The botVMs receive negative VMT from 50% targetVMs and positive VMT from 50% targetVM receives negative VMT from botVM (50%) and positive VMT from peerVM (50%), it also sets a negative CredV. Due to Equation (4.13), the CredV of targetVM is negative. A negative CredV of botVM and targetVM results in a zero MCC value.

5 Case 16: With 99% botVMs, 1% targetVM, 0% standardVM and 1% peerVM

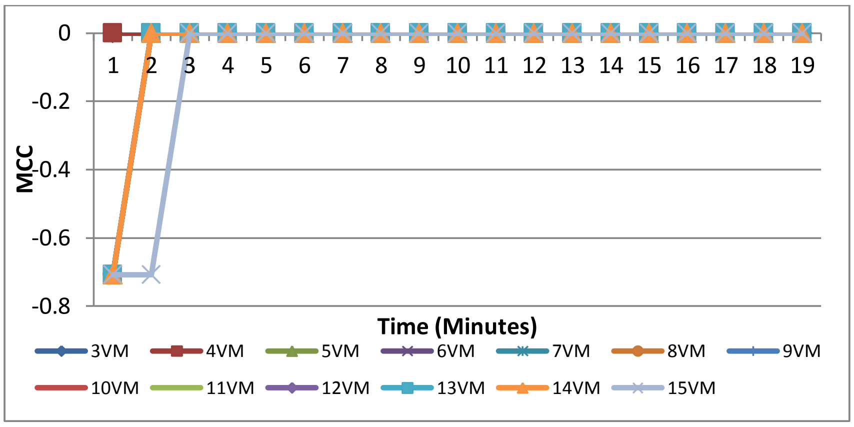


Figure 4.47: Combined Attack on case 16

In this case (Figure 4.47), botVM receives similar CredV as case 16 under ballot stuffing. On the other hand, targetVM receives negative VMT from botVM (99%). Therefore, its CredV is negative. A positive CredV of botVM and a negative CredV of targetVM, MCC value is negative. As the negative VMT propagates with time, the CredV of botVM drops to a negative value and MCC stabilizes at zero.

6 Case 17: With 99% botVMs, 1% targetVM, 0% standardVM and 50% peerVM
In this case, botVM has similar CredV as case 17 under ballot stuffing attack. On the other hand, targetVM receives positive VMT from peerVM (50%) and negative VMT from botVMs (99%). This results in negative CredV. A positive CredV of botVM and a negative CredV of targetVM gives negative MCC value. With time this value stabilizes at zero. In addition, we can observe that the MCC value stabilizes faster than case 16 under ballot stuffing. This is because of increase of number of botVMs giving negative VMT to targetVM (due to badmouthing).

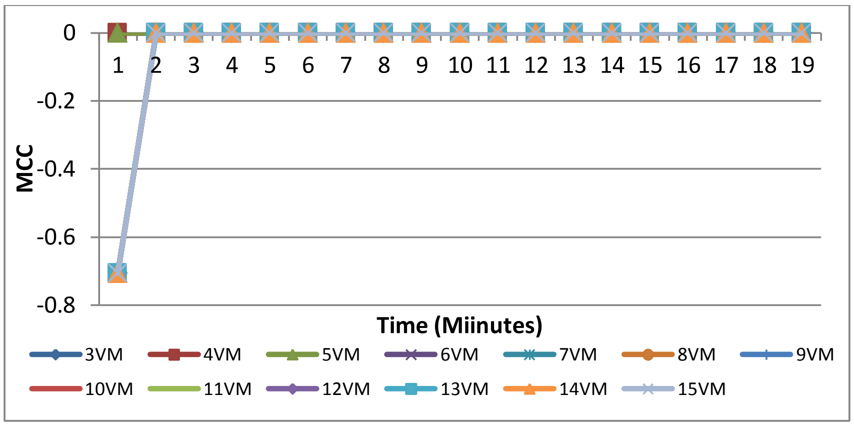


Figure 4.48: Combined Attack on case 17

4.8 Key Findings

The results from our investigations on stage-1 components of bVMD Framework are as follows.

1 **No Attack Situation**

(a) Changes in number of peerVM: This describes the effects of the changes in the number of peerVMs on the performance of bVMD Framework.

- Scenario A: Comparison among Cases 1, 2 and 3 (1% botVM, 1% targetVM and 98% stdVM)
  From figures 4.20, 2.23 and 4.24 we can observe the following.
  - The fluctuation in the MCC value decreases with increase in the number of peerVMs. As larger number of VMs are involved in the calculations, more stdVMs give positive VMT as peerVM and the system is more stable.
  - Case with maximum peerVMs, i.e. case 3 shows the least false detection. This is because a greater number of stdVMs give positive VMT value and thus the MCC value increases.

  Case 2, there are 13.36% positive MCC values and 14.57% negative MCC values, however in case 3, there is 9.72% positive MCC value. Case 3 has an average MCC of +9.72% as compared to -1.72% with case 2.
- Scenario B: Comparison among Case 4, 5 and 6 (1% botVM, 50% targetVM and 49 % stdVM) We can observe that the MCC values in all these cases are either zero or positive. This is because of a larger set of targetVMs as compared to botVMs. In all these cases, the MCC values are initially positive and then stabilize at zero. However, the time taken by MCC value to stabilize at zero differs. The time increases with an increase in the number of peerVMs, i.e. in case 1 it stabilizes after 1 minute, in case 2 and 3 it stabilizes after 3 minutes. However, there are certain sets of VMs (greater than 6 VM) in which the values stabilize at high MCC. The reason is that with increase in the number of peerVMs, the DEofPVMT is spread across a larger group of VMs, and the DEofNVMT takes time to affect the CredV of targetVM. Here we say that DEofPVMT spreads faster than DEofNVMT as the number of VMs receiving positive VMT (i.e. targetVMs) are much more than the number of VMs receiving negative VMT (i.e. botVMs). The case 4, 5 and 6 have +5.26%, +10.53% and +72.87% average MCC value respectively.
- Scenario C: Comparison among cases 7, 8 and 9 (1% botVM, 99% targetVM and 0% stdVM) Similar to above set, with increase in peerVMs, the MCC value is positive for a longer duration. In addition, in case 9 the MCC value stabilizes at a high value for set of 15VMs. Here in case 7, 8 and 9 have +5.26%, +12.55% and +23.88% average MCC values respectively. Therefore case 8 shows best out of these.

- Scenario D: Comparison among case 10, 11 and 12 (50% botVM, 1% targetVM and 49% stdVM) We can observe that cases 10 and 11 show similar results, where the MCC value is initially high and then stabilize at zero within a minute. Case 12 stabilizes after a maximum of 5 minutes. This is because of DEofVMT propagates with greater number of peerVMs. Thus, it takes more time to stabilize.
  Here in cases 10, 11 and 12, there are +5.26%, +5.26% and +6.48% average MCC values. Therefore case 12 is the best out of these.
- Scenario E: Comparison among cases 13, 14 and 15 (50% botVM, 50% targetVM and 0% stdVM) Here the MCC value is same in all these cases, 5.26% positive MCC values.

(b) Changes in number of botVMs: In this section, we discuss the changes on the performance of bVMD Framework with change in the number of botVMs. The cases with 1%, 50% and then 99% botVMs are compared.

- 10% peerVMs (This involves cases 1, 10 and 16)
  - With an increase in botVMs, the average MCC value decreases from +46.15% in case 1 to +5.26% in case 10 and -23.89% MCC value.
  - This shows that with increase in number of botVMs, the MCC value decreases, thus the minimum number of botVMs is preferable.
- 50% peerVMs (This involves cases 2, 11 and 17)
  - With an increase in botVMs, average MCC value of case 2 has a -1.21% MCC value to +5.26% in case 11 and -23.89% MCC value.
  - This shows that with an increase in number of botVMs, the MCC value first increases and then decreases.
- 100% peerVMs (This involves cases 3, 12 and 18)
  - With an increase in number of botVMs, the number of MCC value decreases from +9.71% in case 3 to +6.48% in case 12 and -23.89 in case 18.
  - This shows with an increase in number of botVMs, the MCC value decreases, thus minimum number of botVMs is preferable.

(c) Changes in number of targetVMs: In this section, we discuss the changes on the performance of bVMD Framework with change in the number of targetVMs.

- 10% peerVMs (This involves cases 1, 4 and 7)
  - With an increase in number of targetVMs, the number of average MCC value decreases from +45.74% in case 1 to +5.26% in case 4 and case 7.
  - This shows that, with an increase in number of targetVMs, the MCC value decreases, thus minimum number of targetVM is preferable.
- 50% peerVMs (This involves cases 2, 11 and 17)
  - With an increase in number of targetVM, the case 2 has a -1.21 value to +10.53% in case 5 and +12.55% in case8 average MCC value
  - With an increase in number of targetVMs, the MCC value increases

- 100%peerVMs (This involves cases 3, 6 and 9)
  - With an increase in number of botVMs, the number of MCC value changes from +9.71% in case 3 to +72.87% in case 6 and +23.88% MCC value in case 9.
  - This shows that, with an increase in number of botVMs, the MCC value first increases and then decreases.

(d) Changes in number of stdVMs: In this section, we discuss the changes on the performance of bVMD Framework with change in the number of stdVMs. It includes cases with 0%(minimum) to 98%(maximum) stdVMs.

- 10% peerVMs (This involves cases 13 (minimum) and 1 (maximum)): With an increase in number of stdVMs, the number of MCC value increases from -5.26% in case 13 to net 45.74% in case 1.
- 50% peerVMs (This involves cases 14 (minimum) and 2 (maximum)): With an increase in number of stdVMs, the number of MCC value increases from -5.26% in case 13 to net -1.21% in case 1.
- 100% peerVMs (This involves cases 15 (minimum) and 3 (maximum)): With an increase in number of stdVMs, the number of MCC value increases from -5.26% in case 13 to net 9.71% in case 1.

This shows that a maximum number of stdVMs gives the best result.

From the above analysis, we made four findings in this situation.

- Low number of peerVMs (i.e. 1%) gives low results. Here low results refer to both botVM and targetVM either having positive VMT values or a negative VMT values, thus a zero MCC value. In such a case, these botVMs cannot be detected.
- Medium number of peerVMs (i.e. 50%) gives high results with high number of targetVMs.
- High number of peerVMs (i.e. 100%) gives best results in terms of detection rate, however there are some false positive detections as well.

2 **Badmouthing attack**: We made the following finding in this attack situation.

(a) The cases where there is a zero MCC value have either of the following situations.

- Low number of peerVMs (i.e. 1%) (Cases 4, 7, 10, 13, 16)
- Same number of targetVM and botVM with no stdVMs (13, 15, 15)
- Greater number of botVMs as compared to targetVMs and a low or medium peerVMs (i.e. 1% or 50%) (cases 16, 17)

(b) The cases where there is a constant positive/negative MCC value have either of the following situations.

- Positive value: High number of targetVMs and presence of stdVM. It works well with 100% peerVMs (case 6)

- Negative value: High number of botVMs and presence of stdVM. It works well with 100% peerVMs. (case 12)

(c) The cases where there is a positive or negative MCC value for some time and then settle to zero have either of the following situations.

- Positive to zero: same number of botVMs and targetVMs, and presence of std-VMs. This includes the medium or high number of peerVMs (i.e. 50% or 100%) (case 2, 3, 9)
- Negative to zero: Highest number of botVMs with no stdVMs and 100% peer-choice (case 18)
- Positive to negative to zero: High number of targetVMs (50%) with presence of stdVMs and 50% peerVMs. (case 5)

3 **Ballotstuffing attack:** : We made the following findings in this attack situation.

(a) The cases where there is zero MCC value have either of the following situations.

- Low number of peerVMs (i.e. 1%) (Cases 10, 13) and number of botVMs less than or equal to number of targetVMs
- Greater number of botVMs as compared to targetVMs and a low or medium peerVMs (i.e. 1% or 50%) (cases 10, 11)

(b) The cases where there is a positive or negative MCC value for some time and then settle to zero have either of the following situations.

- Positive to zero: None
- Negative to zero: Highest number of botVMs with no stdVMs and 10% or 50% peerchoice (case 16, 17) or equal number of botVMs and targetVMs with no stdVMs and 50% peerVMs.

4 **Combined attack**: We made the following findings in this attack situation.

(a) The cases with no results (i.e. zero MCC) have either of the following situations:

- Low/medium number of peerVMs (i.e. 1% or 50%) (Cases 10, 13, 11, 14) and number of botVMs less than or equal to number of targetVMs.

(b) The cases where there is a positive or negative MCC value for some time and then settle to zero have either of the following situations

- Positive to zero: None
- Negative to zero: Highest number of botVMs with no stdVMs and 10% or 50% peerchoice (case 16, 17).

4.9 Chapter Summary

This chapter has presented the design and simulation study of a novel stage-1 components of bVMD Framework. The components, VMWatcher and S-VMD aims to identify set of suspected malicious VMs based on their network activity while minimizing the use of additional dedicated components. We use simulation to evaluate the effectiveness of the stage-1 components in detecting suspected malicious VMs.

The results show that in the case of badmouthing attack, firstly, the scenarios with either high numbers of botVMs or targetVMs and 100% peerVMs were classified correctly. Secondly, similar scenarios with average (50%) numbers of peerVMs and scenarios with equal numbers of botVMs and targetVMs were classified correctly for initial couple of minutes. Thirdly, scenarios with lowest (1%) number of peerVMs, same number of targetVMs and botVMs with no stdVMs and high (1% or 40%) botVMs and medium (50%) or low (1%) peerVMs were not classified at all.

In the case of ballotstuffing attack, firstly, scenarios with high (99%) botVMs or equal number of botVMs and peerVMs, no stdVMs and low (10%) or medium (50%) peerVMs were classified correctly, however they were identified incorrectly. Secondly, scenarios with botVMs less than or equal to targetVMs and low (1%) peerVMs were not identified at all.

In the case of combined attack, only scenarios with high (99%) botVMs, no stdVMs and low (10%) or medium (50%) peerVMs were identified incorrectly, whereas all other scenarios were not identified.

Finally, in the case of no attack, we observe that with equal numbers of botVM and targetVM, the MCC value stabilizes with increase in number of peerVMs. In scenarios with high number of targetVM as compared to number of botVMs, the scenarios were correctly identified for initial couple of minutes and then cannot be identified. These initial minutes depend on the number of peerVMs and the proportion of number of targetVM and botVM. In scenarios with high number of botVMs than the number of targetVMs, the scenarios are correctly identified for initial couple of minutes and then cannot be identified. However, the MCC values obtained in these cases are less than the MCC values in cases with high number of targetVMs as compared to the number of botVMs.

Chapter 5

bVMD Stage-2: Bot-VMD and FVM

5.1 Chapter Introduction

This chapter presents the design and evaluation of two components used in Stage-2 of the bVMD architecture, namely Bot-VMD and FVM. These components use a host-based intrusion detection system which follows Virtual Memory Introspection (VMI) technique to analyse the physical memory of a VM for any malicious activity. Forensic VMs (FVMs) are proposed in the literature to implement VMI technique. Our solution proposed a novel algorithm, Hybrid-FVM, to determine the number and type of FVMs with an aim of improving the effectiveness (in terms of time taken by FVMs). An experimental evaluation of the algorithm is carried out by comparing the time taken by FVMs to analyse the set of S-VMs and the associated overheads against the most relevant work.
The structure of this chapter is as follows. Section 5.2 presents the details of Virtual Memory Introspection technique. Section 5.3 discusses the related work in literature. Section 5.4 describes the proposed hybrid approach for estimating FVMs. Section 5.5 describes the details of bot-VMD component. Section 5.6 gives the details of the operations of bVMD stage-2 components. Section 5.7 describes the evaluation methodology and Section 5.8 presents the experimental results and discussions. Finally, Section 5.9 summarises the chapter.

5.2 Virtual Memory Introspection (VMI)

VMI is a technique proposed in 2003 [136] and explained in Figure 2 in [137]. It is defined as a virtualisation-based technique that enables an 'outsider' to monitor and analyse the state of a VM by observing its memory pages [138] [139]. This technique allows the detection solution to be placed out of the probable botVM to detect any malicious event or code executed at run time (at botVM's memory location). This isolation might result in lack of awareness of botVM about the detection solution, thus giving it an added advantage. VMI is used in Intrusion Detection Systems (IDS) for real time monitoring. The VMI technique is implemented using a dedicated component, called a Monitoring and Detection (M&D) component, which analyses and studies the physical memory of a VM. This M&D component can either reside on the hypervisor (M&D-on-hypervisor) [140] or on a local VM (M&A-on-VM) [136]. The M&A-on-hypervisor deployment means that there is a single M&D component to monitor all the VMs on the host that analyse their monitored data. If a hypervisor hosts a large number of VMs, this approach may overburden the hypervisor and it may also become a single-point-of-failure. The M&D-on-VM deployment can also be implemented using a centralised or a distributed approach. A centralised approach means that a dedicated VM hosts the component which monitors all the VMs on the same host. This might have similar problems as the M&D-on-hypervisor deployment due to a centralised structure. A distributed approach means that there is a number of dedicated VMs, each of which hosts a separate M&D component. Each of these dedicated VMs monitor and analyse the data from one or more VMs. This approach reduces the burden on the single VM, but it may increase the overheads in maintaining these additional VMs. This approach is proposed [141], where a special VM, called Forensic VM is used to detect malware activities in a set of VMs.

5.2.1 Forensic VM (FVM)

FVM is defined as a mini VM that can monitor other VMs to discover malware symptoms in real-time using the VMI technique. FVMs have the following features 1) they are small in size, so that they can be easily managed, 2) they work in read-only mode, so that unauthorised alteration of the monitored data can be prevented, and 3) they inspect one VM at a time, and can hop between VMs after inspecting a VM. FVMs work according to the following steps [141][87].

Step 1 **Transferring of VMI meta-data**: Before launching an FVM, they are given read-only access to the meta-data (like offset value, address, memory size) of a VM which is stored in a hypervisor.

Step 2 **Determining and utilising OS offsets**: The meta-data acquired from the hypervisor is used to determine the location of VM's kernel task structure. This task structure describes the applications recently initiated in the VM. A task structure of a particular process is used to determine:1) a pointer to a region that contains page tables that should be loaded when that process is running, and 2) a list of areas of the application's virtual address space that it is using. These two parameters, i.e. the page table and virtual address space, are then used to determine the system physical memory locations being used by that application process.

Step 3 **Converting known target guest kernel address into machine physical address**: The machine's physical address is calculated using the task structure (within the guest's virtual address). The physical page at that address is mapped into FVM for analysis.

