Efficient Techniques for Intrusion Detection in Cloud Environment

A

Ph.D. Thesis

Submitted in partial fulfillment of the requirements
for the award of the degree
of

DOCTOR OF PHILOSOPHY

in

Department of Computer Science and Engineering

by

PREETI MISHRA
(2013RCP9523)

Under the supervision of

Dr. Pilli Emmanuel Shubhakar
Assistant Professor
Department of Computer Science and Engineering
Malaviya National Institute of Technology, Jaipur, India

Prof. Vijay Varadharajan
Global Innovation Chair Professor in Cyber Security
Faculty of Engineering and Built Environment
The University of Newcastle, Callaghan, Australia

Department of Computer Science and Engineering
Malaviya National Institute of Technology, Jaipur, India

December, 2017

Declaration

I, **Preeti Mishra**, declare that this thesis titled, **"Efficient Techniques for Intrusion Detection in Cloud Environment"** and the work presented in it are my own work. I confirm that:

- This work was done wholly or mainly while in candidature for a Ph.D. degree at Malaviya National Institute of Technology, Jaipur, India.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at MNIT or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this Dissertation is entirely my own work.
- I have acknowledged all main sources of help.

Signed: ______________________________

Date: ______________________________

Certificate

This is to certify that the thesis entitled **"Efficient Techniques for Intrusion Detection in Cloud Environment"** is being submitted by **Ms. Preeti Mishra** (Enroll. No. 2013RCP9523) in partial fulfillment of the requirements for the award of the Degree of **Doctor of Philosophy** in the Department of Computer Science & Engineering, Malaviya National Institute of Technology Jaipur, Rajasthan, India. It is a record of the original research work carried out by her under my supervision. The thesis has reached the standards fulfilling the requirements of the regulations relating to the degree. The results contained in this thesis have not been submitted in part or full to any other university or institute for the award of any degree or diploma.

Dr. Pilli Emmanuel Shubhakar
Assistant Professor
Department of Computer Science & Engineering
Malaviya National Institute of Technology Jaipur, India

Date:
Place:

Certificate

This is to certify that the thesis entitled **"Efficient Techniques for Intrusion Detection in Cloud Environment"** is being submitted by **Ms. Preeti Mishra** (Enroll. No. 2013RCP9523) in partial fulfillment of the requirements for the award of the Degree of **Doctor of Philosophy** in the Department of Computer Science & Engineering, Malaviya National Institute of Technology Jaipur, Rajasthan, India. It is a record of the original research work carried out by her under my supervision. The thesis has reached the standards fulfilling the requirements of the regulations relating to the degree. The results contained in this thesis have not been submitted in part or full to any other university or institute for the award of any degree or diploma.

Prof. Vijay Varadharajan
Global Innovation Chair Professor in Cyber Security
Faculty of Engineering and Built Environment
The University of Newcastle, Callaghan, Australia

Date:
Place:

Acknowledgments

First of all, I would like to thank GOD, the Almighty, for his showers of blessings throughout my research work and also enabling me to reach to the milestone of writing this last note in my research work. I wish to express my heartfelt gratitude for the encouragement and sincere guidance that my supervisors **Dr. Pilli Emmanuel Shubhakar** and **Prof. Vijay Varadharajan**, have given me throughout my research work. I consider myself to be very fortunate to be associated with them. They have always been the pillars of support. Their advice-always encouraging and their ideas-always useful has helped me to identify the correct objectives. I can not envision having better mentors for my Ph. D. study.

The foresight and optimism shown by Dr. Pilli has inspired me to undertake research in the emerging field of cloud security. He always ensured that I never lacked any of the resources to carry out my research. When there were difficulties, Dr. Pilli's insistence and encouragement 'to keep on going with full motivation till there is a breakthrough' helped me to gain more confidence. He has spent a voluminous of time which often helped me in understanding the problems and and improving my presentation skills.

I benefited from the fresh inputs which Prof. Varadharajan had given from his vast research experience and exposure that put me on the right track when I was beginning to go to tangential. He has helped me in understanding the research issues and improving my reading & writing skills. He has helped me set milestones and guided me to make steady progress. The suggestions and critical reviews given by him has given the right directions for my work.

I thanks the members of my doctoral research committee: Prof. M. C. Govil, Dr. Girdhari Singh, Dr. Arka Prokash Mazumdar, and Dr. C. Periasamy for their patience during discussions and evaluations. Their inquisitiveness provided a new perspective to the problem. I would also like to give special thanks to Prof. R. C. Joshi for his guidance, continuous motivation and encouragement for pursuing the research work during the time when I felt very low. He also gave his valuable time to review thesis and providing critical suggestions for improving it. My special thanks to Dr. Pilli' wife, Mrs. Phoebe Vanmathy and their wonderful daughter Pramiti for supporting me as a family during the research work. I also express thanks to Mr. Jaipal and Mr. Ravi for providing help to fix the networking related issues during research. I express my thanks to all the Ph.D. scholars and colleagues from Computer Engineering Department for their friendship and support.

I cannot forget the Internship period at Macquarie University Syndey during my research work. I am very much thankful to Prof. Varadharajan for his supervision during Internship. The research environment has helped me a lot for doing my work with full efficiency during my initial learning phase. I would also give a special thanks to Mrs. Meera Varadharajan for her care and support during Internship & motivating me during my research work. I express my deep gratitude to Dr. Udaya Kiran Tupakula for his guidance, encouragement and providing the critical reviews for the research work. He has spent time for reviewing the work and providing the critical suggestions which has often helped me to come up with solutions. The technical talks delivered by Advanced Cyber Security Research Centre (ACSRC) group research members and professors have helped me in exploring the security solutions. I am very much thankful to Kallol Krishna Karmakar, Shalini Tirunagari and other friends, I made in Syndey for always being there for helping me and assisting for adjusting in the new environment.

I am thankful to Dr. Dhilung Kirat from University of California, Santa Barbara for providing us the evasive attack dataset of malicious programs upon request. I also express thank to Dr. Tamas K Lengyel from University of Connecticut, US for his help during understanding the virtualization concepts.

My specially gratitude goes to Jharna, Nisha, Ritu, Sonal and Tanushree for their wonderful friendship and refreshing company. I thank you all for the moral support and fun time during hostel life. I also thank to Vishal, Manisha, Ankit, Ashish, Sunita, Sanchika, Surendra, Kunal and Mr. Suraj for their direct and indirect help. Thanks to my OpenStack team Priya, Shailendra, Rahul and Kapil for working as a team during the Cloud set up at MNIT to understand cloud architecture. I would like to thank Mr. Anand T. S. (Solution Support Engineer in Cisco Systems Pvt Limited and Former Application Administrator: Citrix Cloud Platform at IPsoft, Inc.) for helping us in fixing technical issues. A company of good friends is must needed and they will be remembered always.

My thesis would not have been possible without extensive support from my husband Deepak Joshi. He ensured that I will focus on my work and shouldered many of my responsibilities. I promise him that I will compensate for all the lost fun. I am also very much thankful for his love, understanding and prayers during my research work. I am also indebted to my parents: Mr. Diwakar Mishra and Mrs. Sarla Mishra without whose support this endeavor would be impossible. They always supported me in my career, making me smile by allowing me to do the things which I wanted to do and pursue my dreams. I also thank my brother Mohit Mishra for his motivation and encouragement.

I am very grateful to my in-laws Mr. P. C. Joshi and Mrs. Kamla Joshi for their support, motivation and encouragement during the research work. My father and father-in law have always told me that they are proud of me, that I am the only one in the family pursuing Ph. D. which has kept me motivated.

Finally, I would like to thank all my friends and family members for their direct or indirect support during my Ph.D.

Signed: ____________________________

Date: ____________________________

Dedications

"Commit your work to the God, then it will succeed".

The Thesis is Dedicated to Beloved God, my Husband and my Parents

Abstract

Attack incidents are increasing day by day with the evolution of cloud computing services. Most of the companies are changing the way they operate and moving towards cloud based services. Security in such a complex technological environment is very important for providing assurance to cloud customers. The importance of well-organized architecture and security roles have become greater with the popularity of cloud computing. Some of the researchers working in the field of cloud security have proposed intrusion detection systems (IDS) as a defensive approach. The existing frameworks that deploy IDS at individual Tenant Virtual Machine (TVM) are prone to IDS subversion attacks. They are less efficient in detecting malware activities. Moreover, the TVM-layer security solutions cannot be directly applied at the Virtual Machine Monitor (VMM)-layer because of the semantic gap problem at the hypervisor. Semantic gap refers to interpreting the low-level information of a guest OS into a high-level semantics. VM introspection (VMI) is one of the virtualization-specific approaches that provides possible ways to obtain the high-level view of TVM at hypervisor. However, not enough work has been done in this direction to provide VMI-based security solutions for cloud. The existing VMM-layer solutions do not provide a complete solution to detect both basic and evasive malware attacks in cloud. On the other hand, some of the cloud security frameworks are designed to detect network intrusions only. Most of them apply signature-matching technique as core detection technique, making them prone to signature-manipulation attacks. Some of them apply machine learning at VMM-layer. However, they are not integrated with the available network introspection functions at VMM-layer. As the evaluation of above existing solutions are based on a very older KDD99 dataset, estimation of the real performance of the system is difficult to assess. Moreover, the security at Network-layer is as important as the security at VMM or TVM-layer.

We propose a robust, efficient and VMI-based distributed intrusion detection framework, called *CloudHedge* which provides the detection of both malware and network intrusions at various security-critical positions, covering all three layers in cloud, i.e. TVM-layer, VMM-layer and Network-layer. It provides three lines of defense by providing efficient intrusion detection capability in form of three sub-IDS instances. The sub-IDS instances are called as Malicious System Call Sequence Detection (deployed at TVM-layer), VM

Introspection based Malware Detection (deployed at VMM-layer) and Malicious Network Packet Detection (deployed at Network and VMM-layer). The motivation for the design of CloudHedge is to overcome the limitations associated with the existing security architectures. A centralized IDS becomes a bottleneck when there is an increase in the number of TVMs in the cloud host and when the security tool uses centralized resources. A distributed IDS that deploys the same security solution at all layers in cloud becomes less efficient because of the limitations associated with different layers. Each of the sub-IDS instance of CloudHedge monitors at specific security-layer(s) which are centrally controlled and configured by Cloud Service Provider (CSP). This enables the CSP to assign the specific security solution based on the tenants demands.

The first-line of defense is provided by Malicious System Call Sequence Detection (**MSCSD**), deployed at TVM-layer in cloud. MSCSD is applicable to traditional physical hosts and TVMs of virtualization/cloud environments. MSCSD analyses the run time behavior of the monitored programs, running at TVMs of Cloud Compute Server (CCoS). Hence, it is free from anti-detection techniques such as obfuscation and encryption. It can access all the contextual information without requiring any complex security functions for information extraction. MSCSD provides an efficient approach, called 'Bag of n-grams' (BonG) for representing the system call sequences. BonG considers both frequency and structure of various short sequence of system calls (called n-grams) of each trace. MSCSD derives various n-gram patterns of same size for each trace. It is successful in maintaining the ordering of the subsequent system calls within each sub-sequence. It then converts the extracted n-grams into a numeric feature vector $< c_1, c_2, c_3, c_4, c_k >$, representing their frequency distributions in various types of traces. Each entry c in feature vector represents the occurrences of individual short sub-sequence in the trace. Machine learning is applied to learn the behavior of programs (both normal and maliciously modified) to learn their characteristics. It has been validated using University of New Maxico (UNM) dataset and achieves an accuracy of 72.103%-99.812% for detecting malicious programs. It provides various advantages over existing dynamic analysis approaches proposed for cloud . It improves the accuracy and reduces the storage requirement when compared to other approaches while maintaining the ordering of system calls.

The second-line of defense is provided by VM Introspection based Malware Detection (**VIMD**), deployed at VMM-layer in cloud. The main motivation behind VIMD is to detect the attacks at VMM-layer of the Cloud Compute Server (CCoS) which are bypassed by TVM-layer security solutions. VIMD makes use of the open source VMI libraries and performs two-levels of security check. The primary check ensures that all the security-critical processes, running at TVM such as auto-update, auto-scan are enabled. It also detects the presence of hidden processes/VM rootkits at TVM memory. If any suspicious activity is detected, cloud administrator is alerted about it. It then extracts the execution traces of running processes by using the kernel debugging based VM introspection approach. A detailed behavioral log of all the processes are obtained and secondary security check is performed. The secondary check analyses the program semantic (run-time) behavior at the VMM-layer using two different proposed detection mechanisms to detect the malware attacks from hypervisor: VMGuard and VAED.

VMGuard is based on the frequency model and proposes the integration of BonG (feature representation) method with text mining approach for feature selection particularly by Term Frequency-Inverse Document Frequency (TF-IDF) to extract the system call sequences with a high discriminative power. The extracted features form Feature Vector Matrix (FVM) log, representing the statistics of features. An ensemble learning classifier (Random Forest) is used to learn the program semantics and detect the intrusions. VMGuard considers the structural aspects of the traces and is found to perform well to detect attacks which do not depend on the system artifacts (e.g. privileged program subversion attacks). However, it does not capture the more complex behavioral aspects of the programs. Hence, It is less suitable to detect evasion based attacks which change their behavior on detection of some security tool. VMGuard provides good detection accuracy (94%-100%) in detecting anomalies when validated with UNM datasets.

VAED refers to VMI-assisted evasion detection approach, designed to deal with the evasion based attacks against virtual domains running in cloud. It considers both structural and behavioral aspects of the traces. It captures the semantics in different execution paths of programs and extracts the complex behavior of evasive malware. It is based on probability model which extracts the program semantics in form of system call dependency graph (SCDG), constructed using Markov Chain property. SCDG represents the ordered system call transitions of the program. The probability model considers the frequency of each system call transition with respect to the frequency of all other possible transitions. The transition probabilities are stored in form of Feature Transition Matrix (FTM). The information centric features are extracted by applying Information Gain Ratio (IGR) over FTM and stored in Optimal FTM (OFTM). OFTM is learned by ensemble classifier, based on fusion of diverse classifiers. The trained classifier captures the behavior semantics of evasive malware from FTM. It is used as baseline information

to detect abnormality in future instances. VAED has been validated with evasive attack dataset, obtained from University of California and provides 97.50%-98.8333% accuracy in detecting the evasive malware attacks.

The third-line of defense is provided by Malicious Network Packet Detection (**MNPD**), deployed at Network and VMM-layer in cloud. MNPD ensures the security from network intrusions by monitoring the tenant virtual network traffic with two-levels of security check. The primary security check performs the behavioral analysis of virtual network traffic at Cloud Networking Server (CNS). It provides the primary security from attackers, targeting CNS in cloud. The secondary security check validates the VM traffic at hypervisor of CCoS to detect spoofing attacks (IP/MAC) which is originated from VMs. To perform this, It does network introspection to gain the VM related information using open source tools such as Libvirt, Xenstore, dnsmasq server from Dom0 of hypervisor. The non-spoofed packets are further analyzed using behavior analysis of network traffic to detect any abnormality in the virtual traffic. It provides secondary security from attackers targeting the virtual domains running in the cloud at Cloud Compute Server (CCoS). MNPD employs statistical learning technique (Random Forest) with union of feature selection approaches (Chi Square and Recursive Feature Elimination) to learn the behavior of traffic patterns. MNPD does not incur overhead in monitoring extensive memory writes or instruction-level traces. It is a more secure solution to detect attacks which never pass through physical interface and hence, are not detected by traditional IDS. MNPD has been validated with UNSW-NB and ITOC datasets. It provides an accuracy of 98.88% using ITOC dataset and 95.091% using UNSW-NB dataset.