Step 4 **Crawl through page tables**: After identifying the task structure of the guest OS, the page tables used by the process are located and the corresponding memory regions are tracked.

These FVMs can be classified into two categories, Symptom-specific FVMs (S-SF) and VM-Specific FVMs (V-SF).

1 **Symptom-Specific FVMs (S-SF)**
Scenarios using the 'Symptom-Specific' FVM approach are those in which one FVM is responsible to detect one symptom in all the VMs. In other words, a 1-to-1 ratio is maintained between the number of FVMs and the number of malware symptoms. This approach involves a scheduling algorithm, which defines the way in which an FVM has to hop from one VM to another VM. This method may be more scalable as the number of agents doesn't depend on the number of VMs. However, the hopping of FVMs among VMs introduces computational overheads and the overheads increase as the number of VMs increases. The mobility algorithm used for scheduling the hopping process generally use priority system. A VM with less priority might not be checked for a long duration, thus reducing the effectiveness of the solution. In addition, on-off attacks may be hard to detect with this approach.

2 **VM-Specific FVMs (V-SF)**
Scenarios using the 'VM-Specific' FVM approach are those in which one FVM is responsible for analysis of all the symptoms in a single VM. In other words, a 1-to-1 ratio is maintained between the number of FVMs and the number of VMs. This approach may be effective in terms of early detection compared to scenarios using the S-SF approach. This is because each VM is continuously monitored by its respective FVM. However, with this approach, the number of FVMs increases with the increase in the number of VMs, so, as the number of VMs increases, overheads incurred in managing and securing the FVMs will also increase.

5.2.2 Malware Symptoms

A malware symptom is an indicator of a presence of a malicious activity. They are based on the observation of a trend of component reusability by writers of malware which produces

symptoms (detectable traces of activities that facilitate a malicious activity) such as shutting-down anti-virus, changing registry key values, etc.
Definition: A malware symptom is an abstraction of an observable (via VMI) characteristic, which can be linked to malicious behaviour and appearance of a symptom indicates possible malicious behaviour [141].
Some examples of malware symptoms are given below [141][142].

Sym 1 **Shutting down the processes**: Some malware interrupts the normal working of important security processes like antivirus, etc. For example, botnet Conflicker C shuts down almost 20 processes as soon as it enters a system.

Sym 2 **Tampering with files**: Some malware might create fake files, delete or modify original files from the system directories. For example, botnets Tidserv and Pilleus malware does all these three file operations once they enter a system.

Sym 3 **Change registry keys**: Some malware changes the values in the hierarchy of directories associated to a registry key. For example, botnet Conflicker adds strings (audio, image, etc.) to the registry to obfuscate registry configuration changes.

Sym 4 **Modifying the time attributes of a file**: Some malware modifies the time attribute of a file so that it deceptively looks like the file exists since the launch of the operating system. For example, Zeus modifies the time of the creation of some of the malicious exe files to the time of the installation of the operating systems.

Sym 5 **Identifying suspicious snippets of code**: Use of crypto algorithms is very popular with malware writers. Snippets of program code containing known crypto algorithms can be a sign of malicious behaviour. For example, three variants of Conficker (A, B and C) make use of RC4, RSA, and MD-6 and keep updating the implementations.

5.3 Related Work

This section describes two pieces of related work, which use FVMs for detecting malware in a cloud environment.

1 The author [87] uses S-SF approach, where FVMs hop between VMs based on the mobility algorithms (Round-robin and Random algorithms). These FVMs communicate with each other and can notify each other about an identified symptom on a VM, thus enabling other FVMs to check their respective symptoms in the identified VM. The FVM communication is implemented using a shared message-board mechanism. The FVMs store information about the VMs they have identified, the time a VM was last visited and a list of symptoms it has checked. Mini-OS is used for implementation of these FVMs, thus making them computationally cheap to run and less prone to attacks (due to their small size). However, the continuous search for a symptom across multiple VMs might introduce additional computational and communicational overheads. The mobility algorithm used by FVMs might affect the total time taken for analysis. Scenarios where a VM was infected immediately after it was inspected by an FVM might have to wait for the next round of inspection before detection. The use of a black-board mechanism for communication among FVMs might be prone to attacks.

2 The author [81] uses V-SF FVMs for detection of botcloud attacks. They use an FVM to monitor all the symptoms of a VM thus minimising the requirement of communication among FVMs. However, as an FVM is supposed to examine all the symptoms in a VM, they might require a larger space than the S-SF approach, therefore they may be more expensive to run. In addition, as the number of VMs increases, the number of FVMs also increases, thus increasing the associated overheads and the attack surface.

5.4 A Hybrid Approach to FVM Estimation

One of the tasks of a Bot-VMD in bVMD framework is to estimate the number and type of FVMs required for forensic analysis of the identified S-VMs. This is done by using FVM estimation method. The method is based on two parameters 1) the number of S-VMs to be analysed and 2) the number of malware symptoms to be checked by each FVM. These parameters affect the time taken for analysing a S-VM.

Assuming that the number of VMs to be analysed is 'n', namely VM_1 through to VM_n, and there are a set of 's' symptoms to be analysed, namely Sym_1 through to Sym_s. Then the number of FVMs is estimated using the following steps.

1 If the number of VMs to be monitored is more than fvm-delta times the number of symptoms, i.e.

$$if(n > (fvm - delta \times s)) \tag{5.1}$$

is true, then Number of FVMs = s
Otherwise follow step 2.

2 If the number of VMs to be monitored is less than or equal to fvm-delta times the number of symptoms, i.e.

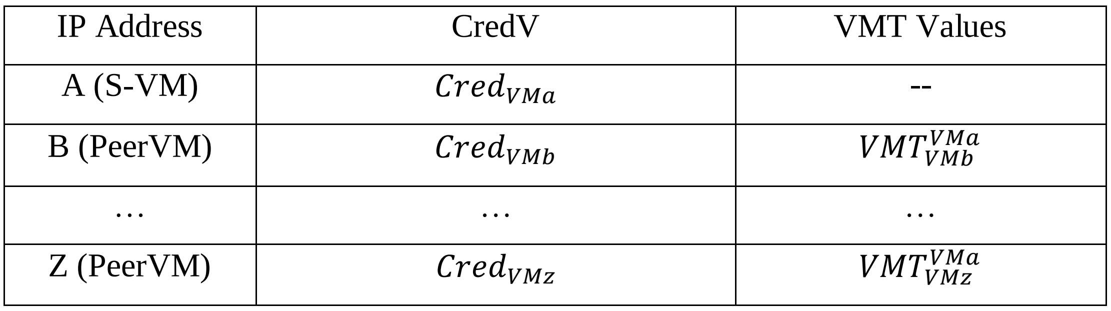
$$if(n \leq ((fvm - delta) \times s)) \tag{5.2}$$

is true, number of FVMs = n

5.5 Bot-VMD

A Bot-VMD is responsible for conducting the forensics analysis of S-VMs. This component is placed in each hypervisor and it carries out the following four tasks: 1) receive S-to-B packets from S-VMs, 2) determine the number of FVMs, 3) receive monitored data from FVMs, and 4) analyse the monitored data and take actions.

5.5.1 S-VM Value Acquisition



IP Address	CredV	VMT Values
A (S-VM)	$Cred_{VMa}$	--
B (PeerVM)	$Cred_{VMb}$	VMT_{VMb}^{VMa}
…	…	…
Z (PeerVM)	$Cred_{VMz}$	VMT_{VMz}^{VMa}

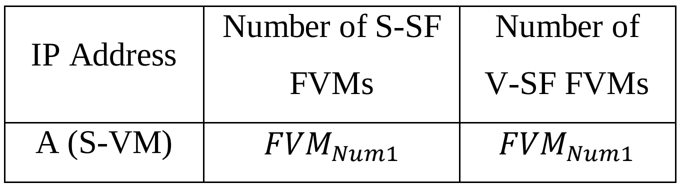
Figure 5.1: S-to-B Packet

Bot-VMD receives the S-to-B packet with the details (IP Address, CredV and VMT values) of S-VMs from S-VMD as shown in Figure 5.1 . The packet includes the IP Addresses of the S-VM (VM_a) along with its CredV ($Cred_{VM_a}$). It also includes the IP addresses of the VMs which has S-VMs as their peerVMs (VM_b to VM_z), the VMT values they provided ($VMT_b{}^a$ to $VMT_z{}^a$) and their CredV ($Cred_a$ to $Cred_z$).

There is a need to pass the information about the VMs which have S-VM as their peerVM so that bot-VMD can classify them into groups based on the forensic results of S-VM.

5.5.2 FVM Value Estimation

The S-VM values acquired by bot-VMD determine the sequence in which FVMs are estimated, created and dispatched. If a S-VM has high AL, then it has the highest priority and it is checked first. Otherwise the S-VMs are checked in the order in which their respective S-to-B packet is received. The number and type of FVMs are determined using FVM Estimation method described in Section 5.4. The type and number of FVMs are communicated to FVM Dispatch & Analysis method using Est_Packet given in Figure 5.2.



IP Address	Number of S-SF FVMs	Number of V-SF FVMs
A (S-VM)	FVM_{Num1}	FVM_{Num1}

Figure 5.2: Estimation Packet

This Est-Packet contains the IP address of the S-VM to be inspected, the number of S-SF FVMs required, and the number of V-SF FVMs required.

5.5.3 FVM Creation, Dispatch and Analysis (D&A)

Based on the Est-Packet received, the FVMs are created and then dispatched. The results of analysis are received from the FVMs. These results are stored in bot-VMT-T with a timestamp. This is because of the following case.

Case 1 Where a S-VM is identified as botVM and is being notified to the hypervisor and there is another request from a different S-VMD to analyse the same S-VM in the meantime. In this case, this stored data (with a timestamp) can be checked instead of analysing the VM again.

Cbse 2 Where a S-VM is identified as a benign VM, but there are repeated requests from different S-VMDs about the analysis of S-VMs. In this case, the stored data can be checked and used.

If all the symptoms are detected in the S-VM, then it is identified as botVM.

5.5.4 Actions Taken

The bot-VMD takes the following actions, based on the data received from FVMs.

1 If a VM is confirmed as a botVM, then the Bot-VMD notifies its hypervisor and in addition, it notifies its local S-VMD. This is to make sure that the S-VMD can take an immediate action, even before the hypervisor takes an action. In addition, the S-VMD which notified about the identified botVM is also informed.

2 If the VM is not identified as a botVM, then the following S-VMDs are notified:

(a) The S-VMD which requested the bot-VMD to analyse the S-VM: this is to ensure that the S-VMD knows that an analysis is already done, and it is identified as benign.

(b) The local S-VMD where the VM resides: this is to ensure that the local S-VMD can store this information for future references.

Bot-VMD can also group the S-VMs based on the values received by S-VMD (packet S-to-B). This is done as follows.

- If the S-VMD is confirmed as benign VM after the forensic analysis, then the peerVMs of the S-VMD which gave a negative VMT so that VMs are identified. These identified peerVMs are suspected as malicious VMs and a forensic analysis is done on them.
- If the S-VMD was confirmed as botVM after the forensic analysis, then the peerVMs of the S-VMD which gave a positive VMT are identified. These identified peerVMs are also suspected as malicious VMs and a forensic analysis is done on them.

5.6 Operation of bVMD Stage-2 Components

Figure 5.3 illustrates a scenario similar to the one discussed in Section 4.4 in Chapter 4. The scenario has two physical hosts X and Y. Each physical host hosts a bot-VMD component, bot-VMD 1 and bot-VMD 2 respectively, which are placed in the hypervisor. Each bot-VMD has the following components (1) FVM-Estimate method, (2) FVM create, dispatch and analyse method (D&A), and (3) bot-VM table (bot-VMT-T).

In this scenario, the bot-VMD 2 receives a S-to-B packet (Figure 5.1) from S-VMD1 notifying about a low CredV of VMc. This packet is received by FVM Estimation method. The information (CredV of VMc and the details about the peerVMs) from this packet is saved in bot-VMT-T. Based on the number of the S-VMs in the list and the number of malwares to be analysed, it determines the number of FVMs to be used. This number and types of FVMs estimated is communicated to D&A method. This method creates the required FVMs and dispatches them. The FVMs record the symptoms at the VM(s) and save information in bot-VMT-T. This data is analysed, and appropriate actions are taken.

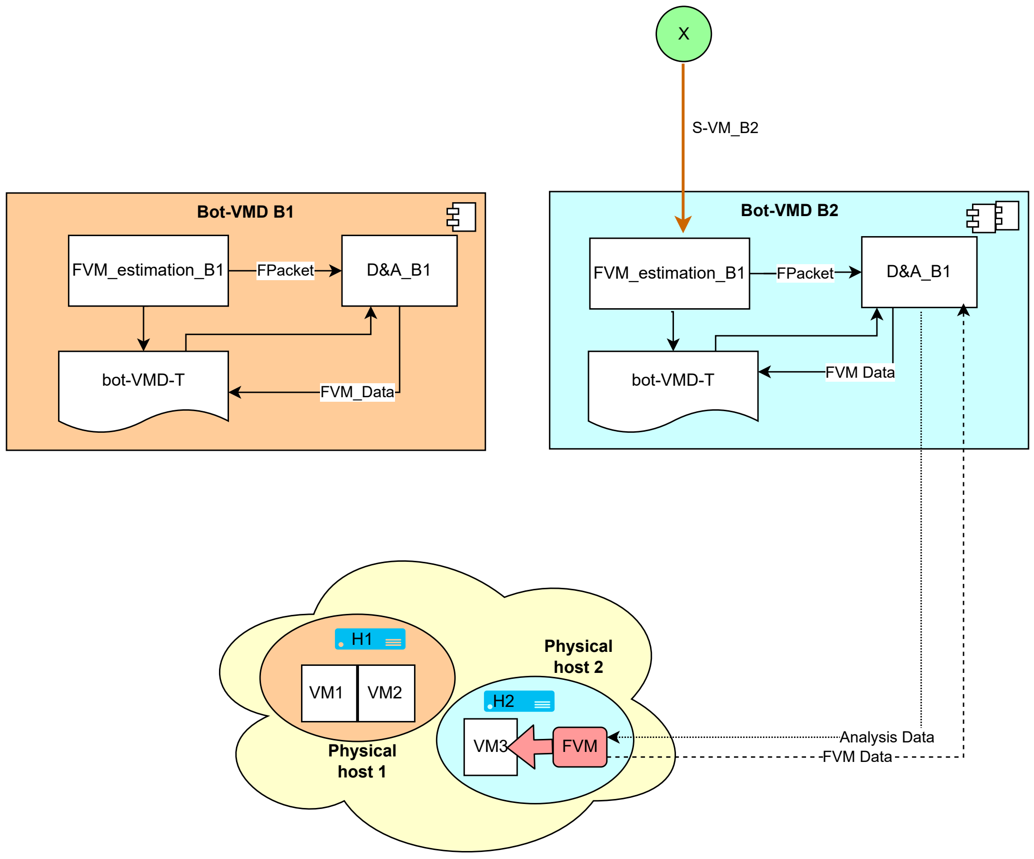


Figure 5.3: botVMD and FVM Operations

5.7 Evaluation Methodology

A test environment (similar to the one described in [81]) was set up to evaluate this stage of the bVMD Framework. A virtualised environment was created using Xen 4.5 [143] as the VMM. Then several VMs running on HVM mode with Windows-7 professional 32-bit as the operating systems were created. VMs running on Xen should run either in Hardware-assisted VM (HVM) or in Para Virtualisation (PV) mode. In HVM, VMs are not aware that they are running in a virtualised environment, whereas in PV mode, VMs experience some modification and are aware that they are running on a virtual platform. LibVMI introspection tool [144] is used to conduct VMI. In conjunction with VMI, volatility framework version 2.4 is also used; it is a forensic memory analysis framework that can significantly aid in performing useful memory analysis tasks. The data is collected from Dom0 [145] for these experiments.
Experiments were done by collecting data from 8 VMs [81] and we consider the same number of malwares, i.e. 8 malwares. We chose the same malware to infect all the VMs. This is to make sure that the FVM takes similar time to study each S-VM. A botnet related malware we used is Zbot [146].

Three scenarios were implemented for comparing the performance and overheads incurred in using different types of FVMs. The scenarios are 1) the V-SF approach, 2) the S-SF approach and 3) the proposed hybrid approach. The performance in terms of total time taken for inspection is investigated with the size and number of FVMs involved. These experiments were repeated 10 times and an average is calculated.

5.8 Experimental Results and Discussions

This section describes the results of the experiments carried out to evaluate the performance of the Stage-2 components of the bVMD Framework. The parameters that have been investigated are: 1) the time taken by the FVM(s), 2) the number of FVMs, and 3) the total memory space taken by FVMs. All parameters are studied for all the symptoms across all the S-VMs with respect to change in the number of S-VMs and the number of symptoms. The results are depicted in Figure 5.4 to 5.8.

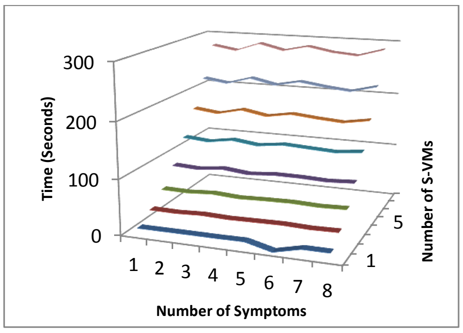


Figure 5.4: Time taken by FVMs to analyse all the symptoms in all the VMs in S-SF

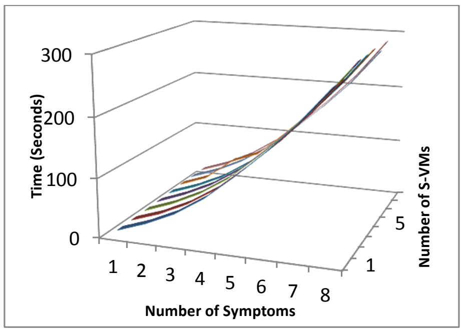


Figure 5.5: Time taken by FVMs to analyse all the symptoms in all the VMs in V-SF

Figure 5.4 and 5.5 depict the time taken by FVMs to detect S-VMs ranging from 1-8 in scenarios using S-SF and V-SF. Figure 5.4 shows the results of S-SF and each line in the figure depicts the total time taken by the FVMs to analyse the S-VMs when the number of symptoms increases. For example, the first horizontal line (blue) from the base shows that when the number of S-VMs are same and the number of symptoms increases, then the total time taken by FVMs remains the same. Whereas Figure 5.5 shows the results of V-SF and each line in the figure depicts that when the number of S-VMs increase and the number of symptoms remain same, the total time by FVMs increases. From these figures, we make the following observations.