In summary, the thesis has presented a threat model and detailed analysis of different attacks that are possible in the cloud. A comprehensive intrusion detection framework, called CloudHedge is proposed with three lines of defense against different types of attacks in the cloud. The proposed techniques have been validated using different datasets and results seem to be promising.

Abbreviations

CSP	Cloud Service Provider
CCS	Cloud Controller Server
CNS	Cloud Network Server
TVM	Tenant Virtual Machine
VMM	Virtual Machine Monitor
VMI	Virtual Machine Introspection
IDS	Intrusion Detection System
SLA	Service Level Agreement
VM	Virtual Machine
NIDS	Network Intrusion Detection System
MSCSD	Malicious System Call Sequence Detection
VIMD	VM Introspection based Malware Detection
MNPD	Malicious Network Packet Packet Detection
TF-IDF	Term Frequency-Inverse Document Frequency
SCDG	System Call Dependency Graph
FTM	Feature Transition Matrix
ISCS	Immediate System Call Sequence
BoS	Bag of System Calls
FML	Feature Matrix Log
SCT	System Call Tracer
DE	Detection Engine
UNM	University of New Maxico
BonG	Bag of n-grams
DT	Decision Tree
SVM	Support Vector Machine
NB	Naive Bayes
TCB	Trusted Computing Base
EPT	Extended Page Table
PV	Process Validator
PET	Program Execution Tracer
VMMI	Virtual Machine Memory Inspection
VMLR	Virtual Machine Library Repository
PDB	Program Database File
PID	Process Identifier

GPA	Guest Physical Memory Address
MPA	Host Machine Physical Memory Address
altp2m	Alternate physical to machine
STIDE	Sequence Time Delay Embedding
TEP	Trace Execution Phase
FGP	Feature Generation and Pre-processing Phase
FVM	Feature Vector Matrix
RF	Random Forest
PIP	Pre-compiled Intrusion Profile
TPR	True Positive Rate
FPR	False Positive Rate
TNR	True Negative Rate
FNR	False Negative Rate
IGR	Information Gain Ratio
OOB	Out of Bag
IDS	Intrusion Detection System
API	Application Programming Interface
vNIC	Virtual Network Interface Card
DomID	Domain Identifier
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name Server
OVS	Open Virtual Switch
BCE	Behave Capturing Engine
TV	Traffic Validator
BAE	Behavior Analysis Engine
ALG	Alert and Log Generator
IP	Internet Protocol
MAC	Media Access Control

Contents

List of Tables

List of Figures

Chapter 1

Introduction and Problem Statement

1.1 Introduction

We are living in the era of cloud computing, where services are provisioned to the users on demand and 'pay-per-use' basis from a resource pool. Cloud computing has evolved gradually over period of time. In 1961, McCarthy [1] suggested that computer time-sharing technology might result in a future where computing will be provided as a public utility like electricity. At the beginning of 1980s clusters became the standard technology for parallel and high performance computing providing load balancing and fault tolerance [2]. A computer cluster is basically a group of computers that jointly performs computation to carry out some task. These computers can be connected via LAN to form a single computing entity.

In 1990s, grid computing evolved as a step further and as an aggregation of geographically dispersed heterogeneous computing nodes or clusters. Grid is utilized as a service on the top of internet. Grid computing provisions the sharing of resources like storage, disk, databases and software applications and computing power. Buyya et al. [3] defined grid as "a type of parallel and distributed system that enables the sharing, selection and aggregation of geographically distributed autonomous resources dynamically at runtime depending on their availability, capability, performance, cost, and users quality-of-service requirements". The first scholarly use of the term "cloud computing" was coined in a lecture talk by Ramnath Chellappa in 1997 [4].

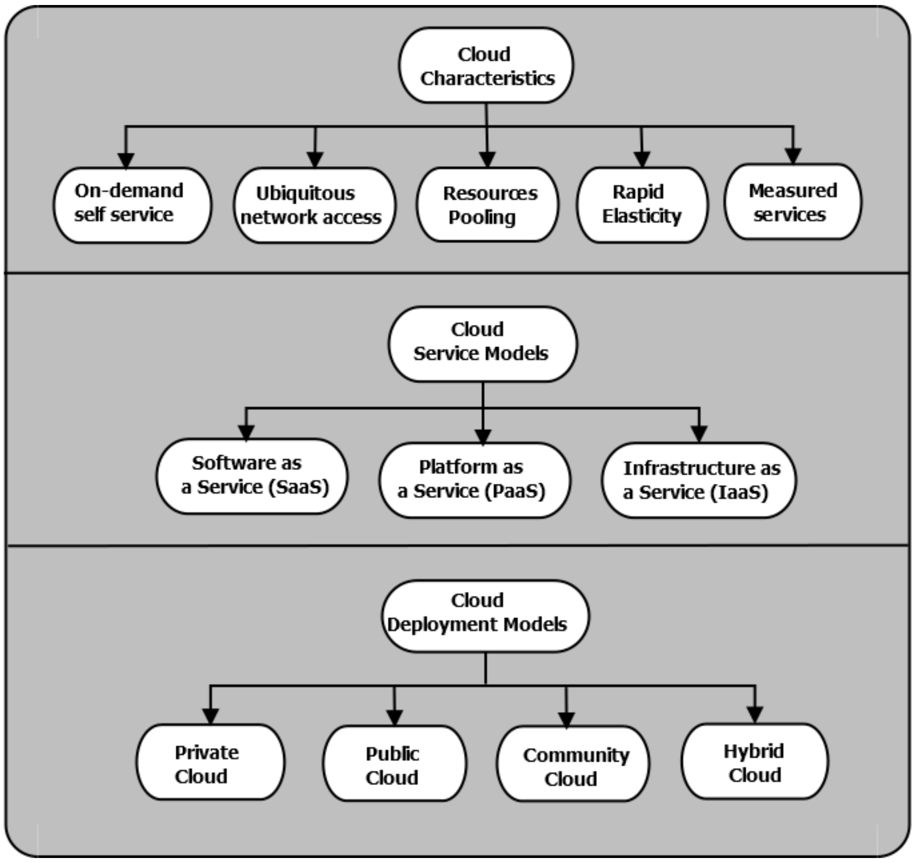


FIGURE 1.1: Cloud characteristics, Service models and Deployment models

Cloud computing goes one step further than grid computing, utilizing the power of virtualization. National Institute of Standards and Technology (NIST) defines cloud computing as "a model for enabling convenient, ubiquitous and on-demand network access to a shared pool of computing resources (e.g., servers, network, storage, services an applications) that can be rapidly provisioned and released with minimal management effort or service provider interaction"[5].

The characteristics of cloud with its service and deployments models are shown in Figure 1.1. NIST [5] has also defined the key characteristics of cloud computing as follows:

1. *On-demand self-service:* A cloud customer can opt for computing capabilities such as network, storage and virtual machines etc. automatically and does not require the interaction between customers and service provider.
2. *Ubiquitous network access:* Cloud customers can access the services over the network via standard methods which encourage the use of the heterogeneous client platforms such as mobile, laptops etc.

3. *Resource pooling:* A cloud services provider can pool its computing resources to fulfill the need of various cloud customers by utilizing the multi-tenant model, with provision and release of physical and virtual resources on a users demand. The resources include network, storage and compute etc.
4. *Rapid elasticity:* Cloud services and resources can be dynamically, elastically and automatically provisioned and released, to scale in-/out to meet with customers demands.
5. *Measured service:* The resource usage can be automatically optimized and controlled by cloud systems by employing the metering capability for each type of service. The resource usage is transparent to both customers and service providers.

Cloud services are provided based on the need of users. Different types of cloud service models [6] are mentioned as follows:

1. *Infrastructure as a Service (IaaS):* IaaS refers to providing resources such as physical or virtual machine on demand. IaaS cloud offers other services such as load balancer, firewall, disk image, block storage, VLAN etc. on demand. E.g. Amazon EC2, GoGrid, Google Compute Engine (GCE), Rackspace.
2. *Platform as a Service (PaaS):* PaaS refers to providing computing platform such as programming language execution environment, operating system, web server, database etc. E.g. Force.com, Google App Engine, Windows Azure.
3. *Software as a Service (SaaS):* SaaS refers to providing access to application software and databases on demand. Cloud provider manages the infrastructure and platform that runs the applications. E.g. Salesforce.com, Citrix GoToMeeting.

An organization must decide the deployment model based on its need before switching to cloud. There are mainly four types of cloud deployment models [7] as mentioned below:

1. *Private cloud:* Private cloud is exclusively used by single organization, managed internally or by third party. It offers high degree of control over security, performance and reliability.
2. *Public cloud:* Public cloud services are used by general public over internet. It lacks fine grain control over their network, data and security.

3. *Community cloud:* Community cloud offers the sharing of infrastructures between several organizations particularly communities with common specific concern (e.g. security, compliance etc.) use the shared resources. It is managed internally or by third party.
4. *Hybrid cloud:* Hybrid cloud exhibits the characteristics of both private and public clouds. It offers more flexibility than public or private cloud.

The key technology in the cloud environment is virtualization which creates an abstraction layer above the underlying hardware or software. It hides the complexity of physical hardware and allows multiple operating system to run on same physical machine. The abstraction layer is called as Virtual Machine Monitor (VMM) or Hypervisor [9]. It is of two types: Type I (Bare Metal Hypervisor) and Type II (Hosted Hypervisor) as shown in Figure 1.2. In Type I Hypervisor, VMM can directly access physical hardware. Hypervisor is booted first and have access to the real device drivers. Xen [10], Hyper-V [11], VMware ESX/ESXi [12] are some examples of Type I Hypervisor. In Type-2 Hypervisor, host OS is loaded first. The Hypervisor is loaded post-boot when the first VM is launched. The Hypervisor runs above the host operating system as a user space application. It shares device drivers from host OS to handle the input-output and completely depends on host OS for its operations. VMware Workstation [13] and Oracle Virtual Box [14] are examples of type II Hypervisor. Gradually software were developed for implementing cloud computing platform such as Open Nebula [15], VMware vSphere [12], OpenStack [16], Apache CloudStack [17], Citrix

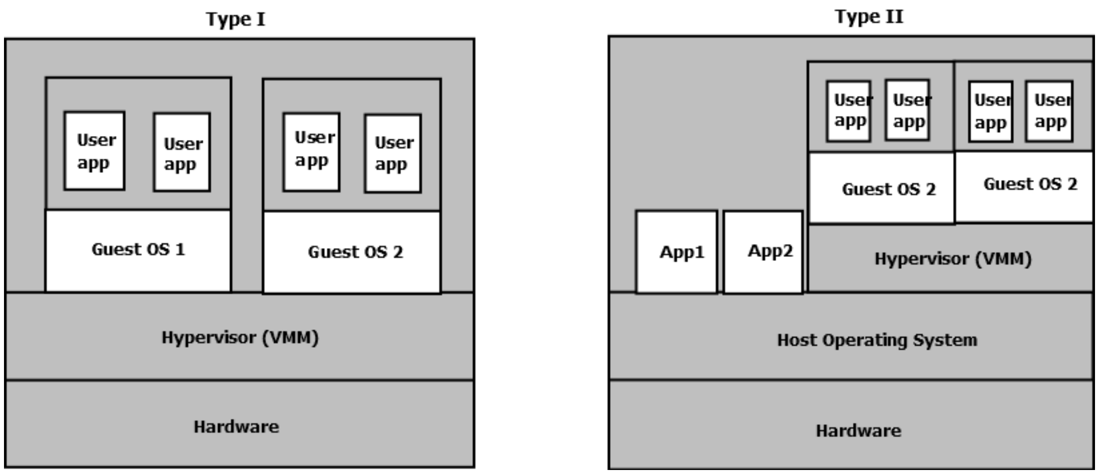


Figure 1.2: Types of Hypervisor [8]

XenSever [18], HP Helion Eucalyptus [19] etc. In our work, we have considered Xen VMM for implementation and considered the cloud architecture based on OpenStack [16], a leading global cloud management software.

Xen VMM is booted first as a primary boot system. Afterwards, Linux kernel is loaded as Dom0 domain by the Hypervisor. Dom0 is the privileged domain (administrative VM) which is used to control, configure and manage all the other VMs by the cloud administrator. Dom0 runs the device drivers and can access the actual hardware as shown in Figure 1.3. The networking between the TVMs is provided by VMM. Networking in VMM bridges the virtual adapter to the physical adapter. The tenant virtual machines (TVMs) are loaded after Dom0 and are also referred as untrusted domains (DomUs). VMM has the highest privilege and full control over any VM running over it.

Openstack is a collection of open source cloud components used for developing cloud platform for public, private and hybrid cloud [21]. Initially OpenStack in jointly launched by Rackspace Hosting and NASA, as an open-source cloud-software in 2010 [22]. Later, SUSE, Red Hat, Oracle, IBM and some other companies launched their OpenStack based distributions. VEXXHOST [23] provides public cloud services, powered by OpenStack. AURO public cloud [24] offers Infrastructure as a Service Cloud (IaaS) and is also powered by OpenStack. RackSpace public cloud [25] also uses OpenStack for implementing the infrastructure for cloud services. DataCentred public cloud [26] is another solution for implementing IaaS services which is based on OpenStack. ELASTX

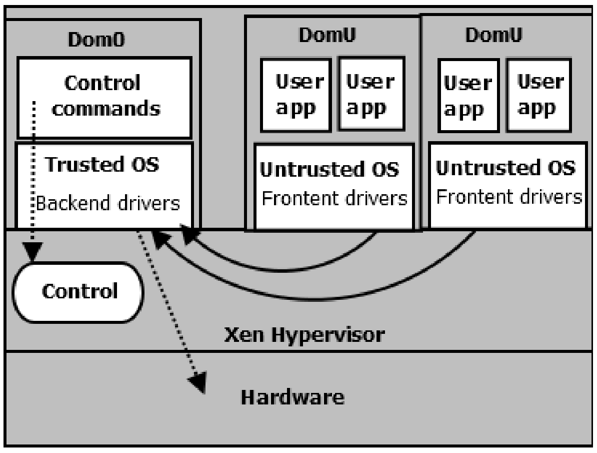


Figure 1.3: Basic Architecture of Xen Hypervisor [20]

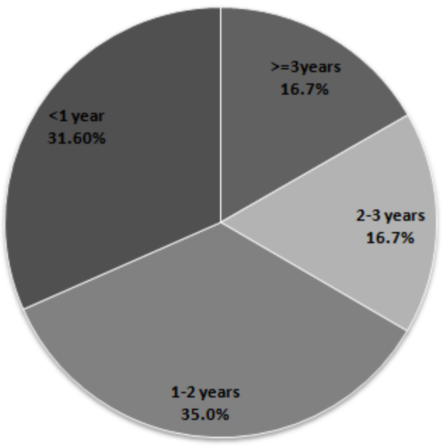


FIGURE 1.4: Adoption of cloud computing by Companies over years [33]

OpenStack:IaaS [27], Dualtec [28], AgileCLOUD [29], are other examples of OpenStack based public cloud. Rackspace private cloud [30] and IBM Bluemix private cloud [31] are also powered by OpenStack. In the OpenStack summit 2016 at Barcelona, Platform9 [32] announced its contribution towards hybrid cloud implementation, using OpenStack. They extended the project and included OpenStack drivers for Amazon Web Services (AWS) to integrate the OpenStack services (nova-compute, neutron etc.) with AWS.