Obs 1 In scenario S-SF, the total time taken to detect all the symptoms in a S-VM is similar. This is because in this scenario there is an FVM dedicated for each S-VM and these

FVMs run simultaneously on all the S-VMs. Therefore, they take similar time to detect the same number and type of symptoms across different S-VMs.

Obs 2 In scenario V-SF, the total time taken to detect all the symptoms in an S-VM increases with the increase in the number of S-VMs. This is because an increase in number of S-VMs leads to increase in the total number of symptoms to be analysed (as there is a specific FVM for each symptom). Therefore, the total time taken by FVMs increases. In addition, the total hopping time between S-VMs also increases, thereby increasing the total time taken.

Figure 5.6 shows the increase in the detection time used by FVMs with increase in the number of S-VMs. In the figure, S-S-'n'S refers to a scenario using a symptom-specific FVM with 'n' number of symptoms and V-SF-'n'S refers to a scenario using a VM-specific FVM with 'n' number of symptoms. From this figure we make the following observations.

Obs 1 In scenario using V-SF, the total time taken by FVMs (to detect all the symptoms in all the VMs) increases at a rate of 73% per additional symptom in the number of S-VMs, whereas in scenario using S-SF, it increases with an average of 9%. This is because in scenarios using V-SF, with increase in the number of S-VMs, the number of FVMs increases. An increase of an FVM involves time for step-1 to step 4 (as discussed in Section 5.2). The more FVMs to be initiated at a time, the more it takes for the host to initiate an FVM (step-1). Whereas in scenario using S-SF, with an increase in number of S-VM, the time for steps 2-3 (as discussed in Section 5.2) increases. Therefore, we can see a steep increase in scenario using V-SF as compared to scenario S-SF.

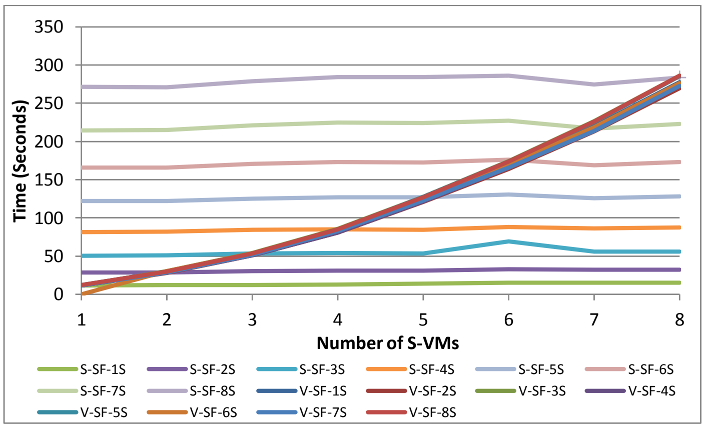


Figure 5.6: Time taken for analysis with increase in the number of S-VMs in scenarios V-SF & S-SF

Obs 2 Comparing the two scenarios in Figure 5.6, we observe that the scenario using V-SF takes less time for analysis when the number of symptoms is less than the number of S-VMs. Whereas S-SF takes less time for analysis when the number of symptoms exceeds the number of S-VMs. The reason behind this is that, with increase in the number of symptoms (exceeding number of S-VMs), scenarios using S-SF will have more FVMs, thus analysing the S-VMs quicker. When the number of symptoms is less than the number of S-VMs, scenario using V-SF will have more FVMs, thus analysing the S-VMs quicker.

Figure 5.7 and 5.8 gives the total FVM size required in scenarios S-SF & V-SF with increase in number of S-VMs & symptoms. Here FVM size is calculated as (total number of FVMs · size of each FVM). We assume that each FVM requires the same amount of memory to analyse a symptom. If it requires ‘x’ bytes of memory to store an information for analysing a symptom & there are ‘s’ symptoms, then total memory space required by the FVMs is ‘x · s’ bytes. This space is shown in Figure 5.7 (scenario using S-SF) and 5.8 (scenario using V-SF).

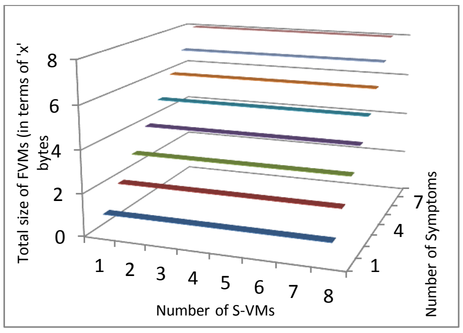


Figure 5.7: Number of FVMs with change in the number of VMs and symptoms in scenarios using S-SF

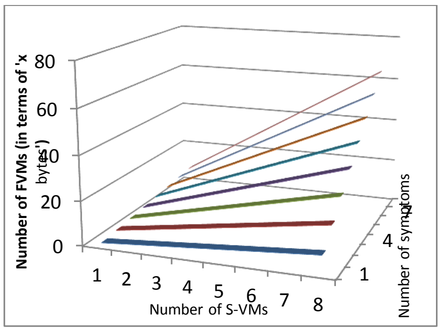


Figure 5.8: Number of FVMs with change in the number of VMs and symptoms in scenarios using V-SF

In scenarios using S-SF, with an increase in number of symptoms, the total size increases at an average rate of 37%. However, with increase in number of S-VMs, the total size of FVMs remains same. This is because with increase in S-VMs, neither the number of FVMs increase nor the size of an FVM increases. In scenarios using V-SF, with increase in the number of S-VMs or the number of symptoms, the total size of FVMs increases at an average rate of 37%. The reason for a similar rate of increase is that with increase in number of S-VMs, the number of FVMs increases, whereas with increase in number of symptoms, the size of each FVM increases. Here we have assumed that the size required for information related to a symptom and size required for a new FVM is similar, which might not always be the case. The size of a new FVM might be more than the size required for information related to a symptom and this would lead to an increase in the total size of FVMs (with increase in the number of symptoms) in scenario S-SF. The findings from these results are as follows.

1 When the number of symptoms is more than or equal to the number of S-VMs, scenario using V-SF gives quicker results, however this might introduce more overheads as compared to scenarios using S-SF.

2 When the number of symptoms is less than the number of S-VMs, scenario using S-SF gives quicker results, however it is accompanied with higher overheads as compared to scenarios using V-SF.

The proposed hybrid method of FVM incorporates these findings. Figure 5.9 and 5.10 show the results for hybrid method.

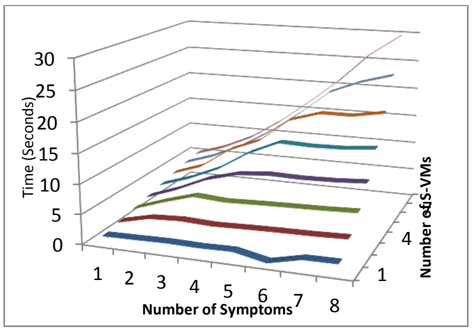


Figure 5.9: Time taken by FVMs to analyse all the symptoms in all the VMs in Hybrid Approach

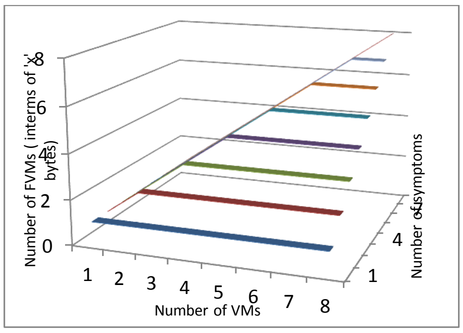


Figure 5.10: Total size of FVMs in Hybrid Approach

From the figures, we can observe that with increase in number of S-VMs, the total time taken by FVMs to analyse all the symptoms across all the VMs increases by 41%. In detail, when the number of symptoms is less than the number of VMs, the time increases by 56% and otherwise the time increases by 25%. Also, with increase in number of S-VMs, the total size of FVMs increases by 24% In detail, when the number of symptoms is less than the number of VMs, the total size of FVMs increases by 25%, otherwise it increases by 23%.

5.9 Chapter Summary

This chapter has presented the design and simulation study of stage-2 components of bVMD Framework. The components, Bot-VMD and FVM aim to detect the set of botVMs by doing an in-depth forensic analysis on the identified S-VMs. A novel algorithm is proposed to determine the number of FVMs, and the effectiveness is depicted along with the overheads. The simulation study has shown that with increase in number of S-VMs, the detection time increases at a rate of 41% with an increase of 24% overheads.

Chapter 6

bVMD Overall Evaluation and Discussions

6.1 Chapter Introduction

This chapter gives a detailed analysis of the results (in terms of effectiveness, efficiency and scalability) obtained from the evaluation of the bVMD Framework. The results are studied against the requirements specified for an effective, efficient and scalable botcloud detection solution. Also, performance of the bVMD is analysed in scenarios with no attacks and in scenarios with attacks. This chapter also compares the obtained results with the most related work.
The structure of this chapter is as follows. Section 6.2 analyses the bVMD Framework in terms of requirements, detection accuracy and security. Section 6.3 compares the results with the most relevant work. Finally, Section 6.4 summarizes the chapter.

6.2 Analysis of the bVMD Performance

The bVMD Framework analysis is performed in terms of requirements, detection accuracy and security analysis.

6.2.1 Requirements Analysis

The section analyses the bVMD framework against the requirements specified in Section 3.4.

1 Functional Requirements (FUL)

- **FUL1**: The FUL1 requirement is fulfilled by bVMD Framework, i.e. it can detect botclouds with known and unknown signatures. This is achieved by using anomaly-based techniques instead of signature-based techniques. These anomaly-based techniques are network-based IDS (Stage-1) and host-based IDS (Stage -2).
- **FUL2**: The FUL2 requirement is fulfilled by bVMD Framework, i.e. it can detect botclouds regardless of their Command and Control (C&C) message formats, structures and protocols. This is achieved by using frequency of packets for detection malicious activities. This is independent of the contents, protocols or structures of the packets.

2 Effectiveness Requirements (EFS)

- **(EFS1) Minimise False Positive Rate (FPR)**: The EFS1 is partially fulfilled by bVMD Framework, i.e. it detects botVMs with as low FPR as possible in some cases.

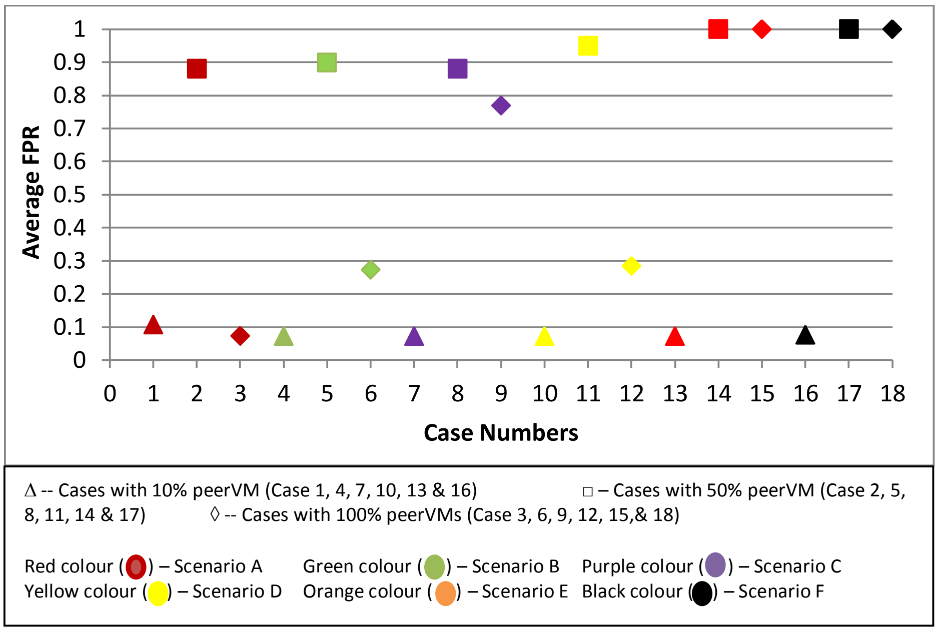


Figure 6.1: Average FPR for each case

The Figure 6.1 shows average FPR for 18 cases. To study the FPR of each case, an average FPR is calculated for 13 set of VMs ranging from 3 to 15 over a period of 19 minutes. Every point in the graph depicts the average FPR for a case. From Figure 6.1 we can make the following observations.

– The cases with 10% peerVMs (Δ) always have lower FPR than the other cases in its scenario (Figure 6.2). This is because, in these cases, CredV of a VM is affected by a smaller number of VMs as compared to other cases. Thus, the propagation of negative VMT is slower, leading to a low FPR.

Table 6.1: EFS for bVMD Framework

Effectiveness Requirements					
Scenarios	Cases	EFS1 Minimise FPR	EFS2 Maximise TPR	EFS3 TNR	EFS 4 FNR
A	1	10.77%	54.23%	89.23%	45.77%
	2	88.08%	85.38%	11.92%	14.62%
	3	7.31%	17.31%	92.69%	82.69%
B	4	7.31%	7.69%	92.69%	92.31%
	5	90.00%	100.00%	10.00%	0.00%
	6	27.31%	100.00%	72.69%	0.00%
C	7	7.31%	7.69%	92.69%	92.31%
	8	88.08%	100.00%	11.92%	0.00%
	9	76.92%	100.00%	23.08%	0.00%
D	10	7.31%	7.69%	92.69%	92.31%
	11	95.00%	100.00%	5.00%	0.00%
	12	28.46%	35.00%	71.54%	65.00%
E	13	7.31%	7.69%	92.69%	92.31%
	14	100%	95.00%	0.00%	5.00%
	15	100%	95.00%	0.00%	5.00%
F	16	7.69%	7.31%	92.31%	92.69%
	17	100%	90.00%	0.00%	10.00%
	18	100%	76.92%	0.00%	23.08%
Average		52.71%	60.38%	47.29%	39.62%

- The cases with 50% peerVMs (□) always have high FPR as compared to the other two cases in its scenario. Comparing with corresponding case with 10% peerVM, the negative VMT value propagates faster, thus the FPR increases. Compared to corresponding cases with 100% peerVM, the negative VMT propagates faster, however at the same time, the number of VMs rating each other increases. This increases the effect of positive VMT value and decreases the overall effect of negative VMT. Therefore, the FPR decreases.
- As the proportion of targetVM to botVM increases (i.e. from scenario A to B to C), the FPR increases with increasing the number of peerVMs. This is because with higher number of targetVMs, the CredV of botVM goes down swiftly and with increasing number of peerVMs, this CredV propagates quickly.

  In scenario D, the proportion of targetVM and botVM is equal, leading to a decrease in FPR. This value is almost equal to Case 6, i.e. 50% targetVM and 100% peerVM. This is because with equal number of targetVMs, the CredV of botVM is affected in a similar way. The difference in the number of botVMs

doesn't impact the results.
In scenario E and F, the FPR is high. This is because of high number of botVMs in these scenarios.

– As the percentage of botVMs increases in these cases, the FPR increases. This is because of the obvious reason of an increase in the number of botVMs, which makes the effect of negative VMT propagate quickly.
– The results are shown in Table 6.1. The average FPR is 53%. The FRR with 10% peerVM is 8%, with 50% peerVM is 94% and with 100% peerVM is 57%.

- **(EFS2) Minimise False Negative Rate (FNR)**: The EFS2 is partially fulfilled by bVMD Framework, i.e. it detects botVMs with as low FNR as possible in some cases.

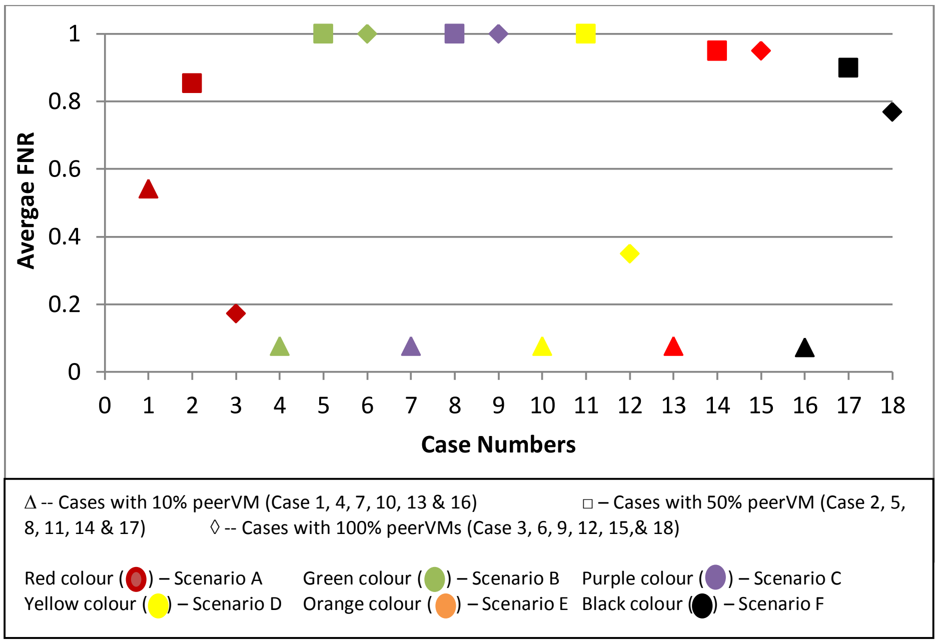


Figure 6.2: Average FNR for each case

The Figure 6.2 shows average FNR for 18 cases. Every point in the graph represents the average FNR for a case. The average FNR is 40%. The FNR with 10% peerVM is 85%, with 50% peerVM is 5% and with 100% peerVM is 23%.

- **(EFS3) Maximize True Positive Rate (TPR)**: The EFS3 is partially fulfilled by bVMD Framework, i.e. it detects botVMs with as high TPR as possible in some cases. The Figure 6.3 shows average TPR for 18 cases. To study the TPR of each case, an average TPR is calculated for 13 set of VMs ranging from 3 to 15 over a period of 19 minutes. Every point in the graph depicts the average TPR for a case.