Users are gradually adopting cloud for hosting their applications and data. According to the survey done by Cloud Security Alliance (CSA) [33], 62.0% users are active users who have adopted cloud for hosting some of their applications and data. There are 27.3% potential users who are evaluating the cloud vendors based on trial periods and 10.7% users have no plan to move to cloud. They have further classified active users based on their years of experience. There exist 31.6% active users who have been using cloud for less than a year, 35.0% active users have hosted their applications for more than a year but less than two years. There exist 16.7% active users who have been taking benefits of cloud for more than three years. The same amount of users (16.7%) have been using cloud between two to three years as shown in Figure 1.4.

The ease and flexibility with cloud services have opened doors for attackers. Some of the attacks have been reported by cloud providers and users. For instance, the French research outfit VUPEN Security [34] discovered the Virtual Machine Escape attack. The exploit targets

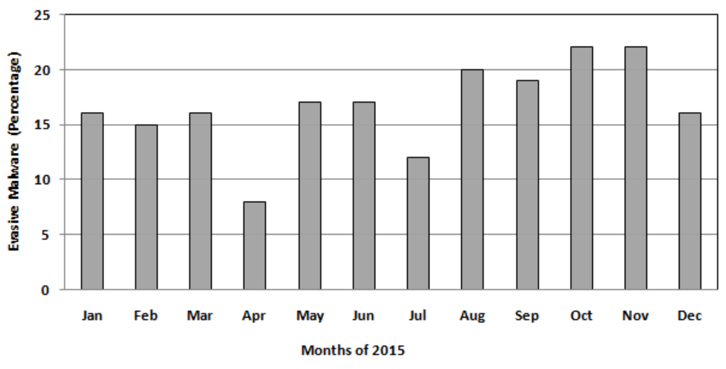


FIGURE 1.5: The attack statistics of virtualization-aware evasive malware samples in cloud environment [39]

a vulnerability that affects the way Intel processors implement error handling in the AMD SYSRET instruction. In Jan 2013, European Network and Information Security Agency (ENISA) reported [35] that Dropbox was attacked by Distributed Denial of Service (DDoS) attacks and suffered a substantial loss of service for more than 15 hours affecting all users across the globe. DDoS botnet attacks also launched against the Amazon Cloud. Security researchers [36] have found the exploit on the Amazon Cloud platform through the ElasticSearch distributed search engine tool. Hackers attacked Amazon EC2 virtual machines using cve-20143120 exploit in ElasticSearch ver. 1.1 x. Cyber threat defense reported that 75% attacks use the publicly known vulnerabilities present in the commercial software [37]. In 2014, Code Space was hacked by attackers which caused the destruction of most of the customers' data [38]. Internet Security Threat Report [39] stated the proportion of evasive-malware samples in cloud environment in the year 2015 that can detect the virtualization environment. Approximately 16% malware samples can detect the virtualization environment and try to evade the security tool running in the virtual machines as shown in Figure 1.5. According to Symantec report [40], 494 vulnerabilities and two zero-day vulnerabilities were disclosed during the month of January in 2015. W32. Ramnit! html was the most common malware that had been blocked. Verizon [41] reported 55% of the incidents were insider abuses/attacks in the year 2015. Stolen credentials accounted for 50.7%, backdoors were 40.5%, SQL Injection were 19%, brute force were 6.4%, and cross site scripting (XSS) attacks were 6.3%. Cisco [42] reported that malware and unwanted applications are being distributed by malware developers by using web browser add-ons as a medium. The

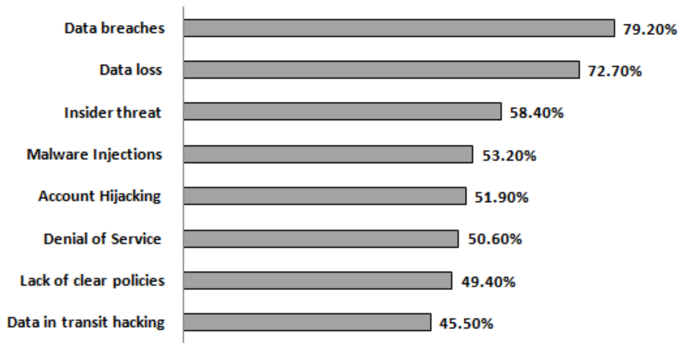


FIGURE 1.6: Key security attacks in cloud [33]

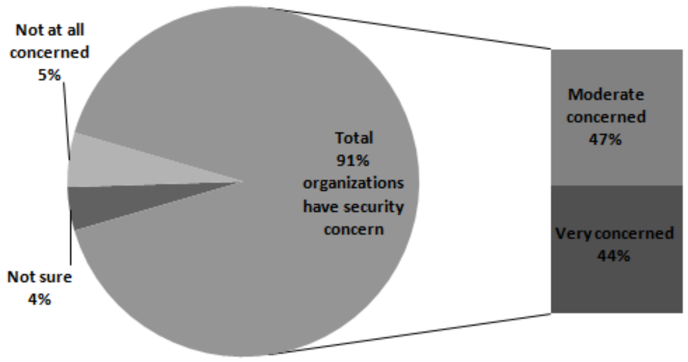


FIGURE 1.7: Cloud security concern-level by cloud service providers [44]

proportion of top attacks on cloud are reported by CSA [33] as shown in Figure 1.6. Most of the attacks are above 50% of total proportion. On a recent survey done by Cisco in 2017, Trojans was classified as one of the top five malware attack and is used to gain initial access to the user's computers and organizational networks. Their investigation reveals that around 75% organizations are affected by malicious software infections. These malware can be used to launch further attacks [43].

1.2 Cloud Security

The attack incidents in cyberspace raise a big concern for security. According to the survey done by Information Security Group [44] in 2016, 91% of organizations are very much or moderately concerned about the cloud security as shown in Figure 1.7 and only 4% to 5% organization are less or not concerned about security. Some of the important security aspects for cloud are shown in Figure 1.8. These are Application level, Network level, Virtualization level, Data storage level, Identity management and Role based access control, Cryptographic key management level, SLA and trust level, Auditing, governance and regulatory compliance and Cloud & CSP migration level security. They are discussed below in detail.

Application level security issues are concerned with the security of web applications running in the cloud to provide cloud services. The SaaS application has to be managed over the web (using a browser). The web application security is tightly coupled with the security of web browsers. A web browser is a platform independent program, used to access the cloud services (SaaS), web 2.0 or web pages. A web browser uses SSL/TLS protocol for secure transmission. The security loop holes in the web applications create the vulnerabilities in the SaaS applications. The web applications are prone to a number of threats such as cross-site scripting (XSS), SQL injection attack, broken authentication, insecure transport layer protection, cross-site request forgery (CSRF) etc.

Network level security issues are concerned with the security of the cloud network. One of the key issues at network level is unavailability of services. Denial of Service (DoS) and Distributed Denial of Service (DDoS) are main threats to service unavailability. These attacks cause inconvenience to customers and prevent their access to the cloud services. HTTP based and XML based DDoS, are called as Economic Denial of Sustainability (EDoS) which affect the pricing model of cloud [45]. The key security issues at the network-level are authorization, authentication, intrusion detection, vulnerability assessment, session hijacking, etc. Some common attacks at network-layer are sniffing, scanning, IP/MAC spoofing and DNS poisoning etc.

Virtualization level security issues are concerned with security of virtualization layer. The major vulnerability is the multi-tenancy in which multiple tenants share and utilize the cloud platform [46]. As the number of TVMs running above Hypervisor increase, the security issues with the new TVMs also increases. Maintaining the security policies of all TVMs is challenging task. Malicious code running inside a TVM may try to gain root privilege of the Hypervisor with an intention to take full access of the system. Security of TVMs is one of the crucial concerns in the cloud environment. Once Hypervisor is compromised, all TVMs running on it will be under the control of the attacker [47]. Infact, a improperly configured Hypervisor can fail to provide proper

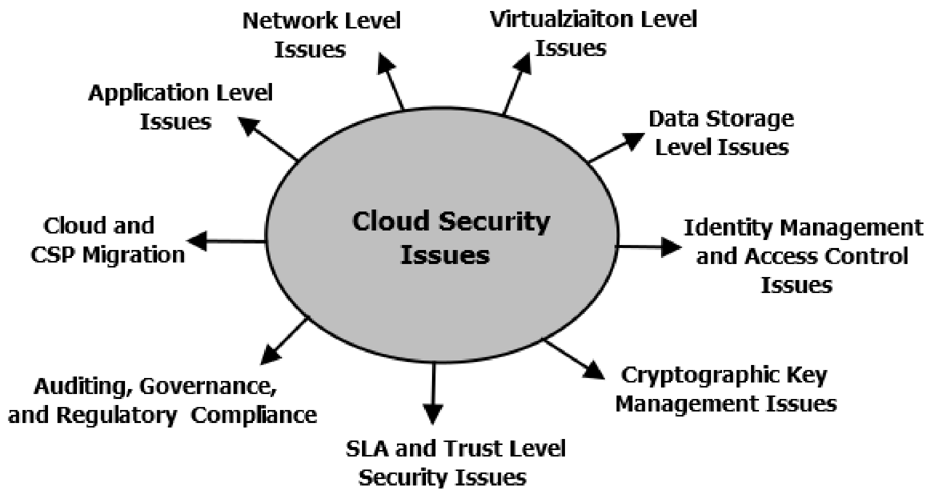

Network Level Issues
Virtualziaiton Level Issues
Application Level Issues
Data Storage Level Issues
Cloud Security Issues
Cloud and CSP Migration
Identity Management and Access Control Issues
Auditing, Governance, and Regulatory Compliance
Cryptographic Key Management Issues
SLA and Trust Level Security Issues

FIGURE 1.8: Cloud security issues in cloud

isolation among TVMs, leading to disclosure of the tenants' sensitive data.

Data security is vastly an open research area. Data can be in transit (communicated via network channels) or in rest (stored in data centers). The vulnerabilities in the network protocols and/or poor encryption directly affect the confidentiality and integrity of data. The data stored in the servers needs to be physically and logically segregated and have control policies. A few years back, Amazon reported that its Elastic Block Store (EBS) volumes were trapped which affected its EC2 instances [48]. Some of the related security issues are data remanence, data recovery, data segregation and data integrity etc. Data remanence refers to data which is left out after transfer or removal of VM. Data recovery is preferred when data is lost because of some accidental damage. Data segregation is the organization of the data of various users residing in the same location. Ensuring the isolation between the user's data is an important security concern. Data integrity ensures that there is no illegitimate modification in the user's data [6]. Data deduplication is a approach for removing duplicate copies of data. Secure data deduplication is a major research concern [49].

Identity management and access control issues are also important security concern. Identity management (IDM) deals with identifying the entities in cloud and controlling their access to resources. As the customers' credentials are transmitted via internet, it imposes a great risk to user's sensitive data. The issue is addressed by providing the support for federation protocols such as Service Provisioning Markup Language

(SPML) or Security Assertion Markup Language (SAML) [50] to some extent. SAML supports both authentication and authorization. Some other protocols are created after SAML such as OpenID and OAuth2. OAuth2 is an open standard for authorization and OpenID is an open standard for authentication. The cloud based IDM are prone serious threats such as brute-force attack, cookie replay attacks, eavesdropping attack, denial of service attack and data tampering attack, etc [51]. There is a need to design strong security measures for IDM systems. There is a need of providing fine-grained access control mechanisms for controlling access to user's data. For example, Google App uses eXtensible Access Control Markup Language (XACML) for authorization and access control. Mon et al. [52] combined the Attribute-based Access Control with Role-based Access Control (RBAC) to ensure the privacy and security of user's data.

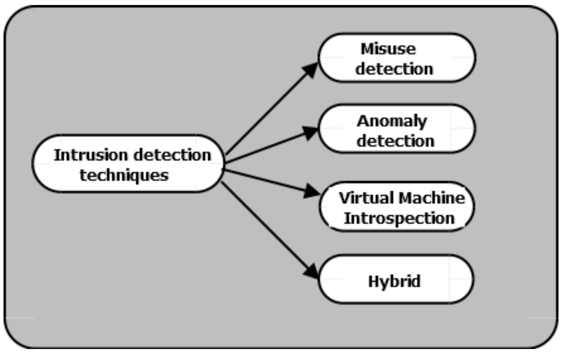
Improper cryptographic keys management leads to failure of cloud security measures [53]. The cryptographic approaches such as cryptographic hash function, digital signature and message authentication code etc. are used to authenticate the VM templates in cloud. They may prone to the bootstrapping problem and hence, requires a strong security analysis. The key security requirements for key management systems must be ensured. Some of them are discussed as follows. The key management commands and data should be secure from spoofing and illegitimate modification. The third party who does key management should be authentic. All the secret and private keys should also be protected from disclosure. The cryptographic mechanism employed for protecting keys should be strong enough and robust from attacks [54].

Service level agreement (SLA) and trust level security is another important concern. The customers lose their control over data and programs which are outsourced to cloud servers. Cloud service providers limit the visibility of data location, network and system monitoring to customers which generate the trust issues with service provider. It is very difficult to assure trust in cloud environment. However, the use of signature techniques and advanced cryptographic techniques can be used to deal with the trust issues to some extent. SLA is another way to deal with the trust issues to certain limit. An SLA is signed at the time of registration, describing the minimum performance criteria a CSP should meet when delivering services. If a certain service fails to meet the customers need or quality of service (QoS) do not meet the SLA, cloud customers can lose their trust with the CSP [55].

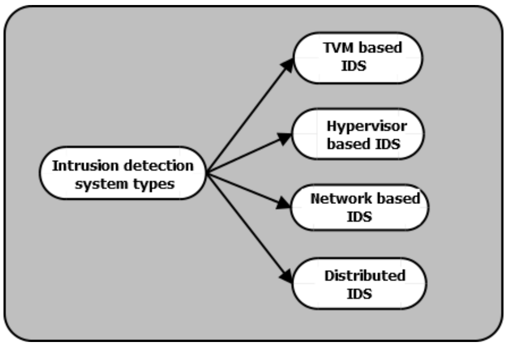
Regular audit and compliances to manage cloud resources must be done to ensure whether internal and external processes are meeting the customer requirements, regulations and laws. The policies should be monitored regularly. There are some general governance standards that are also applicable to cloud computing environment such as ISO/IEC 38500 IT Governance [56], Control Objectives for Information and Related Technology (COBIT) [57], Cloud Security Alliance (CSA) Cloud Controls Matrix [58] etc. The law and regulations of different countries are different. Therefore, some of the compliance operate at country-level, or regional-level [55]. Some of the standards are applicable to specific company or data. The Health Insurance Portability and Accountability Act (HIPAA) [59] requires the U.S. health care organization to maintain the confidentiality of protected health information (PHI). Payment Card Industry Data Security Standard (PCI-DSS) [60] defines the minimum security controls to secure the customer data. The Federal Information Security Modernization Act (FISMA) [61] is a compliance framework that enforces the protection of information systems and assets of all federal government agencies and contractors. Sarbanes-Oxley Act (SOX) [62], a federal regulation, provides the standards for all U.S. publicly traded companies to ensure security to all shareholders and public from fraudulent actions. It maintains the information policies and prevent the illegitimate data tampering.

There are some other security issues associated with cloud and CSP migration. When an organization or cloud customer is entering into the cloud or shifting from one CSP to another CSP, the following migrations will be considered: Data (application) migration and Cloud migration. Migration is one of the challenging research area. It involves the secure transmission of the tenants' data with strong application and network security measures together with governance compliance. There are many questions that need to be resolved with tenants such as What technology is used in migration? Is the CSP migrating the data with appropriate policies in place? Is the migrated data secure? Is the migration secure from attackers? etc [63].

Researchers are working in different domains of security as discussed above to address the security issues. We have considered intrusion



(A) Types of intrusion detection techniques



(B) Types of IDS

FIGURE 1.9: Broad classification of intrusion detection techniques and types of intrusion detection system (IDS) in cloud

detection as one of the key security aspects to detect attacks at different layers in cloud.

1.3 Intrusion Detection in Cloud

In the last few years, research has been carried out to tackle security problems in cloud environment. Various researchers working in the area of cloud security have proposed intrusion detection systems (IDS) as one of the defense approaches. An IDS is a security tool that captures and monitors the network traffic and/or system logs, and scans the system/network for suspicious activities. It further alerts the system or cloud administrator about the attacks.

Different intrusion detection techniques used by IDS in a cloud environment include misuse detection, anomaly detection, virtual machine introspection (VMI) and hybrid techniques as shown in Figure 1.9a. Misuse detection techniques maintain rules for known attack signatures. These rules can be derived either by using the knowledge based systems which contain database of known attack signatures or by using machine learning algorithms that are used in the determination of behavioral profiles of the users based on known suspicious activities [64]. Anomaly detection systems detect anomalies based on the expected behavior of the system. Any deviation from the expected behavior is signaled as anomalous [65].

Another well known technique is that of Virtual Machine Introspection (VMI). The basic principle behind the VMI technique is that it performs introspection of programs running in a VM to determine any malicious program change or execution of some abnormal or malicious code [66]. Different methods of VM introspection are based on guest-OS hooks, VM state access, kernel debugging, interrupt and hypercall authentication, etc. They bridge the semantic gap in interpreting the low-level information available at a VM to high level semantic state of a VM. Hybrid techniques make use of the combination of misuse, anomaly and/or VMI approaches.

There are four types of Intrusion Detection System (IDS) in cloud, classified based on the location of deployment and target of attack, i.e. TVM based IDS, Hypervisor based IDS and Network-based IDS and Distributed IDS as shown in Figure 1.9b. Each of these IDS makes use of the one of the types of techniques as mentioned before. They are described below:

TVM monitoring based IDS performs monitoring at TVM-layer. At TVM-layer, the IDS is configured and executed inside a TVM and hence it has good visibility of the monitored TVM. It performs host audit log analysis, system call analysis and program analysis of the monitored TVM. It sends alerts to tenants on detection of the suspicious activities. It detects suspicious activities such as modification or deletion of system files, unwanted sequence of system calls or unwanted configuration changes at TVM or in other cloud regions. It has direct access to all contextual information of monitored TVM. Hence, performance in such a deployment scenario is good. However, it is less attack resistant as the security monitor can be easily compromised. The advantage with guest monitoring solution is that it does not require any modification in the Hypervisor and runs as an application in a tenant virtual machine, which is configured and controlled by the tenants. It is suitable in all cloud deployments to provide defense from attackers without the need of introspection functionality. Bag of System Calls (BoS)-IDS [67] and Secure In-VM Monitoring (SIM) [68] are some examples of TVM-based IDS tools.

Hypervisor based IDS performs monitoring at VMM-layer. Security tool is installed at the Hypervisor (VMM/Dom0) and is completely controlled by cloud administrator. There are some VM Introspection libraries such as Ether [75], LibVMI [76], Libvirt [77], XenAccess [78] and VMsafe [79] which can be used to retrieve the information about activities happening in a VM. The information is further analyzed by security tools running at the VMM, which are configured and controlled by the

Table 1.1: Types of IDS in Cloud

Parameter	**TVM-based IDS (A)**	**Hypervisor-based IDS (B)**	**Network-based IDS(C)**	**Distributed-IDS (D)**
Placement of IDS	TVM	VMM	virtual/physical network points (TVM/VMM/Network)	TVM, VMM or network points, or CCS
Visibility of monitored machine	Good	Average	Poor	Good for DTVM-IDS average for DVMM/Detwork-IDS
Performance	Good	Average	Poor	Better than A for DTVM-IDS and Better than B/C for DVMM/Dnetwork-IDS
Attack Resistance	Less	High	Higher than others	High for DVMM/DNetwork-IDS and Less for DTVM-IDS
Hypervisor Dependency	Independent	Dependent	Independent	Dependent for DVMM-IDS and Independent for DTVM/Dnetwork-IDS
Configured and Controlled by	Tenants	Cloud Administrator	Tenant/Cloud Administrator	Cloud Administrator
Introspection Functionality	Not Applied	Applied	Not Applied	Applied in DVMM-IDS only
Tools used	BOS [67], SIM [68]	XenIDS [69] VMwatcher [70]	SNORT-IDS [71]	Collabra [72], ISCS [73] Cooperative-agent [74]

cloud administrator to detect any suspicious activity in the monitored VM. These libraries are Hypervisor dependent. Hypervisor-based IDS can access VM-specific information from the privilege domain of VMM and there is a moderate level of visibility of contextual information of the monitored machine. Moreover, VMM is more secure than a TVM, making the security tool highly resistant to attacks when compared to other approaches where security tool runs in the same monitored machine. However, the performance of Hypervisor-based IDS is average when compared to other IDS. This is because of low-level semantic gap [80] between guest OS and host OS and due to privacy concerns of users preventing access of VM information. Hypervisor-based IDS is VMM dependent and is best suited to provide defense from attackers in IaaS cloud where security tool is under monitoring of cloud administrator. XenIDS [69] and VMwatcher [70] are some examples of Hypervisor-based IDS tools.

Network-based IDS performs the network traffic monitoring and are independent of underlying operating system. This is flexible to be deployed at any layer (TVM/VMM/Network). However, the network locations such as virtual bridge (joining a group of TVMs) or physical network switches connecting the sub-networks of cloud physical servers are most suitable locations to detect network attacks in cloud. In this case, the security tool has poor visibility of the monitored VM and its performance is not as good as above two types of IDS, particularly for detecting host based anomalies such as rootkits, virus, worms and program-subversion attacks etc. Detection of host based anomalies requires host audit log information such as system calls invoked by monitored programs. Such information cannot be accessed from network points. However, traffic analysis can be helpful in detecting network attacks such as denial of service, port scanning and spoofing attacks. The network security tool that is deployed outside the monitored machines at network servers, is more resistant to attacks compared to IDSes as discussed before; it is also Hypervisor independent and does not require any introspection functionality. The Network-based IDS, deployed at TVM is better suited to all cloud deployments and is controlled by tenants. However, Network-based IDS deployment at VMM/network server is well suited to IaaS cloud and is configured by cloud administrator on demand of tenant user. SNORT based Cloud-IDS [71] and Cloud-NIDS [81] are some examples of network-IDS.

Distributed IDS consists of multiple IDS instances (Guest based IDS, Hypervisor-based IDS or Network-based IDS) which are distributed over the large network of cloud. These instances either communicate with each other or are centrally controlled by cloud administrator. Distributed IDS inherits the qualities of the IDS instances (TVM-based/Network-based/Hypervisor-based) deployed at the different regions. The visibility and attack resistance depends entirely on the type of IDS instances deployed, as discussed above and shown in Table 1.1. However the performance of distributed IDS is better than other type of IDS as discussed above. It can be Hypervisor dependent or independent, depending on the location of the IDS sensor. Gupta et al. [73] proposed distributed architecture where Guest-IDS are deployed at each TVM (DTVM-IDS) and centrally controlled by cloud administrator. It is Hypervisor independent. Bharadwaja et al. [72] proposed a distributed architecture (DVMM-IDS), called 'Collabra' in which Hypervisor based IDS instances are deployed at each VMM and communicate with each other to update about attacks. In Collabra-IDS, the communication is done via VMM event channels. This makes it Hypervisor dependent. In the case of distributed IDS in which Network-based IDS instances are deployed at the guest or network level, communication is done via network interface in a traditional manner which makes the approach Hypervisor independent. For example, Lo et al. [74] proposed a distributed architecture (DNetwork-IDS) which is called cooperative intrusion detection framework. Multiple Network-based IDS instances are deployed in each cloud server which cooperate with rest of the instances for detecting attacks. A cooperative agent in each IDS is responsible for receiving alerts from other IDS components. It decides on the majority vote based on a threshold value. If majority vote is higher than the threshold, a new blocking rule is added in the block table alongside the signature database.

The detailed description of intrusion detection systems (IDSes) with their security architectures and techniques are provided in Chapter 2.

1.4 Motivation

Attack incidents are increasing day by day in cloud computing environment. Hence, there is an immense desire to understand the security solutions applicable to cloud environment from research point of view. Traditional IDS systems have been applied to a cloud environment by several researchers. For example, Roschke et al. [82] proposed a Snort based IDS architecture named as VM-Integrated IDS to detect anomalies. Modi et al. [83] used Snort and machine learning classifiers to detect anomalies in the network traffic between VMs. Alarifi and Wolthusen [67] used traditional Bag of System Calls based approach to detect anomalous sequences present in the user programs during execution. Gupta and Kumar [84] proposed Immediate System Call Sequence approach (ISCS) which is similar to traditional look-ahead based approach [85]. Li et al. [86] applied Artificial Neural Network (ANN) to detect intrusions in cloud. Singh et al. [87] applied Decision Tree (DT) and ANN with SNORT to detect intrusions in the cloud. In all the above approaches, the IDS works in a standard manner and is deployed at the end host cloud servers or tenant virtual machines. However, a recent survey by Information Security Group [44] confirms that only 14% security professionals expressed that traditional security tools are sufficient to manage their security needs in cloud. A total of 59% security professional claim that traditional security tool either ‘does not work at all’ or ‘somewhat works’ as shown in Figure 1.10.

Modern malwares can easily thwart traditional HIDS based on signature matching or static analysis techniques by using obfuscation and encryption techniques [88]. Dynamic analysis based traditional IDS can be evaded by checking the presence of specific security processes in the memory of monitored tenant VM or end host as security analyzer is deployed in the monitored machine. In addition, a malicious malware program can sense the virtual environment by checking the

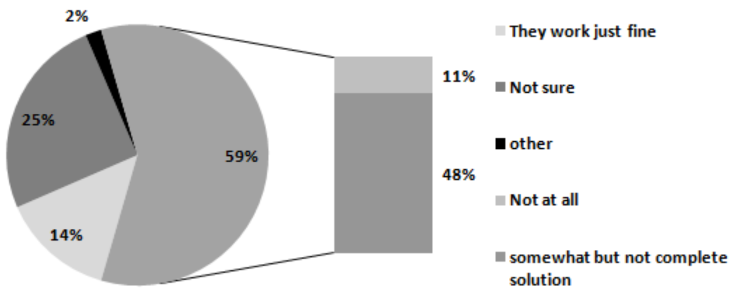


FIGURE 1.10: Traditional security tools in cloud [44]

registry key values and the presence of drivers specific to virtualization. An attacker can also try to sense the periodic behavior of security analyzer by observing the monitored machine [89]. Traditional NIDS tools such as Snort-IDS employed by existing cloud security solutions [82][90][87], fail to detect VM attacks targeted from one tenant VM to another on the same physical server. The internal virtual network traffic of co-resident TVMs never passes through the physical network.

Introspection based approaches are more specialized intrusion detection approaches developed to work with virtualized systems and cloud environment. Hypervisor-based IDS makes use of these approaches and has been proposed in virtualization environment. Maitland [91] and VMWatcher [70] are examples of Hypervisor-based IDS. However, they provide limited detection functionalities and lack in providing an efficient detection mechanism at different layers in cloud environment. Moreover, some of Hypervisor-based IDS are not compatible in cloud environment. For example, VMST [92] requires that the OS of the security VM (monitoring VM) must match the OS of the TVM being introspected. This is not feasible in cloud. Some approaches impose rigid design constraints. For example, ShadowContext [93] enforces the execution of all monitored system calls of monitored VMs at Dom0 of Hypervisor. Improperly designed system call redirection module can crash the system and will impose significant overhead.

The previous research indicates that existing IDS frameworks, as discussed above provide same security solution at same or different regions in cloud to detect a variety of attacks in cloud. Network-layer solutions do not provide good detection rate for low frequency attacks such as virus, rootkits, evasive malware etc. Some of the TVM-layer solutions target the detection of low frequency attacks. However, they are less efficient and are not feasible to be applied at Hypervisor because of the semantic gap issues [92]. None of the research presented so far provides a comprehensive solution to the problem. Rather, they perform monitoring at a particular area, providing security at the TVM-level, Hypervisor-level or Network-level. There is need to provide a complete solution which provides security to all the vulnerable areas of the environment.

As the existing methods alone are insufficient to handle various attacks against virtual domains running in cloud, the motivation behind this work is to provide an efficient, robust, VMI-based and distributed security framework for securing the cloud environment. The framework is designed to detect attacks with three line of defense at TVM, VMM and Network layer. The distribution of the security components with different detection strategies makes the proposed solution more efficient and suitable to cloud environment.

1.5 Statement of the Problem

The main objective of the present research work is as follows:
"*To develop a comprehensive intrusion detection framework to detect attacks in cloud environment by provisioning three-lines of defense, covering all three layer of cloud, i.e. TVM-layer, VMM-layer and Network-layer*".

The framework is intended to provide different detection strategies deployed at various security-critical positions such as tenant virtual machine (TVM), Hypervisor/virtual machine monitor (VMM) and cloud network server (CNS) to facilitate the detection of both network and malware attacks in cloud. The design of framework with proposed solutions for detecting intrusions at various cloud layers strengthens the detection mechanism. The framework aggregates existing open source network security tools and virtual machine introspection tools and integrates them with efficient detection mechanism to facilitate the objective of detecting network attacks and basic & modern evasive malware attacks.

The contributions can be subdivided as following objectives:

1. To propose a threat model and attack taxonomy at various layers in cloud environment.
2. To propose a classification of intrusion detection mechanisms in cloud environment.
3. To propose a robust, efficient and VMI-based distributed security framework to detect intrusions at different layers of cloud environment.
4. To propose and implement an efficient security approach to detect intrusions at the TVM-layer.
5. To propose and implement a robust VM Introspection based security architecture to detect malicious activities at the VMM-layer.
6. To provide the design and implementation of an efficient VMI-assisted malware detection approach based on system call sequence analysis for detecting the basic malware attacks at VMM-layer .
7. To provide the design and implementation of an efficient VMI-assisted evasion detection approach based on system call transition analysis for detecting stealthy evasive malware attacks at VMM-layer.
8. To propose and implement a malicious network packet detection approach to detect network intrusions at Network and VMM-layer.

1.6 Thesis Organization

Rest of the thesis is organized as follows:

Chapter 2 proposes a threat model and an attack taxonomy with the description of cloud attackers and attack scenarios in cloud. A classification of detection techniques of IDS has also been proposed followed by the detailed discussion on various types of IDS in cloud environment. Various research gaps have also been identified and discussed.

Chapter 3 proposes an efficient, robust and distributed security framework, called CloudHedge which provides three-line of defense at different security-critical layers namely TVM, VMM and Network. Three security proposals has been provided for TVM-based monitoring, Hypervisor-based monitoring and Network-based monitoring respectively to deal with hidden rootkits, stealth malware and network intrusions. Each of the detection mechanism is distributed at different security positions in cloud which is centrally controlled, configured and monitored by cloud administrator. A brief summary of conceptual working of all three proposals has been described with diagrams and unique qualities.

Chapter 4 provides the design and implementation of proposed 'Malicious System Call Sequence Detection (MSCSD)', one of the sub IDS instance of CloudHedge. MSCSD is based on the run time behavior analysis of the programs at TVM-layer. It can directly access all contextual information of TVM as the security tool is deployed inside the TVMs. A detail description of various detection components has been presented with their implementation.