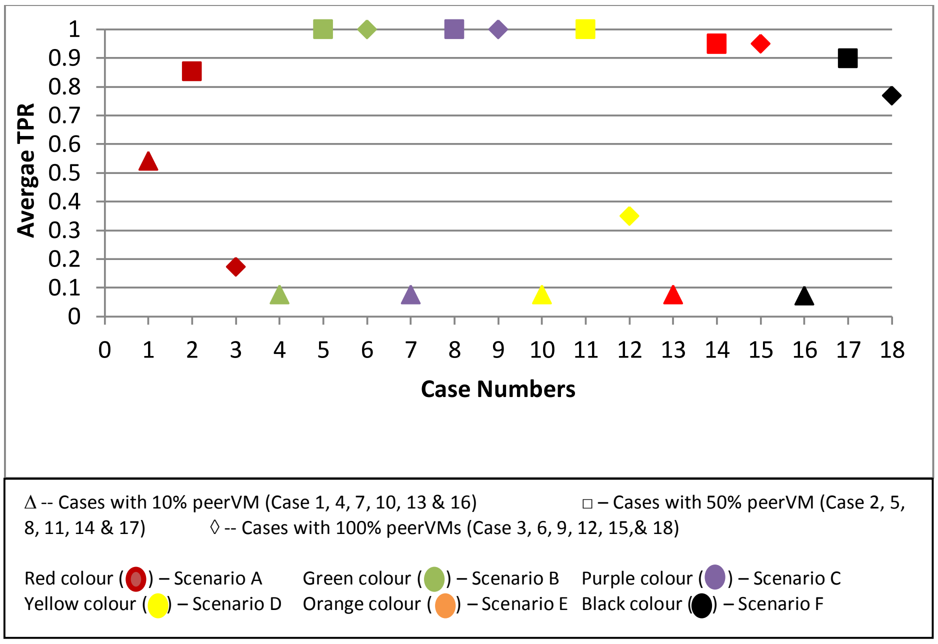


Figure 6.3: Average TPR for each case

From Figure 6.3 we can make the following observations.

- The cases with 50% peerVM always have more TPR as compared to the other two cases in the same scenario. This is because, with high peerVMs (50%) as compared to option-1, more peerVMs affect the CredV of a VM, giving positive VMT value. Therefore, the TRP increases.
- With increase in peerVMs from 50% to 100%, the TPR is high (close to 1) for scenarios B and C. It decreases for scenarios E and F and further reduces in scenario D and then A.

  In case 6 and 9, the number of targetVMs increases as compared to the number of botVMs. With increase in number of targetVMs, the TRP increases. As the number of botVM increases as compared to targetVMs, the TRP decreases. This is because of the obvious reason of increase in botVMs which increases the indirect effect of negative VMT.

  In cases 3 and 12, the TRP decreases. This is because of the same number of botVMs and targetVMs. In case 3, botVM and targetVM receives similar positive VMT from stdVM (peerVM). BotVM gives positive VMT to targetVM and targetVM gives positive VMT to botVM. As negative VMT propagates, the CredV of targetVM decreases, thus decreasing TRP. Similarly, in case 12, the negative VMT of botVM propagates, thus decreasing TRP.
- In addition, as the number of targetVMs increases as compared to the number of botVMs, the TPR also increases. This is because of the obvious reason of targetVMs increasing and giving negative VMT to the botVMs. However, as

the number of botVMs increase, the TPR value reduces. This is because of the propagation of negative VMT by a large set of botVMs.

– The results are shown in Table 6.1. The average TPR is 60%. The TPR with 10% peerVM is 15%, with 50% peerVM is 95% and with 100% peerVM is 71%.

- **(EFS4) Maximise True Negative Rate (TNR)**: The solution should be able to detect botVMs with as high TNR as possible.

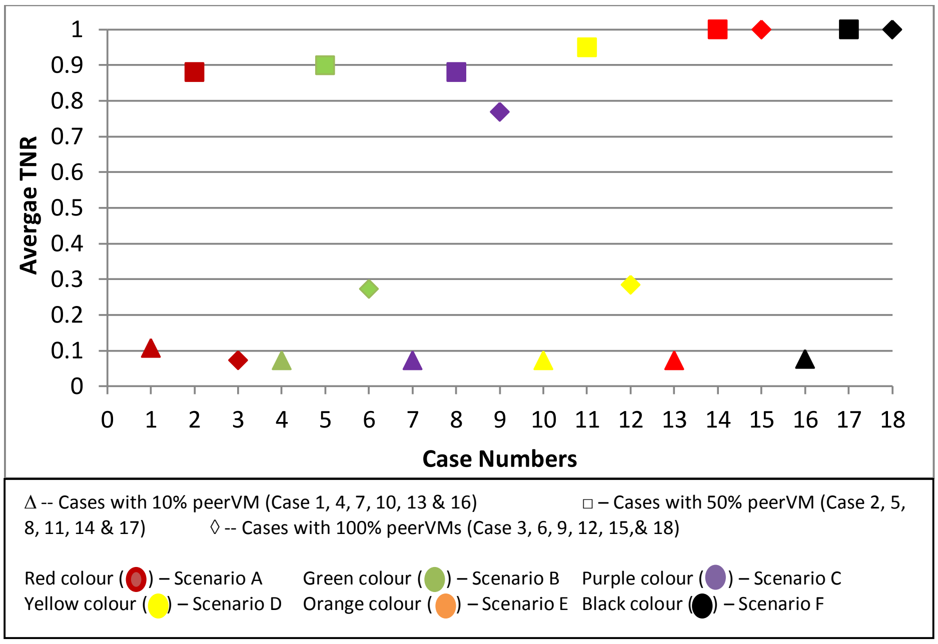


Figure 6.4: Average TNR for each case

The Figure 6.4 shows average TNR for 18 cases. Every point in the graph represents the average TNR for a case. The average TNR is 47%. The TNR with 10% peerVM is 92%, with 50% peerVM is 6% and with 100% peerVM is 43%.

3 Efficiency Requirements (EFY)

- **(EFY1) Low communication overhead**: The EFY1 is fulfilled by bVMD Framework. C-OH is calculated by the number of messages and the length of each of the messages (in bits) exchanged between the architectural components. To do so, we study the messages sent between the architectural components.
  - Exchange between VMWatchers: V-to-V packets (Figure 4.7)
  - VMWatcher to S-VMD: V-to-S packets (Figure 4.8)
  - S-VMD to bot-VMD: S-to-B packets (Figure 5.1)

  The size of each of these are given below.

- The total size of V-to-V packet is 40 bits for each peerVM. If there are 'p' peerVMs of a VMWatcher, then the total size of this packet is $(40 \times p)$ bits.
- The total size of V-to-S packet is 40 bits for each peerVM which has negative VMT value. The maximum number of such peerVMs can be 'p', therefore the maximum size of this packet is $(40 \times p)$ bits and the minimum size is (40×1) bits.
- The total size of S-to-B packet is $(48 \times (p1+1))$, here p1 is the number of peerVMs of the S-VM which has high VMT on S-VM and low CredV.

The total size of the communication packets is $((80 \times p) + (48 \times (p1+1)))$ in bits. Referring to Figure 4.9, the total communication overhead will be calculated as Table 6.2.

Type of Packet	Prime VMWatcher	VMWatcher related to Prime VMWatcher	Size of Packets (bits)
V-to-V	VM_a	2 (VM_b and VM_c)	80
	VM_b	1 (VM_a)	41
	VM_c	1 (VM_a)	41
V-to-S	VM_a, VM_b and VM_c	3	120
S-to-B	S-VMD2 to botVMD (VM_c)	1	96
Total size of packets			378

Table 6.2: Communication overhead for Figure 4.9

The Table 6.2 shows the packet size which is used for communication in an example Figure 4.9. As per the figure, there are a total of three VMWatchers, VM_a, VM_b and VM_c and two S-VMDs (1 and 2). The first packet V-to-V is sent from a VMWatcher to its peerVMs, so here VM_a sends a V-to-V packet each to VM_b and VM_c. The second packet V-to-S is sent by each VMWatcher to its corresponding S-VMs, therefore three are three packets. The third packet S-to-B is sent from S-VMD to bot-VMD for only an identified S-VM. Here we assume that VM_c is a S-VM, therefore one packet is sent to its botVMD. The total of 378 bits are communicated during this process.

- **(EFY3) Low storage overhead**: The EFY3 requirement is fulfilled by bVMD Framework. S-OH is calculated by the length of data stored by each component. To do so, we study the data stored by each architectural component.
  - VMWatcher: It stores VMT-T
  - S-VMD: It stores S-VMT-T
  - Bot-VMT: It stores Bot-Cred-T

The VMT-T table requires 64 bits space for each peerVM, therefore $(64 \times p)$ bits for 'p' peerVMs. The S-VMD-T requires $(136 \times n)$ bits for 'n' number of VMWatchers in the system. The bot-VMT-T has to store S-to-B packet, therefore requires (48 × (p+1)) bits for each VMWatcher. If a 'x' VMWatchers send this S-to-B packets, then it requires a space of ((48 × (p+1)) × x). This could range from ((48 × (p+1)) × 1) to ((48 × (p+1)) × n), where 'n' is the total number of VMWatchers in the system. The total storage space required by bVMD Framework is ((64 × p × n) + (136 × n) + ((48 × (p+1)) × n)) bits.

Storage Table	Prime VMWatcher	VMWatcher related to Prime VMWatcher	Total size (bits)
VMT-T	VM_a	2 (VM_b and VM_c)	128
	VM_b	1 (VM_a)	64
	VM_c	1 (VM_a)	64
S-VMT-T	VM_a, VM_b and VM_c	3	408
Bot-VMT-T	S-VMD2 to botVMD (VM_c)	3	288
Total size of packets			952

Table 6.3: Storage overhead for Figure 4.9

The Table 6.3 shows the total storage size which is used in example Figure 4.9. The first table VMT-T is stored by each VMWatcher. The second table S-VMT-T is stored by each S-VMD and the third table bot-VMT-T is stored by each botVMD. From the table we can see that a total of 952 bits are stored during this process.

4 Scalability Requirements (SCY)

- (SCY1) bVMD solution should be scalable: The SCY1 is fulfilled by bVMD Framework. This is analysed by the MCC curve for all the 18 cases.

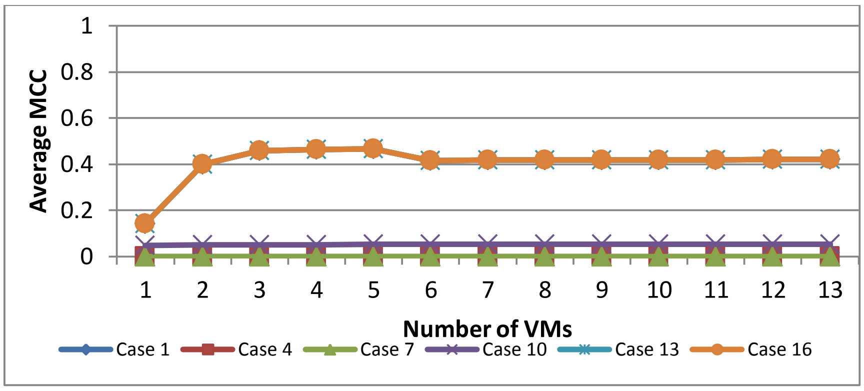


Figure 6.5: Average MCC curve for Cases with 10% peerVMs

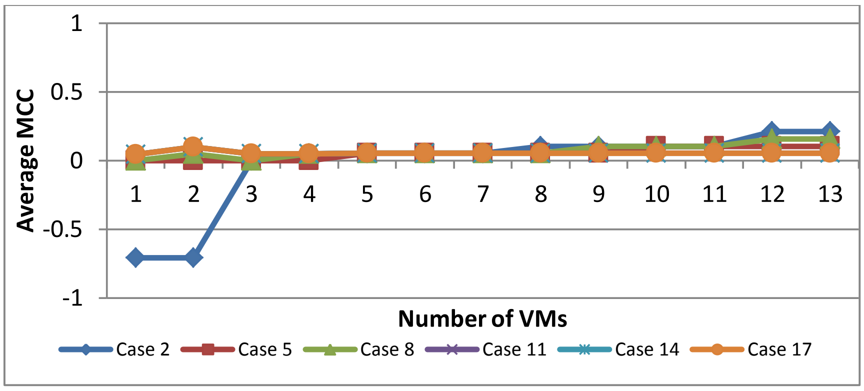


Figure 6.6: Average MCC curve for Cases with 50% peerVMs

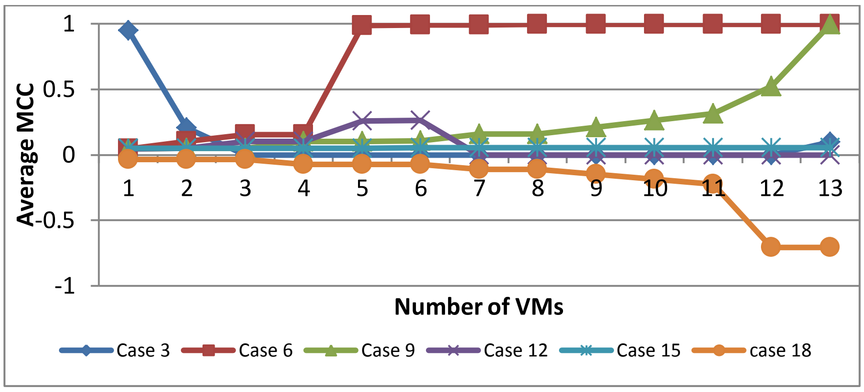


Figure 6.7: Average MCC curve for Cases with 100% peerVMs

The Figures 6.5 - 6.7 shows the average MCC of 13 sets of VMs ranging from 3 to 15 VMs for 19 minutes. They depict the average MCC value of each case with increasing number of VMs. From the Figures we can observe that the average MCC values either increase (Case 9) or stabilize as the VMs increase. This means that with increase in the number of VMs, the effectiveness of bVMD either increases or stabilizes, thus supporting scalability.

6.2.2 Detection Accuracy

The detection accuracy of the bVMD Framework is divided into two categories. First is the effectiveness by which the group of botVMs and targetVMs are grouped correctly and second is the effectiveness with which the groups are identified correctly. The Figure 6.8 shows the effectiveness of bVMD Framework in terms of grouping.

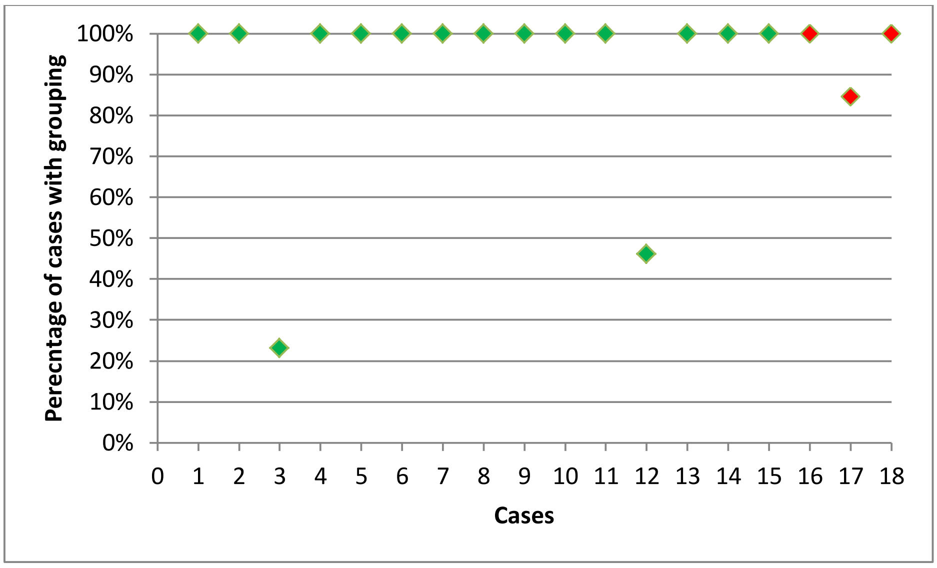


Figure 6.8: Grouping Accuracy

Every point in the graph represents the percentage of the cases of VMs (3-15) which have correctly grouped the VMs into two sections. Here 'correctly' means that all the botVMs are in one group and all the benign VMs are in the other group. The green dots in the graph show that the VMs are correctly grouped and correctly identified, whereas red dots in the graph show that the VMs are correctly grouped but wrongly identified. The results taken from 18 cases for 13 set of VMs, show that in 92% sets, the VMs are correctly grouped. Out of these sets, 83% cases are correctly identified, and rest 17% cases are wrongly identified. The effectiveness of bVMD in terms of detecting the botVMs is same as effectiveness of bVMD in terms of grouping shown in Figure 6.8. This is because if VMs are grouped correctly, then the following can happen.

- The groups can be identified correctly or cannot be correctly identified by Stage-1.

If the groups are identified correctly, then Stage-2 detects them within 't' minutes. Otherwise, the time taken by Stage-2 to detect the correct group increases to '2·t' minutes. This is because, first it analyses the incorrectly identified VMs and when it finds that these VMs (first set) are not botVMs, then it analyses the other set (second set) of VMs. This wrong identification of the first set of VMs is due to the incorrect trust values given to them by the second set of VMs during stage-1 of bVMD Framework. Therefore, the second set of VMs are most likely the correct set of botVMs. This overall process of analysing two set of VMs before identifying the correct set of botVMs increases the overall time for detection, however doesn't affect the detection accuracy.

In the calculations below, total 234 sets were considered. Each Case had 13 sets (VM3-15), therefore with 18 cases the total sets considered were 234.

The bVMD can correctly detect botVMs in 92% sets. In more detail, bVMD can correctly detect 100% sets in cases all the cases except case 3, 12 and 17. In case 3, bVMD can detect 23% sets of VMs, in case 12 and 17 it can detect 46% and 86% cases respectively.

6.2.3 Security Analysis

The performance of the bVMD Framework is analysed in scenarios with three types of attack: i) badmouthing, ii) ballot stuffing and iii) combined attack. The results are shown in Table 6.4. In a badmouthing attack, the bVMD can correctly detect 57% sets. In detail, 100% sets in Case 1, 2, 8, 10, 11 and 13% sets in Case 3, 7, 14, 15, 16 and 17. 85% sets in Case3, 78% sets in Case 5, 46% in Case 6, 54% sets in Case 12, 85% sets in Case 9 and Case 18. Out of the 57% sets, 66% sets were correctly identified by bVMD Stage-1.