Chapter 5 provides the design and implementation of proposed VM Introspection based Malware Detection (VIMD), second sub IDS instance of CloudHedge. VIMD performs the program behavior monitoring from the VMM-layer. The key characteristics has been described with its execution phases and detection components. The detailed description about the design and implementation of VIMD with core detection components (called VMGuard and VAED), used for behavior analysis of programs, are described in more detail in separate sections. VMGuard is based on system call sequence analysis and intended to detect basic malware attacks such as program subversion attacks whereas VAED is system call transition analysis and intended to detect evasive-malware attacks that try to thwart the detection. The detection mechanism of both the approaches is integrated with suitable text-mining and machine learning approaches along with the support of VMI functions.

Chapter 6 provides the design and implementation of proposed 'Malicious Network Packet Detection' (MNPD), third sub IDS instance of CloudHedge. MNPD provides the defense from network intrusions at both Cloud Network Server (CNS) and Cloud Compute Server (CCoS). It leverages the network introspection and machine learning approaches for doing traffic validation and behavior analysis of the network traffic.

Chapter 7 concludes with summary of contributions towards intrusion detection in cloud and also gives directions for future work.

Chapter 2

Literature Survey

This chapter begins with the explanation of the proposed threat model and proposed attack taxonomy in the cloud computing environment. A classification of various intrusion detection techniques has also been proposed with the description of their applicability in cloud environment. The existing Intrusion Detection Systems (IDSes) in cloud environment have been discussed in detail. Research gaps are discussed which are identified during the study.

2.1 Threat Model and Attacks in Cloud Environment

First of all, a brief overview of the cloud architecture, based on OpenStack [16] architecture, is discussed. OpenStack is a global leading cloud management software opted by many companies for developing cloud platform for public, private or hybrid cloud, as discussed in Chapter 1 in detail. Next, a threat model is explained with associate attack surfaces and attackers in cloud environment. The attacks are classified based on target cloud components. A cloud environment typically consists of three types of servers: Cloud Controller Server (CCS), Cloud Compute Server (CCoS) and Cloud Networking Server (CNS) [16] as shown in Figure 2.1. All the management related tasks of a cloud are handled at CCS whereas CCoS hosts various virtual machines (VMs). CNS facilitates configuration of network, IP allocation and traffic routing to cloud servers. It also enables VMs to connect to the Internet. There are typically three networks in cloud: tenant network, administrative network and external network. Tenant traffic flows through the

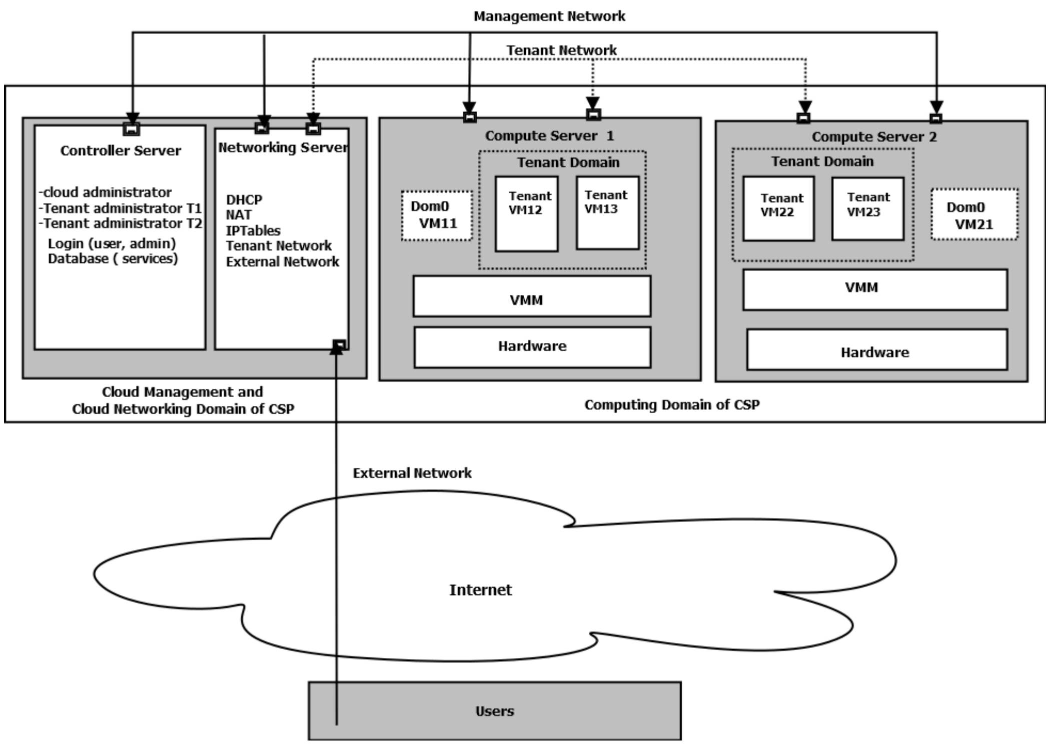


Figure 2.1: Basic architecture of cloud environment

tenant network which is configured to run in a virtualized environment. Each tenant network is associated with a set of tenant virtual machines (TVMs) and hence vulnerable to attacks from one TVM to other TVM in the same tenant subnet. An administrative network is responsible for connecting all cloud servers and is basically used to perform administrative task such as creating a TVM, destroying a TVM, resuming a TVM, allocating storage and are less vulnerable to attacks from tenant VMs due to access privilege issues. External network connects the TVMs to the outsiders through the Internet and is vulnerable to various traditional attacks. There are different roles created for different members in the cloud, such as cloud service provider, cloud administrator, tenant administrator and tenant users. A cloud administrator is an individual employed by a cloud service provider to maintain cloud infrastructure and typically has privileged access. If users want to become tenants (cloud customer/tenant user), they have to register themselves with the service provider. A tenant administrator is responsible for configuring and allocating policies to a set of TVMs and is allocated additional privileges by the cloud administrator. Tenant users can run applications and services in tenant virtual machines.

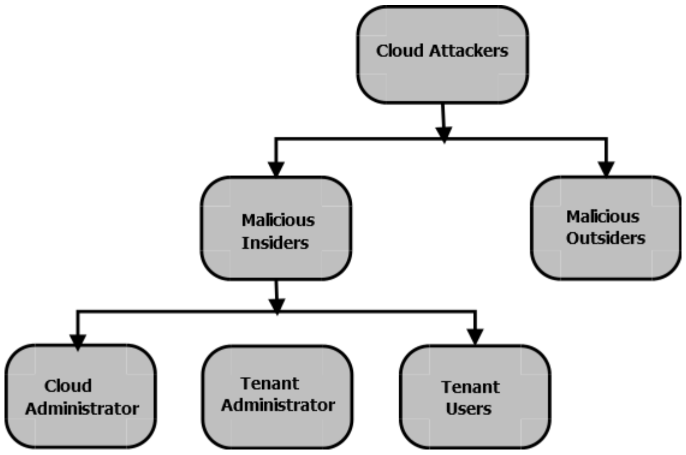


FIGURE 2.2: Classification of attackers in cloud environment

Cloud is vulnerable to attacks targeted against various cloud components. Attacks can be launched by cloud administrators (entities possessing privileges of access and allotment of cloud resources), tenant users, tenant administrator or malicious outsiders (attacking entities outside the cloud environment). The various cloud attackers are shown in Figure 2.2. The description about various attack surfaces and attackers with attack scenarios is given in threat model, described next.

2.1.1 Threat Model

Threat modeling is a critical step in understanding which assets are most likely to be targeted in the cloud environment. It also helps to analyze various vulnerabilities that are present in these assets and how they can be exploited. Let us now consider a typical configuration of our system architecture as shown in Figure 2.3 and discuss the threat model. We propose a threat model in which the direction of attack is represented from source to target machine. The start of direction is represented by a circle that represents origin point of attack and end of direction is represented by an arrow head that represents victim machine. The attack surfaces considered for security analysis in this thesis, are shown in Figure 2.3. Below, we describe different attack scenarios in threat model:

∗ If a malicious user of a TVM is able to access another TVM illegitimately by using various local privilege escalation techniques, he/she can run VM rootkits in the guest machine to gain the root

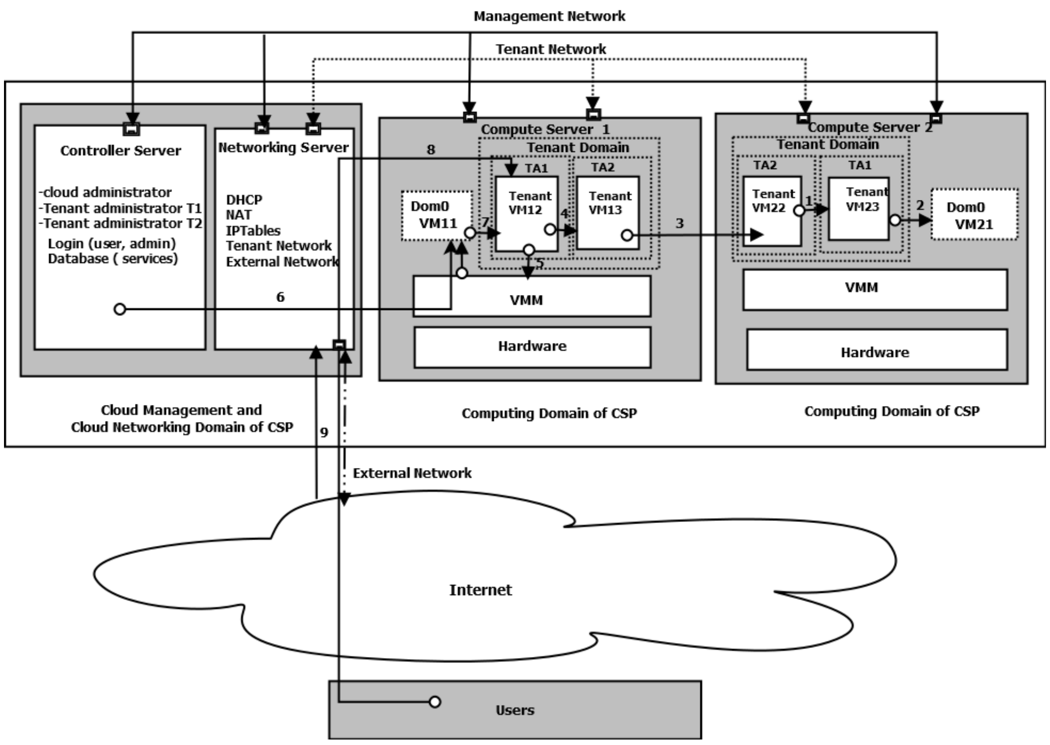


FIGURE 2.3: Proposed threat model in cloud environment

privileges of the guest operating system. Once the rootkits are installed, it is possible to hide the intrusions and run with privileges with an intention to cause harm in the victim VM. In fact some of the advanced malware are evasive in nature and can evade the security tool running in the victim VM (TVM-TVM)(line 1).

∗ A malicious tenant user can also exploit the vulnerabilities present in the operating system of VM and run malicious code (advanced malware) that escape the boundaries of VM [94] and allows an attacker to harm privilege domain (Dom0) of VMM and host OS (VM-VMM/host OS) (line 2). Dom0 is the privileged VM, used to create, destroy, resume, start or stop TVMs and define security policies.

∗ If a tenant administrator has requested free communication between its TVMs (say VM13 and VM22). A malicious tenant can launch the denial of service attacks by starving the resources and crashing the server to other TVM and/or scan the other TVM traffic. If the resources allocated to the other TVMs are exhausted significantly, causing the TVM to be unable to respond to legitimate tenant requests, leading to the violation in service level agreement (SLA) (line 3).

∗ If two tenant VMs belonging to different tenant administrators reside on the same physical server, then they can also become victim of the attacks. For example, a malicious tenant VM (say, VM12 which belongs to tenant administrator 1 (TA1)) can perform scanning of another tenant VM (say VM13 which belongs to TA2) and can obtain detailed information and exploit the vulnerabilities present in the victim VM13 (VM-VM) (line 4). A malicious tenant user can also cause Guest Denial of Service (DoS) attack by starving the resources of the server at the virtualization-layer (VM-VMM) (line 5).

∗ The network intrusions can also be launched in a more complicated way in which the malicious packets are tampered to a spoofed IP and/or spoofed MAC address of victim virtual machine, again leading to disputes between cloud service provider and victim virtual machine (line 3 and line 4).

∗ If a TA has requested free communication between its VMs (say VM13 and VM22) which are placed at different cloud compute servers, then the cloud provider will not monitor their communications. In this case, a malicious tenant user can exploit vulnerabilities in VM13 and launch malware and/or network attacks on other machines (say VM22) (line 3).

∗ There can also be attacks from the cloud administrator on a tenant VM (a powerful malicious insider). Cloud administrator has access to privilege domain (Dom0) of the VMM. A malicious cloud administrator can misuse the information and cause further exploits (lines 6 and 7). This is very rare situation. Generally, CSPs are much interested in their reputation and do not want to destroy their relationship with clients. However, the feasible solution to tackle this problem is to select a most trustable CSP which has been addressed by Habib and Varadharajan [95].

∗ Internet users can also target the cloud infrastructure to gain access to cloud resources and can also use them to generate attacks. Once an outsider is successful in gaining access to a tenant VM, he/she can misuse it to launch further attacks such as flooding and scanning (Outsider-VM) (line 8). The charges incurred by the misuse of resources is paid by the victim tenant, potentially leading to further disputes. In addition, if a tenant VM is compromised, an attacker can generate traffic floods with ICMP/UDP packets having a spoofed source address of another victim tenant VM, making the victim's resources busy. Another form of DDoS is EDoS (Economic Denial of Sustainability) which affects the pricing model of tenant, making him/her suffer from additional billing charges [45]. For example, a malicious tenant user can continuously send requests to websites, hosted in cloud. This causes the bandwidth consumption, which bills to cloud website owner [46].

∗ CNS is responsible for routing and packet forwarding, as shown in Figure 2.3. The packet coming from outside or going to outside, passes though the network server. A CNS can also be subjected to attacks mainly network attacks such as flooding, Denial of Service, scanning, brute force attack etc. (line 9), bringing down the cloud services.

2.1.2 Attacks Taxonomy

An attack taxonomy represents a systematic framework to classify attacks. The attacks are classified based on target-components wise where attacks target specific layers of cloud as shown in Figure 2.4. The brief description of some of the attacks under each category is described below:

(A) VMA: Attacks on Virtual Machine: Virtual machines are most vulnerable part of cloud environment as they are easily accessible by tenant users (line A). Malicious tenants or malicious cloud administrator may perform attacks against VMs. some of such attacks are described below:

VMA1: VM rootkits: VM-rootkit attacks perform the guest OS kernel modification. They hide their presence from the traditional IDS deployed at monitored machine. The detection of such attacks is essential at the primary stage since it can lead to further attacks.

VMA2: Program modification attacks: A malicious user can perform the subversion of specific privileged programs running in TVM to escalate their privileges and launch further attacks on cloud. Subverting files such as those which describe tenant privileges or authentication information can allow cloud users to bypass access control and authentication system.

VMA3: Virus/Worm: Some malicious code is either injected in a normal program or works as an independent code. Virus and worms are

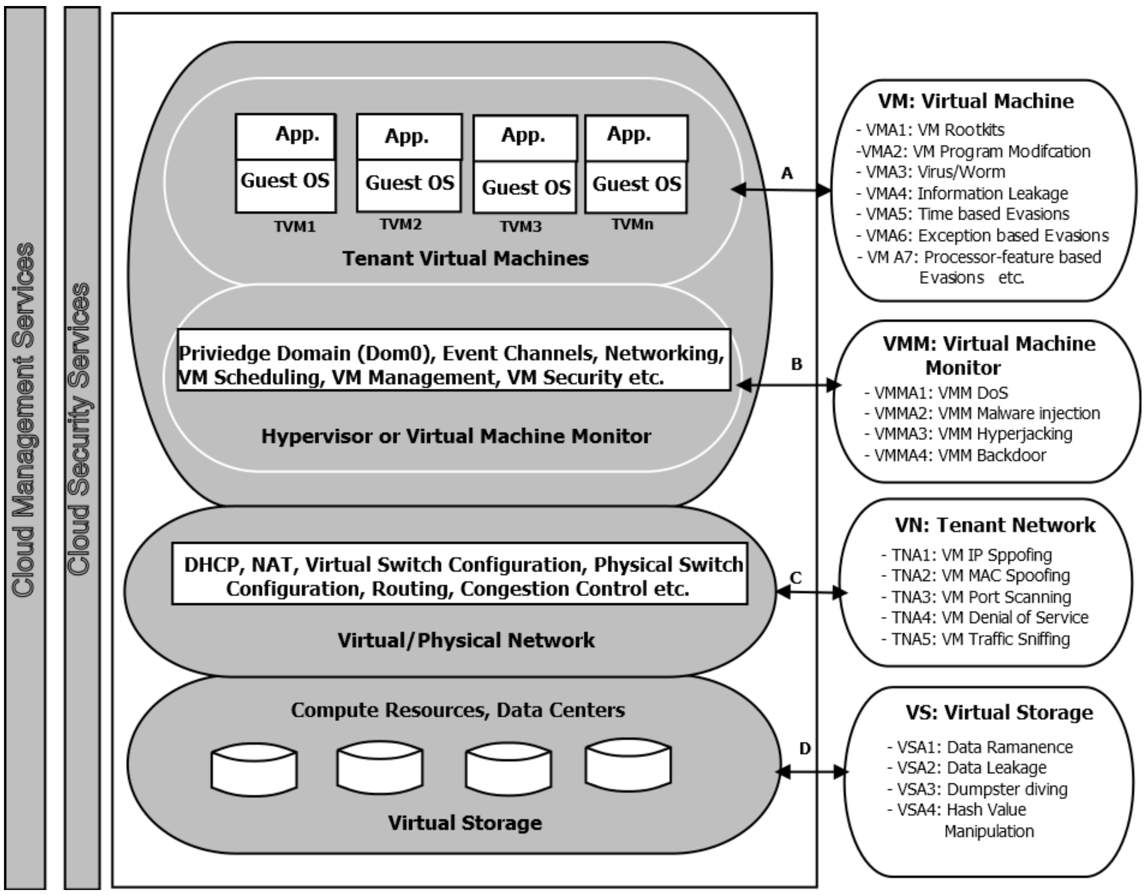

Cloud Management Services
Cloud Security Services
App.
Guest OS
TVM1
TVM2
TVM3
TVMn
Tenant Virtual Machines
Priviedge Domain (Dom0), Event Channels, Networking, VM Scheduling, VM Management, VM Security etc.
Hypervisor or Virtual Machine Monitor
DHCP, NAT, Virtual Switch Configuration, Physical Switch Configuration, Routing, Congestion Control etc.
Virtual/Physical Network
Compute Resources, Data Centers
Virtual Storage
A
B
C
D
VM: Virtual Machine
- VMA1: VM Rootkits
-VMA2: VM Program Modifcation
- VMA3: Virus/Worm
- VMA4: Information Leakage
- VMA5: Time based Evasions
- VMA6: Exception based Evasions
- VM A7: Processor-feature based Evasions etc.
VMM: Virtual Machine Monitor
- VMMA1: VMM DoS
- VMMA2: VMM Malware injection
- VMMA3: VMM Hyperjacking
- VMMA4: VMM Backdoor
VN: Tenant Network
- TNA1: VM IP Sppofing
- TNA2: VM MAC Spoofing
- TNA3: VM Port Scanning
- TNA4: VM Denial of Service
- TNA5: VM Traffic Sniffing
VS: Virtual Storage
- VSA1: Data Ramanence
- VSA2: Data Leakage
- VSA3: Dumpster diving
- VSA4: Hash Value Manipulation

Figure 2.4: Classification of attacks under consideration in cloud

some examples of malware that can corrupt important system or user files, and can even crash an OS by replicating themselves. An attacker can inject the malware in a target VM to gain root access of machine and later launch further attacks.

VMA4: Information Leakage: VM memory information may leak in different ways. A malicious user can install a key logger program in the victim machine in an illegitimate way. Upon execution, the credential information of the victim user is transferred to the attacker, leading to disclosure of tenant's sensitive information.

VMA5: Time based evasions: Automated malware analysis systems have to detect the malicious activities within a limited time frame. If a program does not behave maliciously within 5-10 minutes, such program will be classified as benign or normal. In time based evasions, malicious program perform the malicious activity slowly not exceeding the time window. For example, an attacker can put the stalling code in the malware [96]. Stalling codes are loops containing timing operations to delay the operations. The execution of stalling code is placed before the actual malicious code and hence automated malware analysis system fails to detect such evasions. In addition, there are many introspection techniques such as SIM [68] that place the security hooks inside the guest VM/TVM and provides VM monitoring from security VM. They perform the detection in regular interval of time. An attacker can evade detection by observing the periodic behavior of analysis tool and can attack between monitoring checks [89].

VMA6: Exception based evasions: Exception based evasions can be another attempt to detect the analysis environment. For example, some dynamic analysis tools insert the custom codes in the TVM memory. By properly reading the context structure that is passed as a parameter to a custom exception handler; the context of debug register can be read which reveals the information about these custom codes. This is explained in details by Fratantonio et al. [97]. Such kind of evasions can be used to fool the analyzer.

VMA7: Processor-Specific evasions: Processor based evasions use CPUID instruction to extract the processor-specific features such as Physical Address Extension (PAE), Page Size Extension (PSE) and Virtual Machine Extension (VMX), Hypervisor etc. values depending on the contents of EAX and ECX registers. The information can be helpful to detect the analyzer (if attacker has the knowledge of analyzer) and can also reveal the presence of Hypervisor. For example, some analyzer modifies the default values of PAE and PSE to make the memory writes easier in hardware-assisted virtualization platform [98]. In addition, CPUID instruction executed at TVM with EAX=0 returns a 12 character ASCII string which is stored in EBX, EDX, ECX (in order). If it is "KVMKVMKVM": presence of KVM is detected; if it is "XenVMMXen" : presence of Xen is detected [99]. The detection of such evasion is essential before they can launch Hypervisor-specific attacks.

VMA8: VM Escape: VM Escape is an attack in which an attacker gains access to the memory that is beyond reach of compromised tenant VM (memory of other guest VM, VMM or even host OS). VM escape can also be described as the process breaking out of a VM and interacting with VMM/host operating system. Attacker can further breach the isolation of VMs and can cause harm to other guest VM [94].

(B) VMMA: Attacks on Virtual Machine Monitor: Attacker may exploit the programming code vulnerabilities present in the VMM (line B) and cause harm to VMs under its control. The description of some of the attacks is given below:

VMMA1: VMM DoS: VMM operations can be badly affected by resource starvation (such as RAM, CPU and network Bandwidth), or interrupted if a complete shut-down of the VMM is caused or restarting it every time for the new services in VM. These situations cause DoS. It will affect all VMs running over it [9].

VMMA2: VMM Malware Injection: The virtual environment can be detected by malware which can disable or infect critical components such as the VMM. A malware can roll back the VMs and can restore them to previous states [9].

VMMA3: VMM Hyperjacking: Hyperjacking attack refers to installing a rogue Hypervisor that can take complete control of a server. Rootkits based malware can cause hyperjacking attack. Hypervisor-level rootkits exploit the hardware virtualization technologies such as Intel VT or AMD-V. They run with high privileges than Dom0 and host of the target VMs. Bluepill and SubVert are examples of it [100].

VMMA4: VMM Backdoor: An attacker can take a backdoor entry into Hypervisor's privilege domain by overwriting the Hypervisor code and manipulating the kernel data structures of guest OS; leaving all bytes in privilege domain unchanged. Detection is difficult if conducted from privilege domain. Wojtczuk et al. [20] implemented two backdoors: The first one resides in the Hypervisor code and the another one resides in a hidden domain with artificially elevated privileges.

(C) TNA: Attacks on Tenant Network: Tenant's traffic flows from one VM to another VM through tenant network (via physical/virtual network interfaces). The co-located VMs communicate via virtual network interface at the Hypervisor. The non co-located VMs communicate via both virtual and physical interfaces at Hypervisor and cloud server respectively. The attacks on the tenant network exploit the networking layer vulnerabilities of cloud (line C). Following are some of the attack examples:

TNA1:VM Traffic IP spoofing: A VM which is launched at certain network segment is open to attacks from other VMs launched on the same network. A malicious user can perform IP spoofing from one VM in which attack traffic is generated on behalf of legitimate tenant machine (victim) and is sent to destination VM. [101].

TNA2:VM Traffic MAC spoofing: A malicious user can perform MAC spoofing from one VM in which attack traffic is generated with the MAC address of the victim VM [101], misleading the other VMs.

TNA3: VM Port scanning: Port scanning is an attack that does not cause any harm on the VMs, but it gives the attacker some specific information about the status of the ports that can be used for further attacks such as DoS attacks. There are many scanning tools such as hping3 [102] and nmap [103] which can be applied in virtualized environment as well.

TNA4: VM Denial of Service: At network level, DoS attack is accompanied by IP spoofing and flooding. Attacker floods the broadcast address with spoofed packets. The sender address is target VM's IP address providing a service on the cloud. On receiving the packet, each node responds to the server hosting the VM with particular spoofed IP. This causes consumption in the victim's resources so that it can no longer provide its intended service. If a DoS is initiated from more than one source controlled by a master node then it is referred to as Distributed Denial of Service (DDoS) attack. TCP-SYN flood, ICMP flood, UDP flood are specific types of DDoS attacks [101].

TNA5: VM Traffic sniffing: In the environment where VMs are connected via virtual switches, packet sniffing is done at the virtual switch level. Hypervisor links the VMs via bridge or router and provide logical isolation between the VMs. Physically the VMs share the same hardware resources. Attacker can exploit this vulnerability in sniffing the virtual network to gain sensitive information of VMs.

(D) VSA: Attacks on Virtual Storage: Tenants share the same physical storage, divided into a large number of logical units depending on the storage capacity. Such a multitenancy can cause following major attacks at storage-layer of cloud environment (line D):

VSA1: Data Remanence: Data Remanence represents residual information of the data after its deletion. Various file handling operations such as the reformatting of storage and deletion operation may result into data remanence. Such operations could cause disclosure of user's sensitive information. In a cloud environment, virtual storage is shared by different tenants. If a tenant removes his/her data, the virtual private sector is reallocated to another tenant user of the same service provider.

With the technical expertise, the bits of information allocated to earlier data can be interpreted by enterprise hackers [104].

VSA2: Data Leakage: Attacks such as password guessing and dumpster diving can lead to VM data leakage. Attackers can also use tools such as key logger to gain authentication into target VM and can breach the data. VM backdoors [6] can also leak sensitive information such as VMware I/O backdoor [105].

VSA3: Dumpster Diving: Dumpster diving is an attempt of digging out information from data which was discarded as waste. The data is recovered by the attacker that was discarded by cloud users or admin to gain useful information from it. An attacker can also target a specific user who has shared his/her data with the cloud. Discarded information may include authentication information, cookie information, credit card details, other technical manuals or organizational chart [104].

VSA4: Hash Value Manipulation: An attacker may manipulate the hash value of the message and can get authorized access to the file stored in the server. If manipulated hash value exists in the database, server links the file to that hash value. If the modified hash value does not exist, server request a file from the user. The vulnerability may exist if the server does the calculation using OpenSSL by using wrapper class library Ncrypto. Attacker may modify Ncrypto, which is publicly available [106].

2.2 Taxonomy of Intrusion Detection Techniques in Cloud

The state of the art IDS techniques for detecting attacks against the virtual domains running in cloud environment, are presented. We have classified techniques of intrusion detection for dealing with attacks at different cloud layers (TVM/VMM/Network) into four types: (i) Misuse Detection (ii) Anomaly Detection (iii) Virtual Machine Introspection (VMI) (iv) Hybrid techniques as shown in Figure 2.5. Each technique is further classified based on the detection approach employed and described under each subsection.

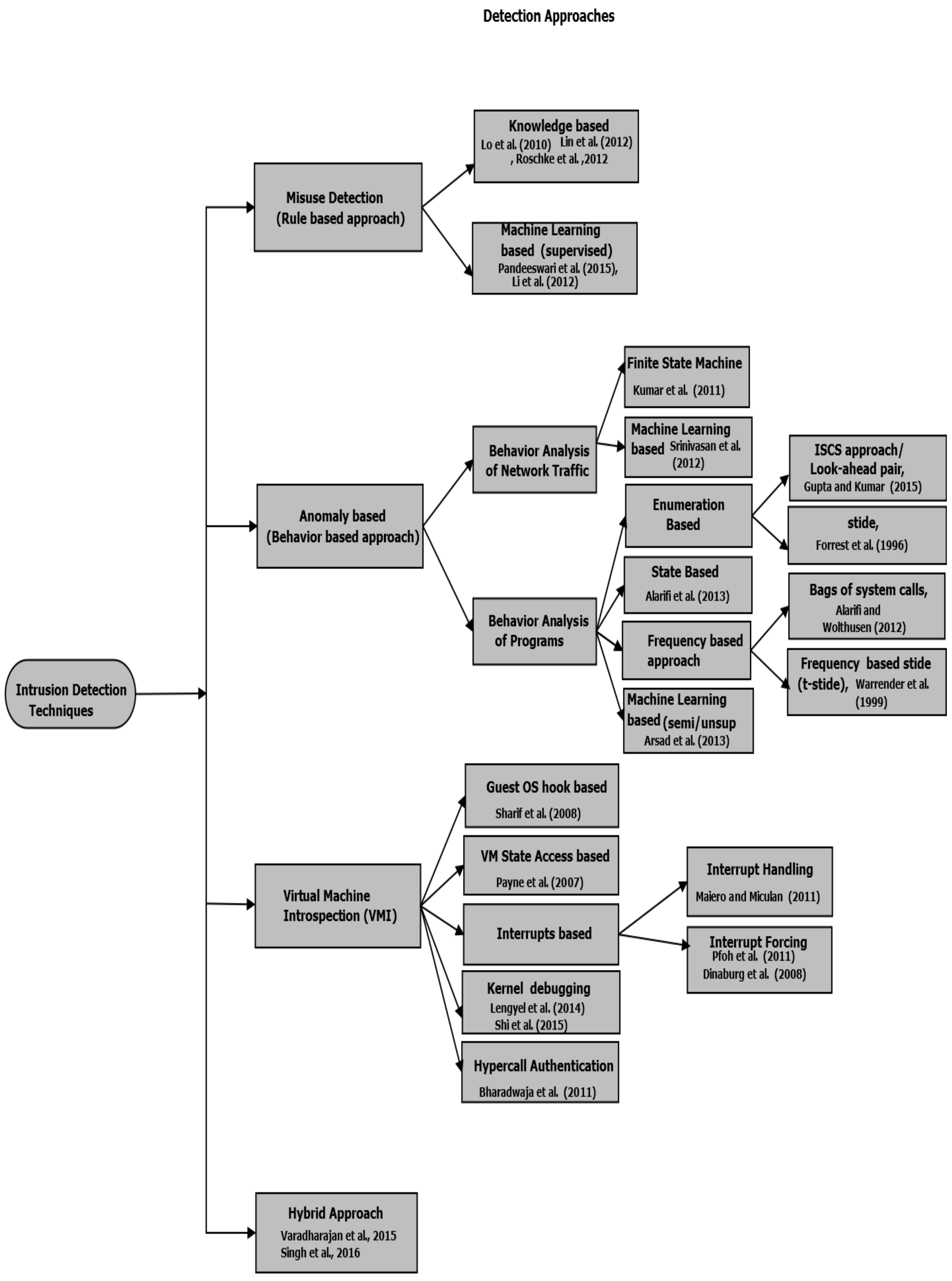

Detection Approaches
Intrusion Detection Techniques
Misuse Detection (Rule based approach)
Knowledge based Lo et al. (2010) Lin et al. (2012) , Roschke et al. ,2012
Machine Learning based (supervised) Pandeeswari et al. (2015), Li et al. (2012)
Anomaly based (Behavior based approach)
Behavior Analysis of Network Traffic
Finite State Machine Kumar et al. (2011)
Machine Learning based Srinivasan et al. (2012)
Behavior Analysis of Programs
Enumeration Based
ISCS approach/ Look-ahead pair, Gupta and Kumar (2015)
stide, Forrest et al. (1996)
State Based Alarifi et al. (2013)
Frequency based approach
Bags of system calls, Alarifi and Wolthusen (2012)
Frequency based stide (t-stide), Warrender et al. (1999)
Machine Learning based (semi/unsup Arsad et al. (2013)
Virtual Machine Introspection (VMI)
Guest OS hook based Sharif et al. (2008)
VM State Access based Payne et al. (2007)
Interrupts based
Interrupt Handling Maiero and Miculan (2011)
Interrupt Forcing Pfoh et al. (2011) Dinaburg et al. (2008)
Kernel debugging Lengyel et al. (2014) Shi et al. (2015)
Hypercall Authentication Bharadwaja et al. (2011)
Hybrid Approach Varadharajan et al., 2015 Singh et al., 2016