Table 6.4: Detection Accuracy in Scenarios without attack and with attack

	Detection Accuracy (%)			
	No Attack Scenario	Bad Mouthing Attack	Ballot Stuffing Attack	Combined Attack
Case 1	100%	100%	--	--
Case 2	100%	100%	--	--
Case 3	23.08%	84.62%	--	--
Case 4	100%	100%	--	--
Case 5	100%	77.92%	--	--
Case 6	100%	46.15%	--	--
Case 7	100%	0%	--	--
Case 8	100%	100%	--	--
Case 9	100%	84.62%	--	--
Case 10	100%	100%	0%	0%
Case 11	100%	100%	0%	0%
Case 12	46.15%	53.85%	--	--
Case 13	100%	100%	0%	0%
Case 14	100%	0%	100%	0%
Case 15	100%	0%	--	--
Case 16	100%	0%	100%	84.62%
Case 17	85.62%	0%	100%	76.92%
Case 18	100%	84.62%	--	--
Overall	91.88%	57.27%	50%	26.92%

In a ballot stuffing attack, the bVMD can correctly detect 50% sets. In detail, sets in cases 10, 11 and 13 cannot be detected at all, while sets in cases 14, 16 and 17 can be correctly detected 100%. Out of the 50% sets which can be detected, there are no sets which are correctly identified by bVMD Stage-1. In a combined attack, the bVMD can correctly detect 27% sets. In detail, sets in cases 10, 11, 13 and 14 cannot be detected at all, while sets in case 16 can be detected 85% times and sets in case 17 can be detected 77% times. Out of the 27% sets which can be detected, there are no sets which are correctly identified by bVMD Stage-1.

6.3 Comparison with Related Work

This section compares the bVMD Framework against the desired requirements and the most related solution.

6.3.1 Comparison against the desired requirements

This section analyses the bVMD Framework against the desired requirements specified in Section 2.5.

Table 6.5: State-of-the-art of the solutions against desired requirements

Solutions	**FUL**	**EFS**	**EFY**	**SCY**
Sol1	✗	✓*	✗	✓*
Sol2	✗	✓*	✓*	✓*
Sol3	✓	✓	✗	✗
Sol4	✗	✗	✗	✗
Sol5	✗	✗	✗	✗
Sol6	✗	✓*	✗	✗
Sol7	✗	✓	✗	✗
Sol8	✗	✓*	✗	✓*
Sol9	✗	✓	✗	✗
Sol10	✗	✓*	✓*	✓
bVMD	✓	✓$^+$	✓$^+$	✓$^+$

✓: The solution has considered the corresponding requirement.
✓*: The solution has considered the corresponding requirement, however there is a scope of improvement.
✓$^+$: The solution has considered the corresponding requirement; however it is not satisfied in all the cases.
✗: The solution has not considered the corresponding requirement.

6.3.2 Comparison of MCC value against the most relevant work

This section describes the effectiveness of bVMD in comparison with related work [82]. To the best of our knowledge, the evaluation in this related work is the most relevant to our work. The performance comparison is based on the detection accuracy in terms of MCC values.

To compare the MCC values with [82], we evaluated the bVMD Framework with similar approach of using UDP flood attack. To compare the number of parameters (i.e. botVMs, targetVMs and stdVMs), we identified that [82] uses 39 botVMs, 24 targetVMs and a large number of stdVMs (1250-24). The nearest bVMD scenario comparable to these parameters is scenario A as it has 1% botVM, 1% targetVM and 98% stdVM. So, the bVMD case 1, case 2 and case 3 (Scenario A) are analysed against the results in [82]. In Figure 6.9 the results of case 2 (red) overlaps the results of case 1 (blue).

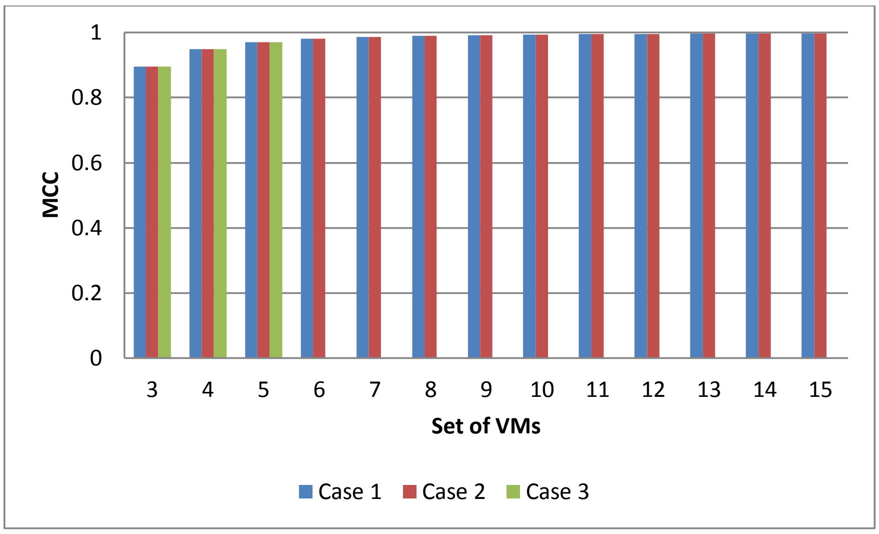


Figure 6.9: bVMD MCC for Case 1, 2 and 3

The results in [82] show that the best MCC value is increasing from 0.72 to 0.78. Whereas, the bVMD Framework shows that the best MCC value is 0.99 (i.e. for 15 set of VMs in the cases 1 and 2). In addition, bVMD Framework shows an average MCC value of 0.99 for cases 1 and 2. This is higher than the best MCC values in [82]. However, case 3 in bVMD Framework has an average MCC value of 0.21, which is lower than the best MCC value in [82]. From these results, we conclude that bVMD shows better results in case 1 and 2, as compared to [82]. As the peerVMs in bVMD increase (case 3), the results deteriorate. The MCC curves of bVMD Framework shown in Figure 6.9.

6.4 Chapter Summary

This chapter has presented the evaluation results of bVMD Framework. The results are analysed in terms of requirements, detection accuracy and security. In addition, these results were compared with the most relevant work.

Chapter 7

Conclusions and Future Work

The focus of this thesis has been to investigate how to detect a botcloud in an effective, efficient and scalable manner. This chapter summarises all the research findings and contributions of this thesis and gives recommendations for future work.

7.1 Contributions

The contributions of this thesis are summarised on a chapter-to-chapter basis as follows.

- **Chapter 3**: In this chapter, a set of requirements for an effective, efficient and scalable botcloud detection solution are specified. In addition, a set of design measures are specified. To support the identified requirements and realise the specified measures, design of a novel BotCloud Detection Framework (bVMD) is proposed. The bVMD comprises of a two-stage novel architecture, with the first stage aiming to identify a set of suspected malicious VMs and the second stage aiming to further analyse the set of identified VMs to ensure that they are certainly malicious (botVMs). The first stage uses a novel network-based intrusion detection method and the second stage uses an improved host-based intrusion detection method. Both these methods are outlined in this chapter.
  This chapter also describes the evaluation methodologies used in evaluating the bVMD Framework including a set of 18 scenarios (with different percentages of botVMs, targetVMs, stdVMs and peerVMs) with their respective settings. These scenarios are used for evaluating the performance of bVMD Framework.
- **Chapter 4**: In this chapter, the novel network-based IDS method for identifying suspected malicious VMs has been proposed.

The network-based IDS is built on a trust-based mutual monitoring approach which is implemented by using two components, namely VMWatcher and S-VMD. The aim of this approach is to reduce the number of VMs to be further analysed in Stage-2. To achieve this, all the VMs in the network are analysed using a peerVM watch method and credibility of each VM is calculated. VMs with low credibility are suspected as malicious VMs and sent for further analysis in Stage-2. The detailed design of both the Stage-1 components is proposed in this chapter.
Experimental evaluation has been carried out on Omnet++ using 18 different scenarios. With this evaluation, the effectiveness in terms of detection accuracy (MCC curve) is measured. The results demonstrate that it is highly effective in case 6 and 9, with an average MCC value of 0.99 (calculated for 13 sets of VMs over a duration of 19 minutes). Whereas it is not effective in case 18, with an average MCC value of -0.04. In addition, it is not effective in cases 4 and 7, with zero average MCC value. In all the other cases, the average MCC value ranges from 0.05 to 0.95.
Also, the attacks that can be mounted on these design components have been studied. The possible attacks are: i) badmouthing ii) ballot stuffing and iii) combination. To measure the effectiveness of the bVMD Framework under these attacks, the 18 scenarios are evaluated in each attack type. The results are as follows.

1 In a badmouthing attack, results demonstrate that Stage-1 is effective in case 3 and 6, with an average MCC value of 0.95 with maximum average value of 0.99 The maximum average value is for set of 15 number of VMs calculated over a duration of 19 minutes. Stage-1 is least effective in cases 10 and 11, with maximum average MCC value of -0.04. In addition, it is not effective in cases 4, 7, 14-17, with zero MCC value. In rest of the cases, the maximum average MCC value ranges from 0.01 to 0.89.

2 In a ballot stuffing attack, results demonstrate that Stage-1 is not effective. Case 14, 15 and 16 have an average MCC value of -0.04 each and cases 10-13 have an average MCC value of zero.

3 In a combined attack, results demonstrate that Stage-1 is not effective. Case 16 and 17 have an average MCC value of -0.03 and -0.03 respectively. The remaining cases, i.e. 10-14 have an average MCC value of zero.

- **Chapter 5**: In this chapter, an improved, host-based IDS for in-depth analysis of suspected-malicious VMs is proposed.
  The host-based IDS is built on two components, namely, botVMD and FVM. They use VMI technique for analysing the memory of the identified VMs. This is facilitated by the use of Forensic VMs (FVMs). These FVMs can be classified into Symptom-Specific FVMs (S-SF) and VM-Specific FVMs (V-SF). With a change in the number of VMs and/or the symptoms, the overheads incurred in both these types of FVMs differ. To minimise the overheads, an algorithm for determining the number and types of FVMs with change in number of VMs and/or symptoms is proposed. In addition, the design of botVMD component is proposed in this chapter.

Experimental evaluation has been carried out using LibVMI introspection tool to measure the overheads used by the proposed algorithm. These are compared with methods using S-SF and V-SF. The results show that with an increase in number of S-VMs, the total time taken by FVMs to analyse all the symptoms across all the VMs increases by 41% as compared to an average of 73% in V-SF and 8% in S-SF. On the other hand, the total size of FVMs increases by 24% as compared to an average of 37% in V-SF and S-SF.

- **Chapter 6**: In this chapter, the performance of the bVMD Framework is analysed and discussed.
  Firstly, the detection accuracy (MCC) of the bVMD is described. Also, the requirements specified for an effective, efficient and scalable botcloud detection solution are analysed against the results obtained from the bVMD Framework. These include functional, effectiveness, efficiency and scalable requirements. The results show that bVMD completely fulfils the functional and scalability requirements, whereas partially fulfils the effectiveness and efficiency requirements.
  Secondly, the detection accuracy of the bVMD Framework is discussed. The results show that in a situation without attacks, an average of 92% sets can be accurately detected. On the other hand (i) in a badmouthing attack, an average of 57% sets can be detected, ii) in a ballotstuffing attack, an average of 50% sets can be detected and iii) in a combined attack, an average of 27% sets can be detected. Thirdly, the performance (MCC) of the bVMD Framework is compared with the most related works available in the literature. The results show that the bVMD Framework has 0.99 (10% & 50% peerVMs) as the best MCC value and 0.22 (100% peerVMs) as the lowest MCC value. Whereas, MCC value in the most related solution ranges from 0.73 to 0.78. Therefore, we conclude that bVMD is more effective when the number of peerVM is 10% or 50%.

7.2 Conclusions

From this research, we can draw the following conclusions.

- The results from the evaluation of the bVMD Stage-1 components demonstrate the following.
  - bVMD Stage-1 results are highly effective (0.99 MCC) in couple of cases. We can also observe that the cases with either high number of botVMs or lesser number of peerVMs cannot be detected effectively (i.e. zero MCC). This ensures that for reducing the false detections there is a certain need of Stage-2 analysis.
  - **No attack scenarios**: In cases with targetVMs more than the number of peerVMs, the bVMD Framework is able to identify the set of botVMs correctly. However, higher the number of peerVMs in a case, larger is the time duration in which the

botVMs can be detected. Whereas in cases with high number of botVMs then targetVMs, the bVMD Framework is either able to detect incorrectly or not able to detect at all. This again depends on the number of peerVMs in a case.

  - **Badmouthing attack**: The results show that the cases with either a low number of peerVMs or a larger number of botVMs or a similar number of targetVM & botVM with some stdVMs having a badmouthing attack cannot be detected by bVMD Framework.
  - **Ballot stuffing attack**: The results show that the cases with either a low number of peerVMs or a larger number of botVMs having a ballot stuffing attack cannot be detected by bVMD Framework.
  - **Combined attack**: The results show that the case with less than 50% of peerVMs with a combined attack cannot be detected by bVMD Framework.

- The results from the evaluation of the Hybrid FVM algorithm proposed in the Stage-2 of bVMD Framework shows that using this approach, the size of FVMs is less than the total size required in V-SF and S-SF. However, along with these encouraging results, we also recognise that this algorithm involves more time for analysing all the symptoms across all the VMs as compared to V-SF and S-SF.
- The results from our evaluation of the bVMD Framework demonstrate the following.
  - Functional Requirements: It fulfils all the functional requirements (FUN1 and FUN2) in all the cases.
  - Effectiveness Requirements: The EFS1, EFS2, EFS3 and EFS4 are partially fulfilled by bVMD Framework. In detail, 50% and 39% cases have FPR (EFS1) and TRP (EFS2) respectively less than 50%. 50% and 39% cases have FNR (EFS4) cases have TNR (EFS3) and FNR (EFS4) more than 50%.
  - Efficiency Requirements: EFY1 and EFY2 are fulfilled by bVMD Framework. The communication overhead (EFY1) is minimized by using in-house analysis and only transferring the results using V-to-V, V-to-S and S-to-B packets. These are discussed in Section 6.2. Similarly, storage overhead (EFY2) is minimized by storing the analysed data, instead of raw data. This is done using VMT-T, S-VMT-T & bot-Cred-T and these are discussed in Section 6.2.
  - Scalability Requirement: The scalability of BVMD Framework is analysed for a maximum of 13 VMs and it shows that bVMD is scalable. However, there is a scope to analyse the performance of bVMD Framework with a larger set of VMs.
  - Detection Accuracy: The results show that bVMD Framework can correctly detect botVMs in 92% sets out of the total 234 sets.
  - Security Analysis: The performance of bVMD Framework was analysed under three types of attacks. Under the badmouthing attack, bVMD Framework could detect 57% cases, under ballot stuffing attack, it could detect 50% of the cases and under a combined attack, it could detect 27% cases.

  - In addition, comparing the results of bVMD Framework to the related solutions, we conclude that bVMD fulfils the functional requirements, whereas it fulfils efficiency, effectiveness and scalability requirements in couple of cases.

  From the results we conclude that the performance of bVMD depends on the mix of number of botVMs, targetVMs, stdVMs and peerVMs in a scenario. The results show that with a high number of botVMs, the performance of bVMD reduces. A high number of targetVMs, the performance of bVMD improves. In addition, stdVMs also play a major role in bVMD performance and the results show that cases with no stdVMs have less detection accuracy as compared to other cases. Also, with an increase in the number of peerVMs, initially the performance increases and then decreases.

- In all the experiments, we have clearly specified the parameters under which different experiments have been carried out. The results obtained from these experiments should only be interpreted based on the specified parameters. With a different specification of the parameters, e.g. in the rate of attack, the time of calculating CredV, etc, different results may be obtained. Therefore, the results reported in this thesis cannot be generalised in the context of the diverse range of cases and parameters.

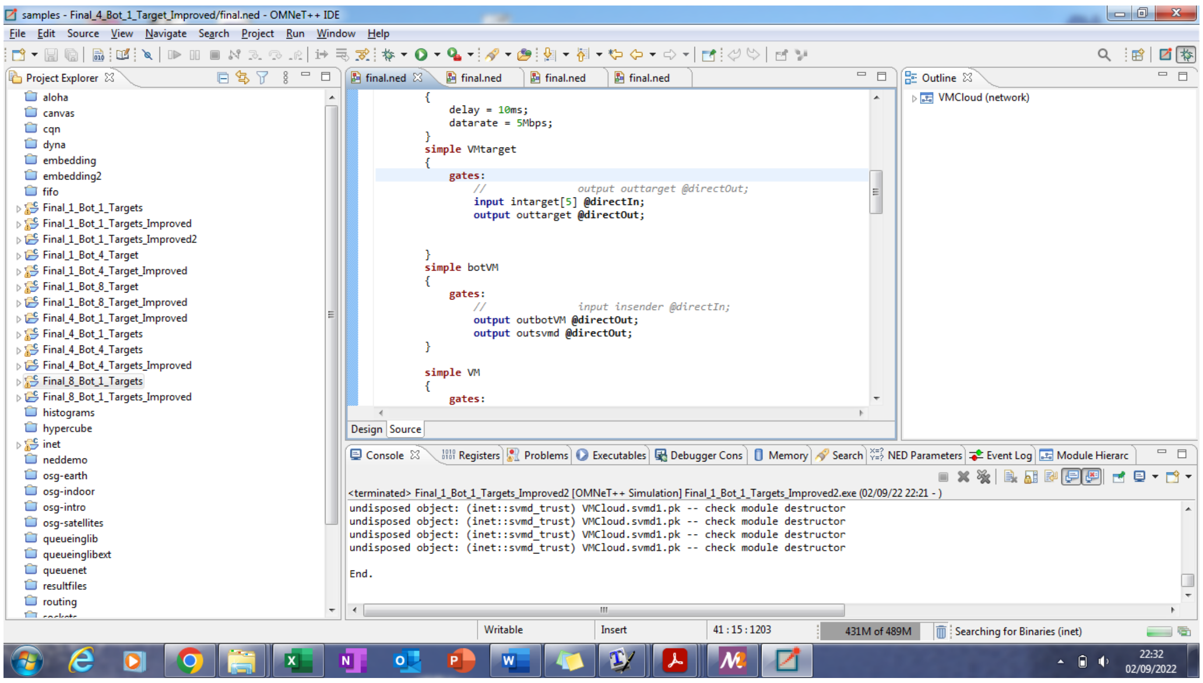
7.3 Future Work

We give following recommendations for future work.

- The bVMD can be extended to detect more type of attacks and the performance of bVMD with these attacks can be evaluated.
- An important consideration with Stage-1 components is the exchange of VMT-T among VMs at regular time-intervals. An algorithm could be placed to minimise the communication and storage overheads incurred from these communications.
- Another important consideration is the inter-host communication used in the bVMD. In our experiments, we use one physical host with all the VMs placed on the same host and an intra-host communication. In a practical deployment of bVMD, inter-host communication could be considered. This could increase the communication time and corresponding overheads.
- In bVMD, Stage-1 and Stage-2 components have been tested on different simulators. To further study the total time taken for detection, they could be implemented on the same environment.