Figure 2.5: Proposed Classification of IDS techniques in Cloud Environment

2.2.1 Misuse detection techniques

Misuse detection techniques match the current behavior recorded in log files with known attack patterns. The conceptual diagram for misuse detection techniques is shown in Figure 2.6. Data is collected from various sources such as packet capture files. The collected raw data is first pre-processed and converted into a useful format before passing it to the Detection Engine. The Detection Engine implements a decision model which decides whether to pass the data or generate an alert based on some known behavior. The known behavior can be in the form of a signature or attack profile generated by machine learning algorithm over a labeled dataset. If the current system activity matches with the known attack behavior, an alarm is raised to concerned authority (cloud administrator/tenant). In the virtualization-layer, misuse detection systems are deployed at the tenant VMs and able to detect outsider attacks targeting VMs and some Inter VM attacks (VM-VM). Misuse detection techniques [64] can be further classified into two types: Knowledge based approaches and Machine learning based approaches, which are briefly explained in detail below:

(i) Knowledge based approaches: In knowledge-based approaches, network traffic or host audit data such as system call traces are compared against predefined rules or attack patterns. State transition analysis, signature-based analysis, rule-based expert systems are some examples of knowledge-based system [107]. Signature-based analysis maintains a database of rules for detecting different types of known attacks. The incoming packets are scanned against fixed patterns. If any of the patterns matches with the packet header, the packet is flagged as anomalous. State transition analysis maintains a state transition model of the system for the known suspicious patterns. Different branches of the model lead to a final compromised state of the machine. The rule-based expert system maintains a database of rules for different intrusive scenarios.

(ii) Machine learning based approaches: Misuse detection can also be performed using supervised classification algorithms such as Back Propagation ANN (BP-ANN)[108], Decision Tree (DT) [109] and Multi class Support Vector Machine (SVM) [110] which are well known machine learning approaches. Machine learning approaches for misuse detection provide a learning based system to classify attacks based on learned

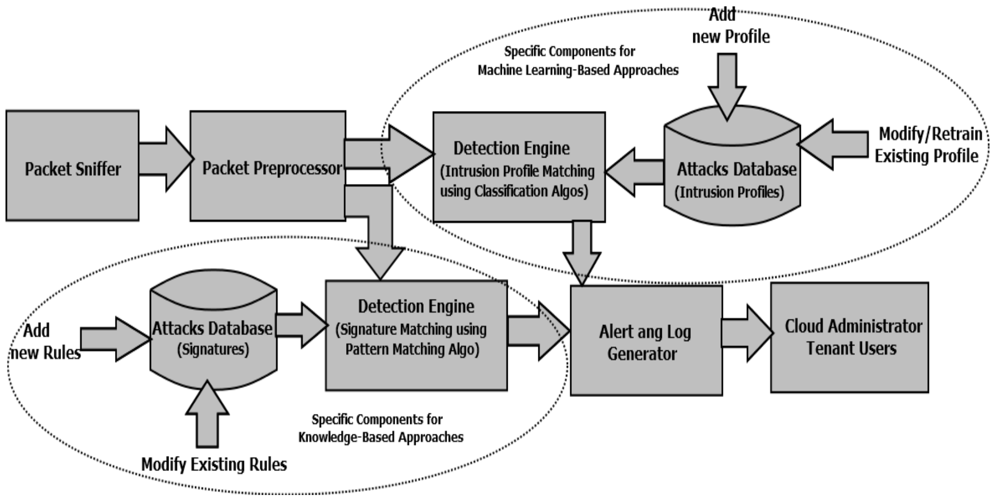

Add new Profile
Specific Components for Machine Learning-Based Approaches
Packet Sniffer
Packet Preprocessor
Detection Engine (Intrusion Profile Matching using Classification Algos)
Attacks Database (Intrusion Profiles)
Modify/Retrain Existing Profile
Add new Rules
Attacks Database (Signatures)
Detection Engine (Signature Matching using Pattern Matching Algo)
Alert ang Log Generator
Cloud Administrator Tenant Users
Specific Components for Knowledge-Based Approaches
Modify Existing Rules

FIGURE 2.6: Conceptual diagram for knowledge-based and machine learning based approaches

normal and attack behaviors. It uses network traffic or host audit data such as system call traces. The goal of machine learning methods is to generate a general representation of attacks (host/virtual based or network attacks). The attack behavior is automatically induced in the representation rather than predefined. The algorithms are used to find the inherent associations in the data or any irregularity with the data.

2.2.2 Anomaly detection techniques

Anomaly based techniques are based on making a behavior profile of system and keep on updating it time to time. Any deviation from the learned behavior flags the suspicious activity in the network/system. They are different than misuse detection as they do not have any knowledge of the attack patterns. Hence, the techniques can detect zero-day attacks. A zero-day attack refers to exploitation of a vulnerability that has not been known earlier. Researchers have proposed anomaly detection systems for cloud using: network behavior analysis and program behavior analysis techniques. The program behavior analysis can be: dynamic analysis (run-time behavior capturing) and static analysis (program code analysis) as shown in the conceptual diagram in Figure 2.7.

(i) Behavior analysis of network traffic: Network traffic analysis techniques capture the incoming and outgoing packets from the system. The traffic features such as port number, source address, destination address, service and the number of connections to same destination

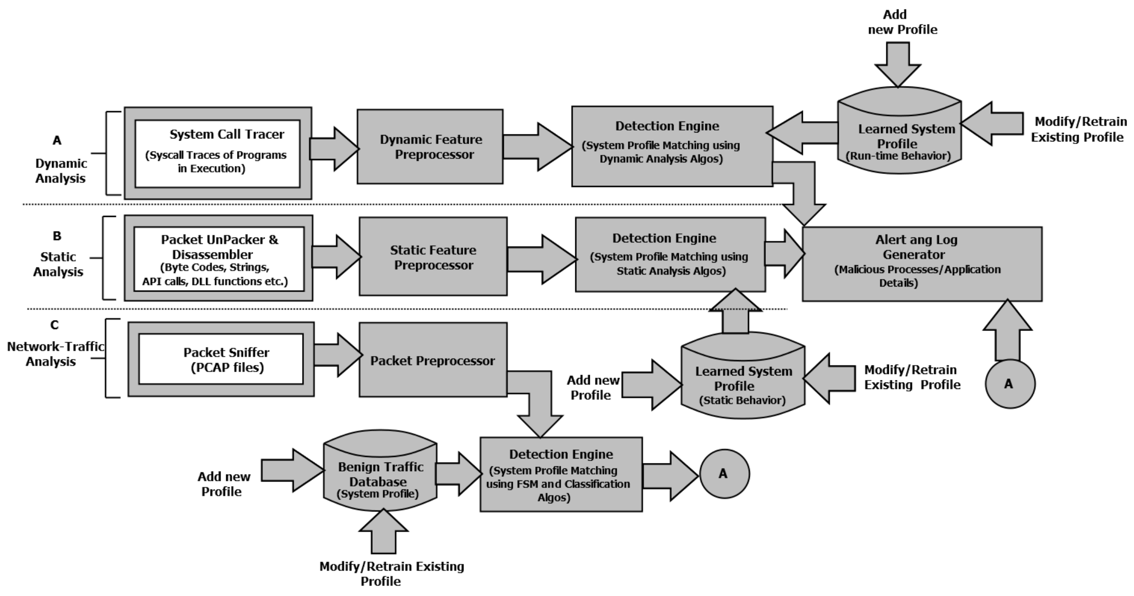

A
Dynamic Analysis
System Call Tracer
(Syscall Traces of Programs in Execution)
Dynamic Feature Preprocessor
Detection Engine
(System Profile Matching using Dynamic Analysis Algos)
Add new Profile
Learned System Profile
(Run-time Behavior)
Modify/Retrain Existing Profile
B
Static Analysis
Packet UnPacker & Disassembler
(Byte Codes, Strings, API calls, DLL functions etc.)
Static Feature Preprocessor
Detection Engine
(System Profile Matching using Static Analysis Algos)
Alert ang Log Generator
(Malicious Processes/Application Details)
C
Network-Traffic Analysis
Packet Sniffer
(PCAP files)
Packet Preprocessor
Add new Profile
Learned System Profile
(Static Behavior)
Modify/Retrain Existing Profile
A
Add new Profile
Benign Traffic Database
(System Profile)
Detection Engine
(System Profile Matching using FSM and Classification Algos)
A
Modify/Retrain Existing Profile

FIGURE 2.7: Conceptual diagrams for (A) dynamic analysis (B) static analysis and (C) network traffic analysis approaches

are the basic attributes in building the normal profile of the virtual/-physical system. Anomaly detection techniques based on analysis of network traffic are widely categorized into three types: statistical approaches, machine learning based approaches and finite state machine (FSM) based approaches [65]. However, we have considered only FSM based and machine learning based approaches in our taxonomy as they are the ones used by researchers working in cloud security. They are briefly described in detail below:

Finite state machine based approaches: A finite state machine (FSM) produces a behavioral model which consists of states, transitions and actions. A state stores the information about the past, a transition represents change from one state to another on occurrence of an event and action represents the response to be taken [111].

Machine learning based approaches: Anomaly detection can also be performed using semi-supervised and unsupervised algorithms such as self organizing map neural network [112], clustering algorithms [113] and one class SVM [114]. Machine learning approaches for anomaly detection provide a learning based system to discover zero-day attacks.

(ii) Behavior analysis of programs: Network Traffic analysis is not sufficient to detect all kinds of attacks. Behavior analysis of programs is helpful in detecting low frequency attacks such as rootkits and VM Escape. Behavior analysis techniques are categorized into two classes: Dynamic Behavior Detection (DBD) and Static Behavior Detection (SBD). Dynamic analysis and static analysis approaches are discussed in brief below:

(a) Dynamic Behavior Detection: Dynamic Behavior Detection (DBD) approaches for anomaly detection, collect the system calls of the monitored programs in normal execution scenarios and make a baseline profile for further analysis. The techniques assume that malicious programs invoke a malicious sequence of system calls which deviate from the normal execution scenario. The dynamic behavior of a program remains the same even if code is obfuscated. The approaches are categorized into four types: enumeration-based approach, frequency based approach, state-based approaches and machine learning based approach. We have included all four classes in our taxonomy as each of the them has been applied for anomaly detection in cloud. They are explained in brief detail below:

Enumeration based approaches: Enumeration based approaches [115] make an ordered list of system call sequences of monitored programs. The ordered list of system calls followed by each unique system call, for the normal programs forms the baseline database for intrusion detection in future. Researchers [84] have used this concept for detecting anomalies in cloud for in-guest/host monitoring.

Frequency-based approaches: A rare pattern may occur because of sudden misuse behavior of a normal user such as invoking the unwanted function, overflowing the buffer or any unexpected error. Just by mismatches, we cannot be sure about the anomalous behavior of a trace. In some cases abnormal sequences may be present in a database and their frequency of occurrences may not be too high. Hence, a further step to compare the mismatches with a predefined threshold value [116]. Frequency based methods found to improve the performance of earlier enumeration approaches, based on just pattern matching. Some researchers have applied this concept in cloud [117] for in-guest monitoring.

Finite state machine based approaches: FSM based approaches have also been used for detecting malicious programs sequences. Cho et al. [118] presented an anomaly detection technique based on Hidden Markov Model (HMM). The authors have observed that HMM takes an unusually longer time to build the model of all normal user processes.

Therefore, they have improved the efficiency of HMM by modeling it only for privileged programs.

Machine learning based approaches: Machine learning techniques are applied with the existing techniques to provide a generic model which represents the normal system call patterns of the programs. Researchers [119] have applied machine learning for intrusion severity analysis at VMM.

(b) Static Behavior Detection: Static Behavior Detection (SBD) approaches perform the analysis of program codes and generation of the behavioral profile of programs without running them. Static analysis approaches try to capture the possible behavior of programs by analyzing the programming codes whereas dynamic analysis approaches capture the actual run-time behavior of the program. They are categorized into two types: Specification based and Profile based. They are discussed in brief below:

Specification based approaches: A specification of program predefines its intended behavior and is defined in terms of policies. Researchers have proposed specification policy based solutions [120] to specify the behavior of each program (such as rdist) to prevent illegitimate actions. The policies are precise and clear; however they will increase the overhead of cloud administrator to set specification policies for all privilege programs running in different cloud servers. Writing such a policy itself is a tedious, time-consuming task, and may prone to programming errors which requires the specialized knowledge of program functions.

Profile based approaches: Profile based approach refers to making a generic behavioral profile of the system based on the static analysis of program behavior. The generic profile, first extracts the required information (features) such as byte-code sequences, DLL functions calls, hexdump codes, function names etc. from the programs and then applies machine learning to the extracted features of monitored programs to distinguish between normal and anomalous behavior [121].