Appendix A

Topologies Used on Omnet++


samples - Final_4_Bot_1_Target_Improved/final.ned - OMNeT++ IDE

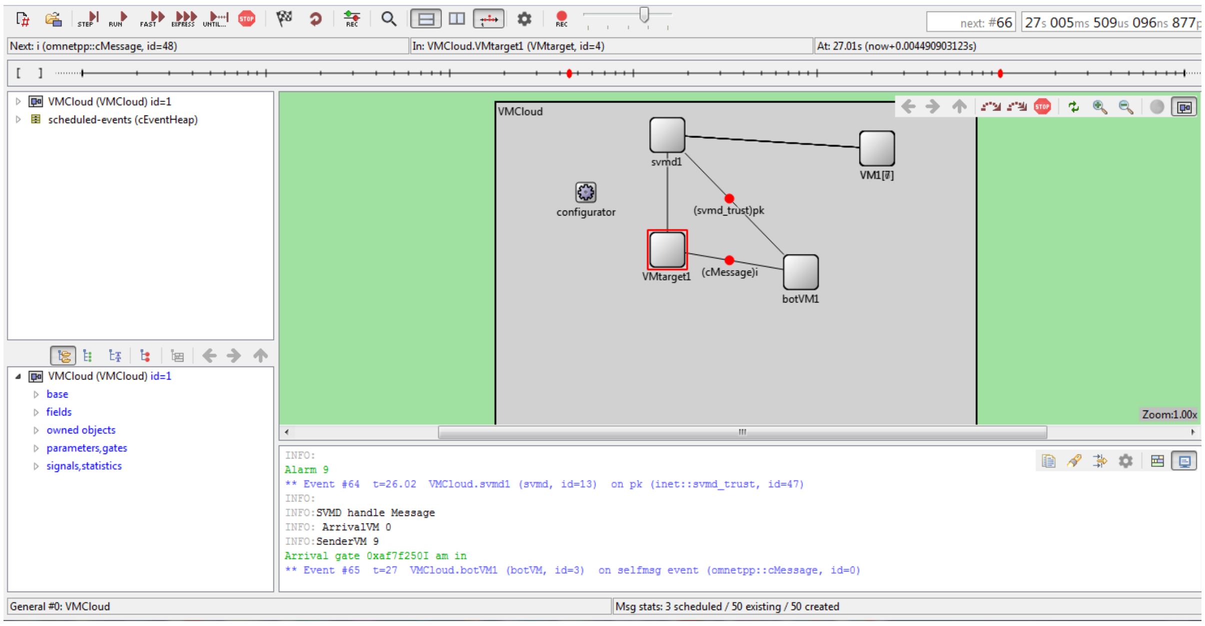
Project Explorer: aloha, canvas, cqn, dyna, embedding, embedding2, fifo, Final_1_Bot_1_Targets, Final_1_Bot_1_Targets_Improved, Final_1_Bot_1_Targets_Improved2, Final_1_Bot_4_Target, Final_1_Bot_4_Target_Improved, Final_1_Bot_8_Target, Final_1_Bot_8_Target_Improved, Final_4_Bot_1_Target_Improved, Final_4_Bot_1_Targets, Final_4_Bot_4_Targets, Final_4_Bot_4_Targets_Improved, Final_8_Bot_1_Targets, Final_8_Bot_1_Targets_Improved, histograms, hypercube, inet, neddemo, osg-earth, osg-indoor, osg-intro, osg-satellites, queueinglib, queueinglibext, queuenet, resultfiles, routing

```
        {
            delay = 10ms;
            datarate = 5Mbps;
        }
        simple VMtarget
        {
            gates:
                //                  output outtarget @directOut;
                input intarget[5] @directIn;
                output outtarget @directOut;


        }
        simple botVM
        {
            gates:
                //                  input insender @directIn;
                output outbotVM @directOut;
                output outsvmd @directOut;
        }

        simple VM
        {
            gates:
```

Outline: VMCloud (network)

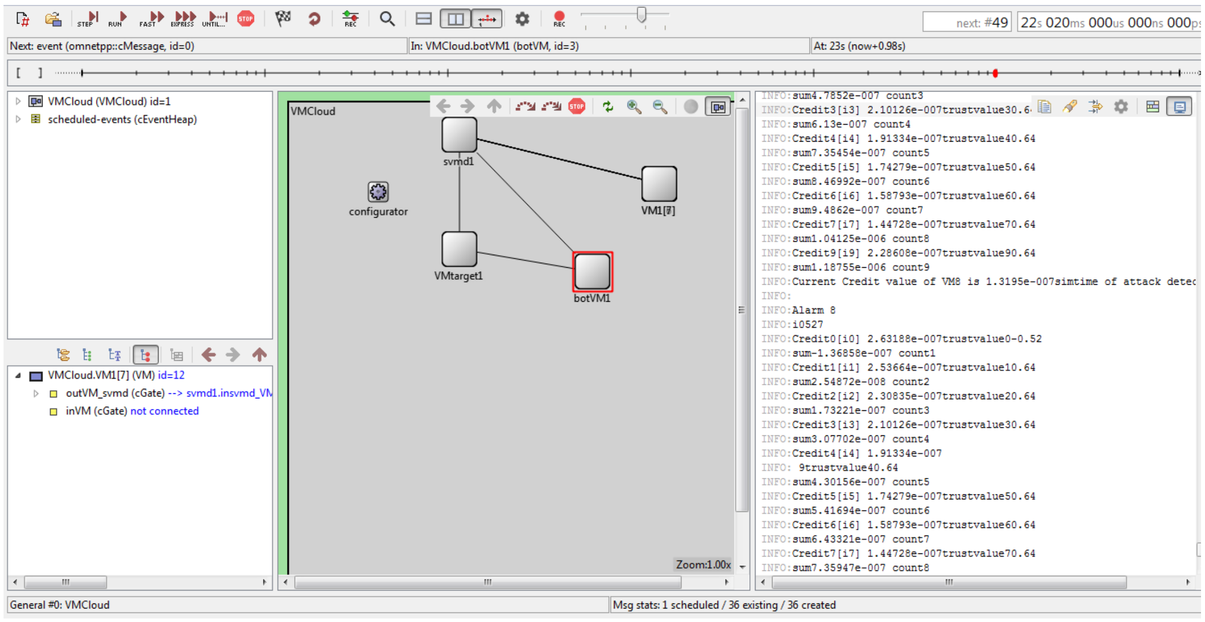
```
<terminated> Final_1_Bot_1_Targets_Improved2 [OMNeT++ Simulation] Final_1_Bot_1_Targets_Improved2.exe (02/09/22 22:21 - )
undisposed object: (inet::svmd_trust) VMCloud.svmd1.pk -- check module destructor
undisposed object: (inet::svmd_trust) VMCloud.svmd1.pk -- check module destructor
undisposed object: (inet::svmd_trust) VMCloud.svmd1.pk -- check module destructor
undisposed object: (inet::svmd_trust) VMCloud.svmd1.pk -- check module destructor

End.
```

Writable | Insert | 41 : 15 : 1203 | 431M of 489M | Searching for Binaries (inet)



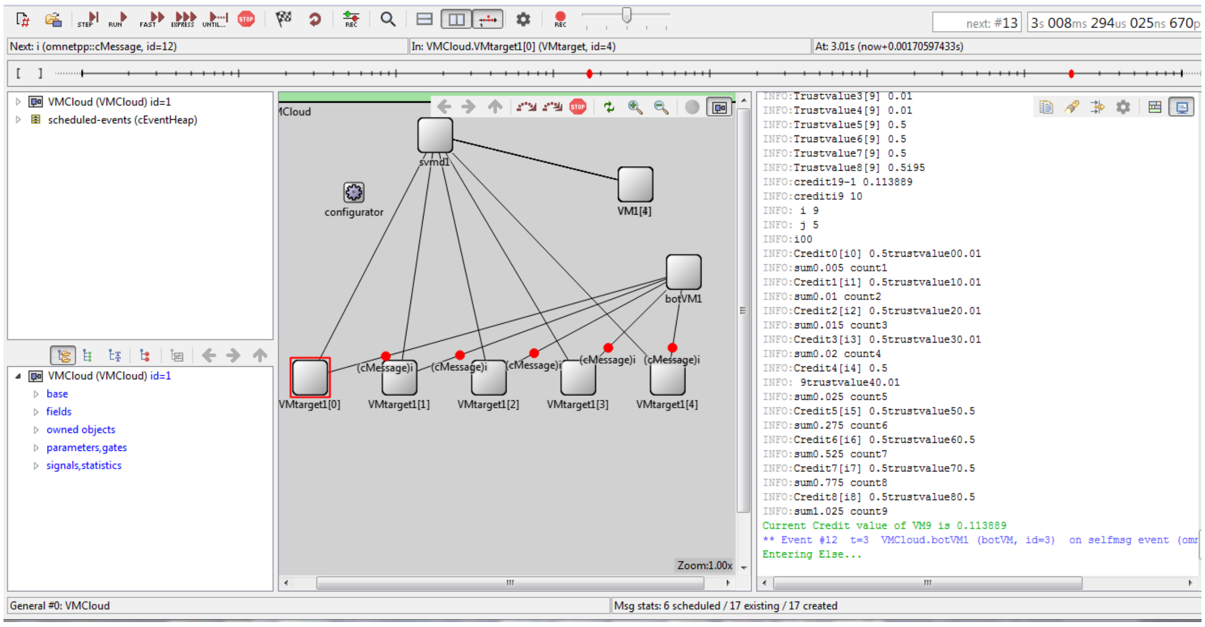
```
INFO:
Alarm 9
** Event #64  t=26.02  VMCloud.svmd1 (svmd, id=13)  on pk (inet::svmd_trust, id=47)
INFO:
INFO:SVMD handle Message
INFO: ArrivalVM 0
INFO:SenderVM 9
Arrival gate 0xaf7f250I am in
** Event #65  t=27  VMCloud.botVM1 (botVM, id=3)  on selfmsg event (omnetpp::cMessage, id=0)
```

Msg stats: 3 scheduled / 50 existing / 50 created

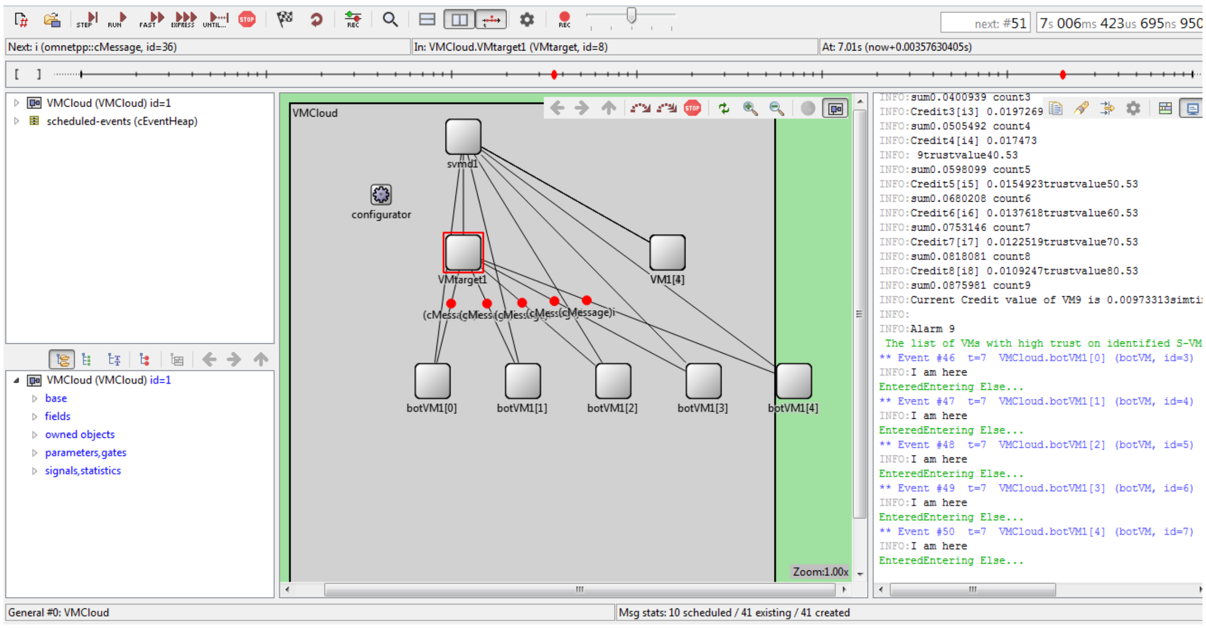

next: #49
22s 020ms 000us 000ns 000ps
Next: event (omnetpp::cMessage, id=0)
In: VMCloud.botVM1 (botVM, id=3)
At: 23s (now+0.98s)
VMCloud (VMCloud) id=1
scheduled-events (cEventHeap)
VMCloud.VM1[7] (VM) id=12
outVM_svmd (cGate) --> svmd1.insvmd_VM
inVM (cGate) not connected
VMCloud
svmd1
configurator
VM1[7]
VMtarget1
botVM1
Zoom:1.00x
General #0: VMCloud

```
INFO:sum4.7852e-007 count3
INFO:Credit3[i3] 2.10126e-007trustvalue30.6
INFO:sum6.13e-007 count4
INFO:Credit4[i4] 1.91334e-007trustvalue40.64
INFO:sum7.35454e-007 count5
INFO:Credit5[i5] 1.74279e-007trustvalue50.64
INFO:sum8.46992e-007 count6
INFO:Credit6[i6] 1.58793e-007trustvalue60.64
INFO:sum9.4862e-007 count7
INFO:Credit7[i7] 1.44728e-007trustvalue70.64
INFO:sum1.04125e-006 count8
INFO:Credit9[i9] 2.28608e-007trustvalue90.64
INFO:sum1.18755e-006 count9
INFO:Current Credit value of VM8 is 1.3195e-007simtime of attack detec
INFO:
INFO:Alarm 8
INFO:i0527
INFO:Credit0[i0] 2.63188e-007trustvalue0-0.52
INFO:sum-1.36858e-007 count1
INFO:Credit1[i1] 2.53664e-007trustvalue10.64
INFO:sum2.54872e-008 count2
INFO:Credit2[i2] 2.30835e-007trustvalue20.64
INFO:sum1.73221e-007 count3
INFO:Credit3[i3] 2.10126e-007trustvalue30.64
INFO:sum3.07702e-007 count4
INFO:Credit4[i4] 1.91334e-007
INFO: 9trustvalue40.64
INFO:sum4.30156e-007 count5
INFO:Credit5[i5] 1.74279e-007trustvalue50.64
INFO:sum5.41694e-007 count6
INFO:Credit6[i6] 1.58793e-007trustvalue60.64
INFO:sum6.43321e-007 count7
INFO:Credit7[i7] 1.44728e-007trustvalue70.64
INFO:sum7.35947e-007 count8
```

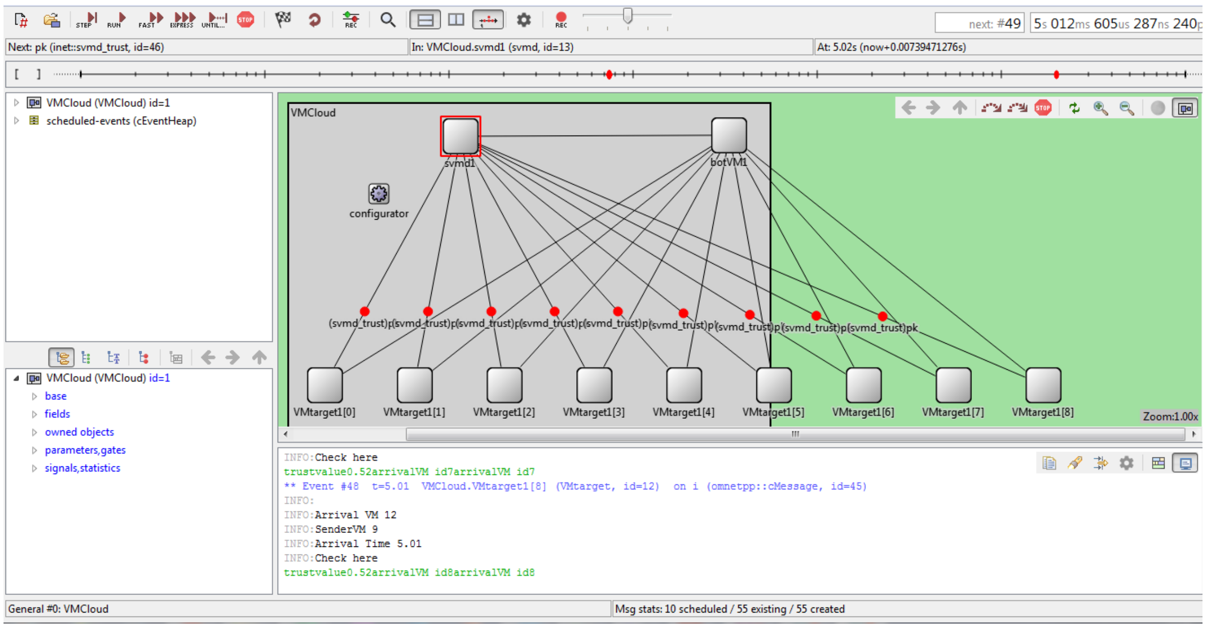
Msg stats: 1 scheduled / 36 existing / 36 created



```
INFO:Trustvalue3[9] 0.01
INFO:Trustvalue4[9] 0.01
INFO:Trustvalue5[9] 0.5
INFO:Trustvalue6[9] 0.5
INFO:Trustvalue7[9] 0.5
INFO:Trustvalue8[9] 0.5i95
INFO:credit19-1 0.113889
INFO:crediti9 10
INFO: i 9
INFO: j 5
INFO:i00
INFO:Credit0[i0] 0.5trustvalue00.01
INFO:sum0.005 count1
INFO:Credit1[i1] 0.5trustvalue10.01
INFO:sum0.01 count2
INFO:Credit2[i2] 0.5trustvalue20.01
INFO:sum0.015 count3
INFO:Credit3[i3] 0.5trustvalue30.01
INFO:sum0.02 count4
INFO:Credit4[i4] 0.5
INFO: 9trustvalue40.01
INFO:sum0.025 count5
INFO:Credit5[i5] 0.5trustvalue50.5
INFO:sum0.275 count6
INFO:Credit6[i6] 0.5trustvalue60.5
INFO:sum0.525 count7
INFO:Credit7[i7] 0.5trustvalue70.5
INFO:sum0.775 count8
INFO:Credit8[i8] 0.5trustvalue80.5
INFO:sum1.025 count9
Current Credit value of VM9 is 0.113889
** Event #12  t=3  VMCloud.botVM1 (botVM, id=3)  on selfmsg event (omr
Entering Else...
```



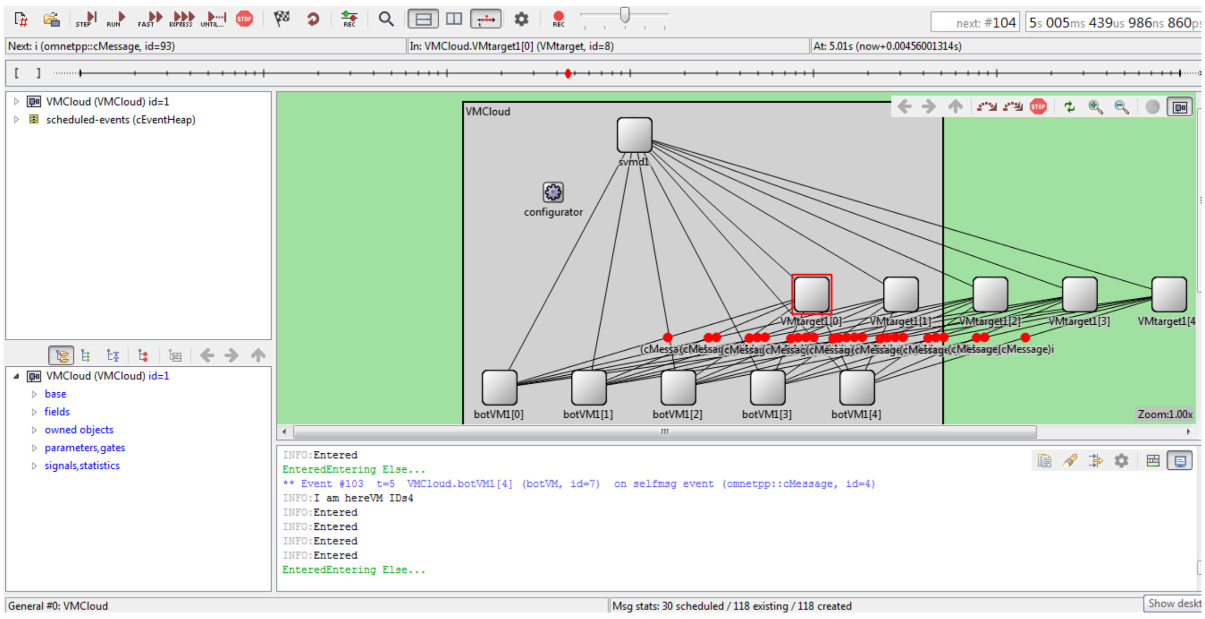
```
INFO:sum0.0400939 count3
INFO:Credit3[i3] 0.0197269
INFO:sum0.0505492 count4
INFO:Credit4[i4] 0.017473
INFO: 9trustvalue40.53
INFO:sum0.0598099 count5
INFO:Credit5[i5] 0.0154923trustvalue50.53
INFO:sum0.0680208 count6
INFO:Credit6[i6] 0.0137618trustvalue60.53
INFO:sum0.0753146 count7
INFO:Credit7[i7] 0.0122519trustvalue70.53
INFO:sum0.0818081 count8
INFO:Credit8[i8] 0.0109247trustvalue80.53
INFO:sum0.0875981 count9
INFO:Current Credit value of VM9 is 0.00973313simti
INFO:
INFO:Alarm 9
 The list of VMs with high trust on identified S-VM
** Event #46  t=7  VMCloud.botVM1[0] (botVM, id=3)
INFO:I am here
EnteredEntering Else...
** Event #47  t=7  VMCloud.botVM1[1] (botVM, id=4)
INFO:I am here
EnteredEntering Else...
** Event #48  t=7  VMCloud.botVM1[2] (botVM, id=5)
INFO:I am here
EnteredEntering Else...
** Event #49  t=7  VMCloud.botVM1[3] (botVM, id=6)
INFO:I am here
EnteredEntering Else...
** Event #50  t=7  VMCloud.botVM1[4] (botVM, id=7)
INFO:I am here
EnteredEntering Else...
```



```
next: #49    5s 012ms 605us 287ns 240p
Next: pk (inet::svmd_trust, id=46)    In: VMCloud.svmd1 (svmd, id=13)    At: 5.02s (now+0.00739471276s)
```

```
INFO:Check here
trustvalue0.52arrivalVM id7arrivalVM id7
** Event #48  t=5.01  VMCloud.VMtarget1[8] (VMtarget, id=12)  on i (omnetpp::cMessage, id=45)
INFO:
INFO:Arrival VM 12
INFO:SenderVM 9
INFO:Arrival Time 5.01
INFO:Check here
trustvalue0.52arrivalVM id8arrivalVM id8
```

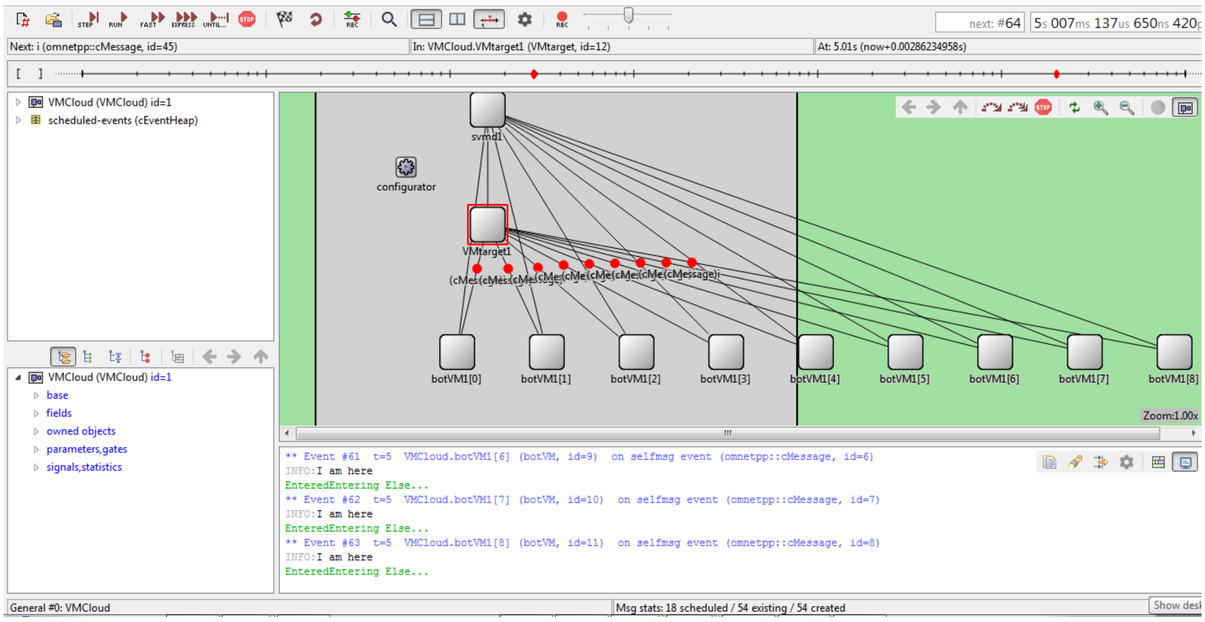
```
General #0: VMCloud    Msg stats: 10 scheduled / 55 existing / 55 created
```



```
next: #104    5s 005ms 439us 986ns 860ps
Next: i (omnetpp::cMessage, id=93)    In: VMCloud.VMtarget1[0] (VMtarget, id=8)    At: 5.01s (now+0.00456001314s)
```

```
INFO:Entered
EnteredEntering Else...
** Event #103  t=5  VMCloud.botVM1[4] (botVM, id=7)  on selfmsg event (omnetpp::cMessage, id=4)
INFO:I am hereVM IDs4
INFO:Entered
INFO:Entered
INFO:Entered
INFO:Entered
EnteredEntering Else...
```

```
General #0: VMCloud    Msg stats: 30 scheduled / 118 existing / 118 created
```



Next: i (omnetpp::cMessage, id=45)
In: VMCloud.VMtarget1 (VMtarget, id=12)
At: 5.01s (now+0.00286234958s)

```
** Event #61  t=5  VMCloud.botVM1[6] (botVM, id=9)  on selfmsg event (omnetpp::cMessage, id=6)
INFO:I am here
EnteredEntering Else...
** Event #62  t=5  VMCloud.botVM1[7] (botVM, id=10)  on selfmsg event (omnetpp::cMessage, id=7)
INFO:I am here
EnteredEntering Else...
** Event #63  t=5  VMCloud.botVM1[8] (botVM, id=11)  on selfmsg event (omnetpp::cMessage, id=8)
INFO:I am here
EnteredEntering Else...
```

General #0: VMCloud
Msg stats: 18 scheduled / 54 existing / 54 created

Bibliography

[1] "Cloud computing." https://csrc.nist.gov/projects/cloud-computing, Feb. 2015. Accessed December 2021.

[2] W. Chai, "Software as a service (saas)." https://www.techtarget.com/searchcloudcomputing/definition/Software-as-a-Service, Feb. 2021. Accessed December 2021.

[3] A. Nair, "Platform as a service (paas)." https://www.alliedmarketresearch.com/platform-as-a-service-market-A06955, Aug. 2021. Accessed December 2021.

[4] S. Shah, "Impact of cloud computing in different industries)." https://www.business2community.com/cloud-computing/impact-of-cloud-computing-in-different-industries-02451160, Jan. 2022. Accessed January 2022.

[5] C. A. Lee, R. B. Bohn, M. Michel, A. Delaitre, B. Stivalet, and P. E. Black, "The nist cloud federation reference architecture 5," *NIST Special Publication*, vol. 500, p. 332, 2020.

[6] "Nist cloud computing program - nccp." https://www.nist.gov/programs-projects/nist-cloud-computing-program-nccp, 2020. Accessed November 2021.

[7] N. Phaphoom, X. Wang, S. Samuel, S. Helmer, and P. Abrahamsson, "A survey study on major technical barriers affecting the decision to adopt cloud services," *Journal of Systems and Software*, vol. 103, pp. 167–181, 2015.

[8] N. Tissir, S. El Kafhali, and N. Aboutabit, "Cloud computing security classifications and taxonomies: a comprehensive study and comparison," in *2020 5th International Conference on Cloud Computing and Artificial Intelligence: Technologies and Applications (CloudTech)*, pp. 1–6, IEEE, 2020.

[9] D. Chen and H. Zhao, "Data security and privacy protection issues in cloud computing," in *2012 International Conference on Computer Science and Electronics Engineering*, vol. 1, pp. 647–651, IEEE, 2012.

[10] R. Charanya and M. Aramudhan, "Survey on access control issues in cloud computing," in *2016 International Conference on Emerging Trends in Engineering, Technology and Science (ICETETS)*, pp. 1–4, IEEE, 2016.

[11] D. Yimam and E. B. Fernandez, “A survey of compliance issues in cloud computing,” *Journal of Internet Services and Applications*, vol. 7, no. 1, pp. 1–12, 2016.

[12] Q. K. Kadhim, R. Yusof, H. S. Mahdi, S. S. A. Al-Shami, and S. R. Selamat, “A review study on cloud computing issues,” in *Journal of Physics: Conference Series*, vol. 1018, p. 012006, IOP Publishing, 2018.

[13] D. Linthicum, “As cloud use grows, so will rate of ddos attacks.” `https://www.infoworld.com/article/2613310/as-cloud-use-grows--so-will-rate-of-ddos-attacks.html`, David Linthicum 2013. Accessed August 2021.

[14] C. S. Alliance, “The notorious nine: Cloud computing top threats in 2013,” *Top Threats Working Group*, 2013.

[15] C. S. Alliance, “Top threats to cloud computing: Egregious eleven deep dive.” `https://cloudsecurityalliance.org/artifacts/top-threats-egregious-11-deep-dive/`, 2020.

[16] C. S. Alliance, “Top threats to cloud computing – pandemic eleven.” `https://cloudsecurityalliance.org/artifacts/top-threats-to-cloud-computing-pandemic-eleven/`, 2022.

[17] “Link11 report reveals ddos attacks reached record high in 2020.” `https://www.link11.com/en/security-wiki/link11-report-reveals-ddos-attacks-reached-record-high-in-2020/`, 2020. Accessed August 2021.

[18] K. Graewe, “Ddos attacks in q3 2021: It infrastructure providers targeted.” `https://www.link11.com/en/blog/threat-landscape/ddos-attacks-in-q3-2021-it-infrastructure-providers-targeted/`, November 2021. Accessed December 2021.

[19] S. Cook, “Ddos attack statistics and facts for 2018-2022.” `https://www.comparitech.com/blog/information-security/ddos-statistics-facts/`, Feb 2022. Accessed Feb 2022.

[20] M. Nawir, A. Amir, N. Yaakob, and O. B. Lynn, “Internet of things (iot): Taxonomy of security attacks,” in *2016 3rd International Conference on Electronic Design (ICED)*, pp. 321–326, IEEE, 2016.

[21] M. Darwish, A. Ouda, and L. F. Capretz, “Cloud-based ddos attacks and defenses,” in *International Conference on Information Society (i-Society 2013)*, pp. 67–71, IEEE, 2013.

[22] S. Dhuria, M. Sachdeva, and G. Kumar, “Recognition and handling of insider and outsider ddos attacks in wsn,” *Indian Journal of Science and Technology*, vol. 10, no. 19, 2017.

[23] W. A. Jansen, “Cloud hooks: Security and privacy issues in cloud computing,” in *2011 44th Hawaii International Conference on System Sciences*, pp. 1–10, IEEE, 2011.

[24] L. M. Vaquero, L. Rodero-Merino, and D. Morán, "Locking the sky: a survey on iaas cloud security," *Computing*, vol. 91, no. 1, pp. 93–118, 2011.

[25] J. Aron, "Botclouds: a cyberattacker's dream," 2011.

[26] S. Li, X. Yun, Z. Hao, X. Cui, and Y. Wang, "A propagation model for social engineering botnets in social networks," in *2011 12th International Conference on Parallel and Distributed Computing, Applications and Technologies*, pp. 423–426, IEEE, 2011.

[27] M. P. Collins, T. J. Shimeall, S. Faber, J. Janies, R. Weaver, M. De Shon, and J. Kadane, "Using uncleanliness to predict future botnet addresses," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pp. 93–104, 2007.

[28] Y.-L. Huang, B. Chen, M.-W. Shih, and C.-Y. Lai, "Security impacts of virtualization on a network testbed," in *2012 IEEE Sixth International Conference on Software Security and Reliability*, pp. 71–77, IEEE, 2012.

[29] K. Munir and S. Palaniappan, "Framework for secure cloud computing," *Advanced International Journal on Cloud Computing: Services and Architecture (IJCCSA)*, vol. 3, no. 2, pp. 21–35, 2013.

[30] L. Zhang, S. Yu, D. Wu, and P. Watters, "A survey on latest botnet attack and defense," in *2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 53–60, IEEE, 2011.

[31] D. Seenivasan and K. Shanthi, "Categories of botnet: a survey," *Int. J. Comput. Control Quantum Inf. Eng*, vol. 8, no. 9, pp. 1589–1592, 2014.

[32] A. Varga, "Omnet++," in *Modeling and tools for network simulation*, pp. 35–59, Springer, 2010.

[33] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation computer systems*, vol. 25, no. 6, pp. 599–616, 2009.

[34] H. Wu, Y. Ding, C. Winer, and L. Yao, "Network security for virtual machine in cloud computing," in *5th International conference on computer sciences and convergence information technology*, pp. 18–21, IEEE, 2010.

[35] M. B. Rashid, N. Islam, A. A. M. Sabuj, S. Waheed, and M. B. A. Miah, "Randomly encrypted key generation algorithm against side channel attack in cloud computing," in *2015 International Conference on Electrical Engineering and Information Communication Technology (ICEEICT)*, pp. 1–5, IEEE, 2015.

[36] J. Reed, "Physical servers vs. virtual machines: Key differences and similarities." `https://www.nakivo.com/blog/physical-servers-vs-virtual-machines-key-differences-similarities/`, 2022.

[37] VmWare, "What is a hypervisor?." `https://www.vmware.com/topics/glossary/content/hypervisor.html`.

[38] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Virtual infrastructure management in private and hybrid clouds," *IEEE Internet computing*, vol. 13, no. 5, pp. 14–22, 2009.

[39] I. C. Education, "Containerization." `https://www.ibm.com/uk-en/cloud/learn/containerization`, 2019.

[40] D. Merkel *et al.*, "Docker: lightweight linux containers for consistent development and deployment," *Linux j*, vol. 239, no. 2, p. 2, 2014.

[41] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE cloud computing*, vol. 1, no. 3, pp. 81–84, 2014.

[42] K. Clark, M. Warnier, and F. M. Brazer, "The future of cloud-based botnets?," in *CLOSER 2011–International Conference on Cloud Computing and Services Science*, pp. 597–603, SciTePress–Science and Technology Publications Noordwijkerhout, 2011.

[43] N. Raghava, D. Sahgal, and S. Chandna, "Classification of botnet detection based on botnet architechture," in *2012 International Conference on Communication Systems and Network Technologies*, pp. 569–572, IEEE, 2012.

[44] J. Aron, "Botclouds: a cyberattacker's dream." `https://www.newscientist.com/article/mg21028175-500-botclouds-a-cyberattackers-dream/`, June 2011. Accessed Feb 2021.

[45] Y. Han, J. Chan, T. Alpcan, and C. Leckie, "Virtual machine allocation policies against co-resident attacks in cloud computing," in *2014 IEEE International Conference on Communications (ICC)*, pp. 786–792, IEEE, 2014.

[46] T. Ristenpart, E. Tromer, H. Shacham, and S. 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*, pp. 199–212, 2009.

[47] C. Y. Cho, J. Caballero, C. Grier, V. Paxson, and D. Song, "Insights from the inside: A view of botnet management from infiltration.," *LEET*, vol. 10, pp. 1–1, 2010.

[48] M. Bazm, R. Khatoun, Y. Begriche, L. Khoukhi, X. Chen, and A. Serhrouchni, "Malicious virtual machines detection through a clustering approach," in *2015 International Conference on Cloud Technologies and Applications (CloudTech)*, pp. 1–8, IEEE, 2015.

[49] A. S. Ibrahim, J. Hamlyn-Harris, J. Grundy, and M. Almorsy, "Cloudsec: a security monitoring appliance for virtual machines in the iaas cloud model," in *2011 5th International Conference on Network and System Security*, pp. 113–120, IEEE, 2011.

[50] C. Kalt, "Internet relay chat: Architecture," 2000.

[51] B. AsSadhan, A. Bashaiwth, J. Al-Muhtadi, and S. Alshebeili, "Analysis of p2p, irc and http traffic for botnets detection," *Peer-to-Peer Networking and Applications*, vol. 11, no. 5, pp. 848–861, 2018.

[52] J. Liu, Y. Xiao, K. Ghaboosi, H. Deng, and J. Zhang, "Botnet: classification, attacks, detection, tracing, and preventive measures," *EURASIP journal on wireless communications and networking*, vol. 2009, pp. 1–11, 2009.

[53] A. Karim, R. B. Salleh, M. Shiraz, S. A. A. Shah, I. Awan, and N. B. Anuar, "Botnet detection techniques: review, future trends, and issues," *Journal of Zhejiang University SCIENCE C*, vol. 15, no. 11, pp. 943–983, 2014.

[54] P. Wang, S. Sparks, and C. C. Zou, "An advanced hybrid peer-to-peer botnet," *IEEE Transactions on Dependable and Secure Computing*, vol. 7, no. 2, pp. 113–127, 2008.

[55] B. Wang, Z. Li, D. Li, F. Liu, and H. Chen, "Modeling connections behavior for web-based bots detection," in *2010 2nd International Conference on E-business and Information System Security*, pp. 1–4, IEEE, 2010.

[56] D. Zhao, I. Traore, A. Ghorbani, B. Sayed, S. Saad, and W. Lu, "Peer to peer botnet detection based on flow intervals," in *IFIP International Information Security Conference*, pp. 87–102, Springer, 2012.

[57] L. Liu, S. Chen, G. Yan, and Z. Zhang, "Bottracer: Execution-based bot-like malware detection," in *International Conference on Information Security*, pp. 97–113, Springer, 2008.

[58] F. C. Freiling, T. Holz, and G. Wicherski, "Botnet tracking: Exploring a root-cause methodology to prevent distributed denial-of-service attacks," in *European Symposium on Research in Computer Security*, pp. 319–335, Springer, 2005.

[59] V.-H. Pham and M. Dacier, "Honeypot trace forensics: The observation viewpoint matters," *Future Generation Computer Systems*, vol. 27, no. 5, pp. 539–546, 2011.

[60] L. Spitzner, "Honeypots: Catching the insider threat," in *19th Annual Computer Security Applications Conference, 2003. Proceedings.*, pp. 170–179, IEEE, 2003.

[61] R. G. Bace, P. Mell, *et al.*, "Intrusion detection systems," 2001.

[62] J. Goebel and T. Holz, "Rishi: Identify bot contaminated hosts by irc nickname evaluation.," *HotBots*, vol. 7, no. 8-8, p. 192, 2007.

[63] Y. Kugisaki, Y. Kasahara, Y. Hori, and K. Sakurai, "Bot detection based on traffic analysis," in *The 2007 International Conference on Intelligent Pervasive Computing (IPC 2007)*, pp. 303–306, IEEE, 2007.

[64] J. R. Binkley and S. Singh, "An algorithm for anomaly-based botnet detection.," *SRUTI*, vol. 6, pp. 7–7, 2006.

[65] E. Stinson and J. C. Mitchell, "Characterizing botsâĂŹ remote control behavior," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 89–108, Springer, 2007.

[66] W. T. Strayer, D. Lapsely, R. Walsh, and C. Livadas, "Botnet detection based on network behavior," in *Botnet detection*, pp. 1–24, Springer, 2008.

[67] S. Saad, I. Traore, A. Ghorbani, B. Sayed, D. Zhao, W. Lu, J. Felix, and P. Hakimian, "Detecting p2p botnets through network behavior analysis and machine learning," in *2011 Ninth annual international conference on privacy, security and trust*, pp. 174–180, IEEE, 2011.

[68] D. J. Dean, H. Nguyen, and X. Gu, "Ubl: Unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems," in *Proceedings of the 9th international conference on Autonomic computing*, pp. 191–200, 2012.

[69] G. Gu, P. A. Porras, V. Yegneswaran, M. W. Fong, and W. Lee, "Bothunter: Detecting malware infection through ids-driven dialog correlation.," in *USENIX Security Symposium*, vol. 7, pp. 1–16, 2007.

[70] G. Gu, J. Zhang, and W. Lee, "Botsniffer: Detecting botnet command and control channels in network traffic," 2008.

[71] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection," 2008.

[72] J. Zhang, R. Perdisci, W. Lee, U. Sarfraz, and X. Luo, "Detecting stealthy p2p botnets using statistical traffic fingerprints," in *2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN)*, pp. 121–132, IEEE, 2011.

[73] J. Francois, S. Wang, W. Bronzi, R. State, and T. Engel, "Botcloud: Detecting botnets using mapreduce," in *2011 IEEE International Workshop on Information Forensics and Security*, pp. 1–6, IEEE, 2011.

[74] B. Li, W. Niu, K. Xu, C. Zhang, and P. Zhang, "You canâĂŹt hide: a novel methodology to defend ddos attack based on botcloud," in *International Conference on Applications and Techniques in Information Security*, pp. 203–214, Springer, 2015.

[75] S.-W. Hsiao, Y.-N. Chen, Y. S. Sun, and M. C. Chen, "A cooperative botnet profiling and detection in virtualized environment," in *2013 IEEE Conference on Communications and Network Security (CNS)*, pp. 154–162, IEEE, 2013.

[76] B. Hammi, S. Zeadally, and R. Khatoun, "An empirical investigation of botnet as a service for cyberattacks," *Transactions on emerging telecommunications technologies*, vol. 30, no. 3, p. e3537, 2019.

[77] V. R. Kebande and H. S. Venter, "A cognitive approach for botnet detection using artificial immune system in the cloud," in *2014 Third International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec)*, pp. 52–57, IEEE, 2014.

[78] W. Lin and D. Lee, "Traceback attacks in cloud–pebbletrace botnet," in *2012 32nd International Conference on Distributed Computing Systems Workshops*, pp. 417–426, IEEE, 2012.

[79] H. Badis, G. Doyen, and R. Khatoun, "Toward a source detection of botclouds: a pca-based approach," in *IFIP International Conference on Autonomous Infrastructure, Management and Security*, pp. 105–117, Springer, 2014.

[80] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1-3, pp. 37–52, 1987.

[81] M. R. Memarian, M. Conti, and V. Leppänen, "Eyecloud: A botcloud detection system," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1, pp. 1067–1072, IEEE, 2015.

[82] H. Badis, G. Doyen, and R. Khatoun, "A collaborative approach for a source based detection of botclouds," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 906–909, IEEE, 2015.

[83] R. Cogranne, G. Doyen, N. Ghadban, and B. Hammi, "Detecting botclouds at large scale: A decentralized and robust detection method for multi-tenant virtualized environments," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 68–82, 2017.

[84] S. Bolton, "Crime prevention in the community: The case of neighbourhood watch," 2006.

[85] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[86] C. Krupitzer, F. M. Roth, S. VanSyckel, G. Schiele, and C. Becker, "A survey on engineering approaches for self-adaptive systems," *Pervasive and Mobile Computing*, vol. 17, pp. 184–206, 2015.

[87] A. L. Shaw, B. Bordbar, J. Saxon, K. Harrison, and C. I. Dalton, "Forensic virtual machines: dynamic defence in the cloud via introspection," in *2014 IEEE International Conference on Cloud Engineering*, pp. 303–310, IEEE, 2014.

[88] M. Achemlal, J.-C. Pailles, and C. Gaber, "Building trust in virtualized networks," in *2010 2nd International Conference on Evolving Internet*, pp. 113–118, IEEE, 2010.

[89] L. E. Li and T. Woo, "Vsite: A scalable and secure architecture for seamless l2 enterprise extension in the cloud," in *2010 6th IEEE Workshop on Secure Network Protocols*, pp. 31–36, IEEE, 2010.

[90] W. Dai, T. P. Parker, H. Jin, and S. Xu, "Enhancing data trustworthiness via assured digital signing," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 6, pp. 838–851, 2012.

[91] C.-J. Chung, P. Khatkar, T. Xing, J. Lee, and D. Huang, "Nice: Network intrusion detection and countermeasure selection in virtual network systems," *IEEE transactions on dependable and secure computing*, vol. 10, no. 4, pp. 198–211, 2013.

[92] D. S. Burke, J. F. Brundage, R. R. Redfield, J. J. Damato, C. A. Schable, P. Putman, R. Visintine, and H. I. Kim, "Measurement of the false positive rate in a screening program for human immunodeficiency virus infections," *New England Journal of Medicine*, vol. 319, no. 15, pp. 961–964, 1988.

[93] D. R. Williams and P. Rast, "Back to the basics: Rethinking partial correlation network methodology," *British Journal of Mathematical and Statistical Psychology*, vol. 73, no. 2, pp. 187–212, 2020.

[94] R. Riffenburgh, "Chapter 10 - risks, odds, and roc curves," in *Statistics in Medicine (Third Edition)* (R. Riffenburgh, ed.), pp. 203–219, San Diego: Academic Press, third edition ed., 2012.

[95] S. Verma and J. Rubin, "Fairness definitions explained," in *2018 ieee/acm international workshop on software fairness (fairware)*, pp. 1–7, IEEE, 2018.

[96] B. Matthews, "Comparison of the predicted and observed secondary structure of t4 phage lysozime," *Biochim. Biophys. Acta*, vol. 405, pp. 442–451, 11 1974.

[97] A. Hayes, "Scalability." `https://www.investopedia.com/terms/s/scalability.asp#:~:text=Scalability%20describes%20a%20system's%20capability,can%20quickly%20ramp%20up%20production.`, Nov 2020. Accessed Feb 2021.

[98] L. Litty, *Hypervisor-based intrusion detection.* University of Toronto, 2005.

[99] J. Nikolai and Y. Wang, "Hypervisor-based cloud intrusion detection system," in *2014 International Conference on Computing, Networking and Communications (ICNC)*, pp. 989–993, IEEE, 2014.

[100] E. Bauman, G. Ayoade, and Z. Lin, "A survey on hypervisor-based monitoring: approaches, applications, and evolutions," *ACM Computing Surveys (CSUR)*, vol. 48, no. 1, pp. 1–33, 2015.

[101] F. Sabahi, "Secure virtualization for cloud environment using hypervisor-based technology," *International Journal of Machine Learning and Computing*, vol. 2, no. 1, p. 39, 2012.

[102] A. M. Azab, P. Ning, E. C. Sezer, and X. Zhang, "Hima: A hypervisor-based integrity measurement agent," in *2009 Annual Computer Security Applications Conference*, pp. 461–470, IEEE, 2009.

[103] E. Nordstrom, P. Gunningberg, and H. Lundgren, "A testbed and methodology for experimental evaluation of wireless mobile ad hoc networks," in *First international conference on testbeds and research infrastructures for the development of networks and communities*, pp. 100–109, IEEE, 2005.

[104] Q. Duan, "Cloud service performance evaluation: status, challenges, and opportunities–a survey from the system modeling perspective," *Digital Communications and Networks*, vol. 3, no. 2, pp. 101–111, 2017.

[105] K. Bahwaireth, L. Tawalbeh, E. Benkhelifa, Y. Jararweh, and M. A. Tawalbeh, "Experimental comparison of simulation tools for efficient cloud and mobile cloud computing applications," *EURASIP Journal on Information Security*, vol. 2016, no. 1, pp. 1–14, 2016.

[106] J. Byrne, S. Svorobej, K. M. Giannoutakis, D. Tzovaras, P. J. Byrne, P.-O. Östberg, A. Gourinovitch, and T. Lynn, "A review of cloud computing simulation platforms and related environments," in *International Conference on Cloud Computing and Services Science*, vol. 2, pp. 679–691, SciTePress, 2017.

[107] R Core Team, *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2013.

[108] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, “Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,” *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.

[109] T. Issariyakul and E. Hossain, “Introduction to network simulator 2 (ns2),” in *Introduction to network simulator NS2*, pp. 1–18, Springer, 2009.

[110] A. Varga and R. Hornig, “An overview of the omnet++ simulation environment,” in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, pp. 1–10, 2008.

[111] A. Nayyar, “The best open source cloud computing simulators.” `https://www.opensourceforu.com/2016/11/best-open-source-cloud-computing-simulators/`, Nov. 2016. Accessed 2022.

[112] M. Mandeep and S. Bhatia, “A critical review & analysis of cloud computing simulators,” *IJLTET*, pp. 29–36, 2016.

[113] J.-K. Jung, N.-U. Kim, S.-M. Jung, and T.-M. Chung, “Improved cloudsim for simulating qos-based cloud services,” in *Ubiquitous Information Technologies and Applications*, pp. 537–545, Springer, 2013.

[114] A. Hamidian, U. Korner, and A. Nilsson, “A study of internet connectivity for mobile ad hoc networks in ns 2,” *Department of Communication Systems, Lund Institute of Technology, Lund University*, 2003.

[115] “What is omnet++.” `https://omnetpp.org/intro`. Accessed Feb 2019.

[116] F. Yu and R. Jain, “A survey of wireless sensor network simulation tools,” *Washington University in St. Louis, Department of Science and Engineering*, 2011.

[117] L. Mészáros, A. Varga, and M. Kirsche, “Inet framework,” in *Recent Advances in Network Simulation*, pp. 55–106, Springer, 2019.

[118] “Inet-4.2.0 release available.” `https://inet.omnetpp.org/2020-01-08-INET-4.2.0-released.html`, Jan 2020. Accessed Feb 2020.

[119] “Omnet++ older versions.” `https://omnetpp.org/download/old`, Jan 2020. Accessed Feb 2020.

[120] G. E. Box and N. R. Draper, *Empirical model-building and response surfaces.* John Wiley & Sons, 1987.

[121] L. Elefteriadou, “Mathematical and empirical models,” in *An Introduction to Traffic Flow Theory*, pp. 129–135, Springer, 2014.

[122] VMware, “Vmware virtual networking concepts.” `https://www.vmware.com/techpapers/2007/vmware-virtual-networking-concepts-997.html`, 2007. Accessed June 2021.

[123] J. Langone, S. Key, U. S. Alder, *et al.*, "Whitepaper: Vmware esx server 3: 802.1 q vlan solutions," 2006.

[124] Y. Wang and J. Vassileva, "Trust and reputation model in peer-to-peer networks," in *Proceedings Third International Conference on Peer-to-Peer Computing (P2P2003)*, pp. 150–157, IEEE, 2003.

[125] B. Yu, M. P. Singh, and K. Sycara, "Developing trust in large-scale peer-to-peer systems," in *IEEE First Symposium onMulti-Agent Security and Survivability, 2004*, pp. 1–10, IEEE, 2004.

[126] S. P. Marsh, "Formalising trust as a computational concept," 1994.

[127] C. Castelfranchi, R. Falcone, and G. Pezzulo, "Trust in information sources as a source for trust: a fuzzy approach," in *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pp. 89–96, 2003.

[128] C. M. Jonker and J. Treur, "Formal analysis of models for the dynamics of trust based on experiences," in *European workshop on modelling autonomous agents in a multi-agent world*, pp. 221–231, Springer, 1999.

[129] M. Nojoumian and T. C. Lethbridge, "A new approach for the trust calculation in social networks," in *International Conference on E-Business and Telecommunication Networks*, pp. 64–77, Springer, 2006.

[130] Y. Sun, W. Yu, Z. Han, and K. R. Liu, "Trust modeling and evaluation in ad hoc networks," in *GLOBECOM'05. IEEE Global Telecommunications Conference, 2005.*, vol. 3, pp. 6–pp, IEEE, 2005.

[131] T. Williams, "The two-dimensionality of project risk," *International Journal of Project Management*, vol. 14, no. 3, pp. 185–186, 1996.

[132] Y. Wang, K.-J. Lin, D. S. Wong, and V. Varadharajan, "Trust management towards service-oriented applications," *Service Oriented Computing and Applications*, vol. 3, no. 2, pp. 129–146, 2009.

[133] A. Wang, W. Chang, S. Chen, and A. Mohaisen, "Delving into internet ddos attacks by botnets: characterization and analysis," *IEEE/ACM Transactions on Networking*, vol. 26, no. 6, pp. 2843–2855, 2018.

[134] J. Guo, R. Chen, and J. J. Tsai, "A mobile cloud hierarchical trust management protocol for iot systems," in *2017 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pp. 125–130, IEEE, 2017.

[135] A. Schaub, R. Bazin, O. Hasan, and L. Brunie, "A trustless privacy-preserving reputation system," in *IFIP International Conference on ICT Systems Security and Privacy Protection*, pp. 398–411, Springer, 2016.

[136] T. Garfinkel, M. Rosenblum, *et al.*, "A virtual machine introspection based architecture for intrusion detection.," in *Ndss*, pp. 191–206, Citeseer, 2003.

[137] T. Win, H. Tianfield, Q. Mair, T. AL Said, and O. Rana, "Virtual machine introspection," vol. 2014, pp. 405–410, 09 2014.

[138] B. D. Payne, "Simplifying virtual machine introspection using libvmi.," tech. rep., Sandia National Laboratories (SNL), Albuquerque, NM, and Livermore, CA âĂę, 2012.

[139] J. Pfoh, C. Schneider, and C. Eckert, "A formal model for virtual machine introspection," in *Proceedings of the 1st ACM workshop on Virtual machine security*, pp. 1–10, 2009.

[140] K. Kourai and S. Chiba, "Hyperspector: virtual distributed monitoring environments for secure intrusion detection," in *Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*, pp. 197–207, 2005.

[141] K. Harrison, B. Bordbar, S. T. Ali, C. I. Dalton, and A. Norman, "A framework for detecting malware in cloud by identifying symptoms," in *2012 IEEE 16th International Enterprise Distributed Object Computing Conference*, pp. 164–172, IEEE, 2012.

[142] P. K. Chouhan, M. Hagan, G. McWilliams, and S. Sezer, "Network based malware detection within virtualised environments," in *European Conference on Parallel Processing*, pp. 335–346, Springer, 2014.

[143] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *ACM SIGOPS operating systems review*, vol. 37, no. 5, pp. 164–177, 2003.

[144] H. Xiong, Z. Liu, W. Xu, and S. Jiao, "Libvmi: a library for bridging the semantic gap between guest os and vmm," in *2012 IEEE 12th International Conference on Computer and Information Technology*, pp. 549–556, IEEE, 2012.

[145] Oracle, "Introduction to oracle vm." `https://docs.oracle.com/cd/E35328_01/E35332/html/index.html.`, 2014.

[146] N. Falliere and E. Chien, "Zeus: King of the bots," *Symantec Security Response (http://bit. ly/3VyFV1)*, 2009.