Static analysis approaches are very older techniques, proposed in traditional environment. They fail to detect the modern malware attacks which hide/change their behavior from security tool on detection of monitoring tool or virtualziation environment. However, the existing antivirus tools based on static analysis (reverse engineering of code) can be directly used by tenants. To provide a secure monitoring place for

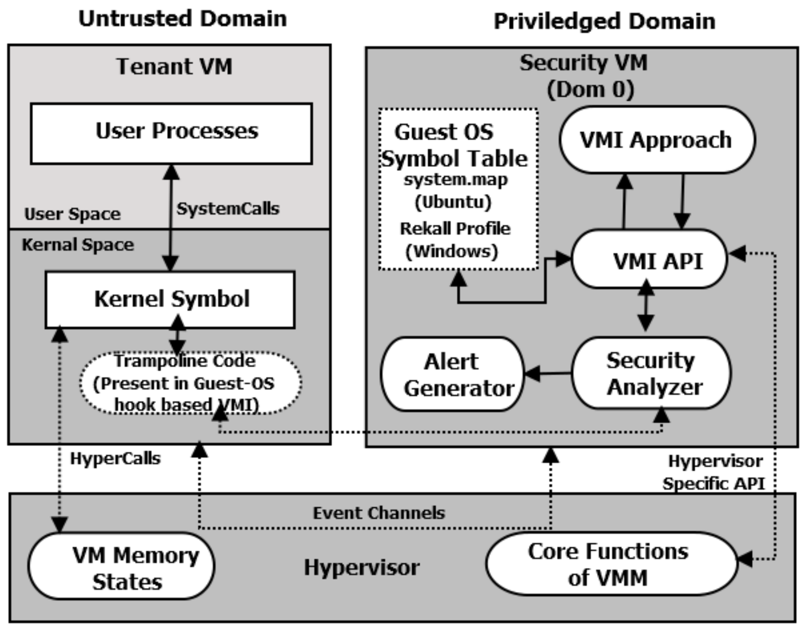


FIGURE 2.8: Conceptual diagram for VMI-based approaches

security analyzer, VMI based techniques have been proposed, discussed in next section.

2.2.3 Virtual Machine Introspection (VMI) techniques

Advanced malware programs are intelligent to detect the malware analysis components running in the monitored machine. They try to disable or compromize the security tool without leaving any trace in the system. Hence it is essential to monitor the VMs by placing the security monitor at VMM. VMI techniques are specifically designed to provide VM introspection from outside the tenant VM (generally deployed at security VM (Dom0)). The techniques utilize the features of the Hypervisor and introspection libraries such as LibVMI [122] which uses guest symbol table to access the VM memory regions as shown in Figure 2.8. VMI approaches provide the high-level view of the VM memory which is analyzed by security analyzer working in security VM (Dom0). For most of the VMI techniques, all the security analysis code run in the security VM. However, some approaches place a security module (trampoline code) in the monitor VM which communicates the information to other security modules running in security VM. If any of the VM memory region is found to be suspicious, security analyzer generates an alert to the cloud administrator. VMI approaches are helpful in cloud environment where the provider wants to monitor the VM behavior from the VMM. VMI-IDS are deployed at the VMM or in a privileged VM. They can detect VM-VM attacks, insider VM attacks and especially VM-VMM attacks. We have categorized them into five types based on the introspection approach used: (i) Guest OS hook based (ii) VM state access based (iii) Hypercall authentication based (iv) Kernel debugging based (v) Interrupt based. Each of the VMI based approaches is discussed below:

Guest OS hook based approaches: Guest OS hooks based VMI approaches inject hooks (In-VM agent) into the kernel of guest OS. These hooks are kernel modules which send the required information to VMI applications. These methods require modification of guest OS and are less attack resistant than out VM approach [91].

VM state access based approaches: VM state access based approaches rely on the state information of VMs provided by VMM. A VM state is defined by its memory space, CPU registers, I/O access etc. VMI applications derive semantic knowledge and intelligence about a guest OS from low level details of VM state information [123].

Hypercall authentication based approaches: As syscall is a software trap from an application to kernel; similar to that a hypercall is also a software trap which is generated from a virtual machine to the Hypervisor. Virtual domains use hypercalls to request privileged operations like updating pagetables. A hypercall integrity is checked in these type of solutions [72].

Kernel debugging based approaches: These method use the kernel debugging data to extract the location of kernel functions and perform break point injection at desired locations. An interrupt is generated when break point is executed and kernel functions are trapped to collect necessary information about the process [124].

Interrupt based approaches: Interrupt based VMI approaches trap the monitored function calls at the same time when an interrupt is generated by target machine. These techniques are categorized into two types: Interrupt handling based and Interrupt forcing based. Below, we describe each of the approaches in brief:

∗ *Interrupt handling based:* Interrupt handling based VMI approaches trap the general purpose interrupts such as protection

Table 2.1: Summary of IDS Techniques

Technique Name	**Approach**	**Characteristics**	**Limitations**
Signature based	Misuse	-Maintains database of known attack signatures -Vulnerable to Zero day attacks	-Performance is good for known attacks -Requires regular maintenance of attack signatures
Profile based	Misuse, Anomaly	-Maintains behavioral profile of users (normal and/or suspicious) -Good for detecting unknown attacks	-Requires retraining for new cloud user behavior -More false positives
ISCS (key value pair)	Anomaly	-Maintains a log of immediate ordered list of system calls for each system call -Can detect unknown attacks	-More storage requirement -More false positive -Low performance for infinite or long traces
STIDE	Anomaly	-Maintains a log of fixed size short sequence of system calls -Can detect unknown attacks	-No consideration for frequency counts of n-grams -Performance is average
TSTIDE	Anomaly	-Similar to STIDE but Considers the frequency count of n-grams -Performance is good	-Needs Various tweeks in setting threshold value -Requires maintenance of log for new applications
Bag of System calls	Anomaly	-Maintains a log of frequencies for each system call -Low storage requirement	-Less Attack resistant as it losses order of system calls -High false positives
Hidden Markov Model	Misuse, Anomaly	-Maintains state transition diagram for program execution -Based on probability calculation of output	-Requires prior information of total number of states -Huge time in retraining
Specification based approach	Anomaly	-Maintains specification policy for each monitored program -Each specification defines intended behavior of program	-Overhead in writing and maintaining program policies -Needs program expertise and skills
CFG based	Anomaly	-A CFG of program is prepared where each branch corresponds to possible system call execution -Uses Machine learning to Learn the patterns	-Vulnerable to code obfuscation techniques -Requires maintenance of CFGs on every program change
Program features based	Anomaly	-Extracts the program features such as byte code sequences DLL function calls etc. -Uses Machine learning to Learn the patterns	-Vulnerable to code obfuscation technique -Program analysis is compiler and OS specific
Guest OS hook	VMI	-Hooks are kernel modules injected in the Guest OS to gain useful VM information which is passed to Out-VM component	-Vulnerable to kernel rootkits -Compromised hook may give incorrect VM information
VM state Access	VMI	-Gains semantic information of VM from VM state information such CPU registers, memory stack to gain Guest VM information.	-Vulnerable to kernel rootkits -Compromised VM will give incorrect state information
Kernel Debugging	VMI	-Uses Break point injection technique to trap kernel functions -kernel debugging data is used to trace the kernel location	-Needs expertise knowledge of kernel modules -Limited support to trap user level functions
Hypercall Authentication	VMI	-Hypercalls are authenticated based on the MAC code and origin verification -Hypercalls are classified based on threshold score	-Needs MAC generation scheme for each hypercall -Limited to paravirtualized systems
Interrupt Forcing	VMI	-Forces traps on occurrence whenever a system call (syscall/SYSENTER) is executed -VM information is gained from CPU registers	-Needs hardware virtualization support (Intel VT x) -Information interpretation needs expertise
Interrupt Handling	VMI	-VM information is gained from CPU registers on occurrence on general interrupts such as 0x2e or 0x80 depending on OS	-Overhead in keeping track of all interrupts -Information interpretation needs expertise

exception, page fault and VM_EXIT events; information about the monitored machine from processor context (register such as %eax) is used to perform the active monitoring of VM [69].

∗ *Interrupt forcing based:* VMI approaches force the interrupts (traps, exceptions, etc.) by using the hardware-based hooks (e.g. unsetting setting specific register value and flag value) and derive the information about monitored machine from processor context to do active monitoring of VM [75].

VMI techniques provide a rich set of functionalities to introspect VMs from outside the VM. Researchers are trying to adopt it for intrusion detection applications in cloud environment. There are some other techniques such as Hypervisor introspection (HVI) approaches. HVI techniques addresses the attacks below virtualization layer particularly hardware attacks which are used to bypass the Hypervisor security and are not under the scope of this thesis.

2.2.4 Hybrid techniques

Misuse detection and anomaly detection techniques are combined to improve the effectiveness of the detection engine. However, anomaly and misuse detection techniques are not sufficient enough to detect intrusions at Hypervisor-level. VMI techniques utilize the features of the Hypervisor to gain access inside a VM and provide useful information to other IDS techniques. Hybrid techniques refer to the combination of more than one intrusion detection techniques which make use of the synergy of misuse based, anomaly based and introspection techniques to detect intrusions in cloud.

The above discussed techniques require the high-level semantics of the monitored machine. A brief summary is shown in Table 2.1. The high-level view of TVM can be easily obtained for misuse and anomaly detection approaches as they can be directly applied for in-guest monitoring at the TVM-layer. The security tool performs well when it works inside the monitored machine as it has all contextual information of the machine. However, IDS subversion is easy in such a scenario, as TVM has less attack resistance capability. Hence, this led to works where security tools are placed outside the TVM at VMM-level. However, it is not that easy to get the high-level view of the TVM and get the complete view of the system to provide strong security measures and detect the attacks at the VMM. Various OS features such as demand paging, multi-threading, parallel computing makes this task more complex. The problem of interpreting low-level bits and bytes and transforming it to high-level semantics is called as semantic gap problem [125]. The major challenges for monitoring the TVM from outside is to overcome the semantic gap problem and provide a high level semantic view of the system. VMI techniques prove to be powerful in filling this gap and providing the high-level view of the TVM memory. However, VMI alone is not sufficient. VMI functions need to be integrated with good detection mechanism, leading to works towards Virtual Machine Introspection based IDS. However, TVM monitoring from inside or outside are mainly intended for detecting host based anomalies. Some authors [126] have addressed the issue of network intrusion detection by using the above discussed approaches for misuse, anomaly and hybrid detection. The categorization of various IDS security proposals is described below.

2.3 Classification of Intrusion Detection Systems in Cloud

Researchers working in the field of cloud security have proposed intrusion detection systems (IDSes) incorporating the suitable detection mechanisms, discussed in previous section, in their security architecture for cloud. The existing intrusion detection system (IDS) for cloud have been categorized into four major categories [127]: (i) TVM-based intrusion detection system (ii) Hypervisor-based intrusion detection system (iii) Network-based intrusion detection system (iv) Distributed intrusion detection system. TVM based IDS monitors the audit logs of specific host or guest machine to detect intrusions. It analyzes the audit logs of specific guest or host machine. Hypervisor-based IDS monitors the specific set of guest machines (TVMs) from Hypervisor for the presence of any abnormal activity. It introspects the audit logs of TVMs from outside. Network-based IDS monitors the network traffic logs to detect intrusions. It examines the network packet header information of ingress and outgress tenant network traffic. Distributed IDS monitors the guest, network and/or host by placing IDS instances at different locations which are controlled by central administrator and/or communicate with each other. Each of the key IDS proposals under each of the category with the details of their detection mechanism have been discussed in subsequent sections.

2.3.1 TVM-based Intrusion Detection System

TVM-based IDS analyzes the specific actions of the guest by monitoring the interaction between user/system applications and guest operating system. The existing traditional security solutions for host monitoring are applicable to traditional physical server and guest machines (TVMs) at SaaS/PaaS/IaaS cloud environment. Some of them have been applied by researchers for cloud environment at TVM-layer. They can not be directly adopted at VMM-layer because of some technical issues associated with different layers, discussed in Section 2.4. The details of some of these security proposals for TVM monitoring are discussed below.

Initially, Ko et al. [120] propose intrusion detection approach based on the specification of programs. A specification of program pre-defines its intended behavior and is defined in terms of policies. Hackers can exploit privilege programs. For example, `rdist` (Remote File Distribute) program that is used to maintain consistency of files, distributed in multiple hosts, can be exploited in the following way: A normal user triggers `rdist` to update one of his files residing on local host. Afterwards, a temporary file is created by `rdist` before finishing copying. An attacker can rename the temporary file and create a symbolic link with the same name as the temporary file. After finishing copying, when `rdist` changes the permission of the temporary file using chmod command, the ownership of symbolic link is also affected. Hence, an attacker can access the temporary file even after `rdist` finishes copying. Authors have proposed specification policy to specify the behavior of each program (such as rdist) to prevent illegitimate actions. Although program policies are concise and clear, it is not easy to write such specifications as it requires the specialized knowledge of program function.

Specification based approaches are not suitable for the cloud environment. They will increase the overhead of cloud administrator to set specification policies for all privilege programs running in different cloud TVMs. Writing such a policy itself is a tedious and time-consuming task and may be prone to programming errors. It also requires regular maintenance of the policy database.

There also exist enumeration based approaches, used for doing the program behavior analysis using system calls. Forrest et al. [85] initiated the use of system calls many years ago. Initially, they propose look-ahead pair based approach in which a database of system calls is prepared in the form of two values. Each entry in the database represents a system call, and an immediate sub-sequence of system calls in a window of size n. All unique system call entries are recorded. The sliding window moves by one position each time, and various system call entries are recorded with their immediate sub-sequences. The database serves a baseline for future system call traces. In the extended work [115], they proposed Sequence Time Delay Embedding (STIDE) approach based on the analysis of the short sequence of system calls. The approach observed that fixed-length contiguous sequences have better-discriminating power than look ahead pairs. A sliding window of fixed size is used to produce short sequence of system calls. The approach is found to perform better than look-ahead pair approach. However, STIDE is a very older enumeration based detection mechanism which maintains a large database of normal system call sequences and is prone to more false alarms for evolving normal behavior.

Warrender et al. [116] extended the above methodology, STIDE [115]. They added a frequency threshold to Sequence Time Delay Embedding (STIDE) with Frequency Threshold (T-STIDE). If a match occurs, it further checks for the frequency of occurrences of the sequence in the database. If the frequency of occurrence is lower than a predefined threshold, the instance is treated as a rare sequence. All the mismatch sequences are checked against Locality Frame Count (LFC). LFC which tells how many of the last 20 (fixed) sequences are a mismatch. The approach will fail if new normal sequences come in the trace which are likely to be mismatched with the nearby sequences.

Kang et al. [128] combined the frequency-based approach with machine learning approaches. They have considered all sequences in the database. The approach first converts the input system call traces into numeric feature vector called as Bag of system calls (BoS) which represents the occurrences of each system call. Therefore, the ordering information between system call is lost. Here each feature is defined by $X_i = \{s_1, s_2, s_3, s_4 ... s_m\}$ where s_i is the total occurrences of a system call s_j and m is the total number of system calls in input sequence Z_i. An attacker may fool the technique by creating the similar frequency count as of normal sequences by injecting the malicious and legitimate system calls without changing the attacks pattern. Therefore, BoS based approach is not much effective in discriminating the anomalous and normal sequences.

Sharif et al. [68] propose Secure In-VM Monitoring (SIM), a general-purpose framework in which the security tool is placed in the untrusted guest VM for efficiency. Their work incorporates the hardware virtualization and memory protection feature to create Hypervisor protected address space called as SIM virtual address space. The space is utilized to execute the monitor where the execution is transferred in a controlled manner. SIM maintains the log activities in the secure space which are accessed by monitoring program in a controlled manner. A prototype of the security architecture has been implemented using KVM VMM and Intel VT hardware virtualization technology. The placement of security code at TVM requires strong security measures for security tool. The framework only focuses on architectural aspects rather than on a good detection mechanism.

Motivated from the frequency based approaches, some researchers present the use of traditional Bag of system calls (BoS) [128] in a cloud environment. For example, Alarifi and Wolthusen [67] used BoS for detecting anomalies in TVMs. The IDS instances are distributed at each TVM and perform analysis on the system calls generated at TVM. Initially data collection is carried out in which a large collection of traces is collected over a period from VMs. The approach assumes that VMs are not malicious for some duration when they are initialized and are limited. They have collected IOCTL (input/output control services and application) system calls in KVM-based virtualization using Linux strace utility. The time complexity of the approach is linear O(n) where n is total number of lines in input file. Detection rate is 100% with 11.11% false positive with sliding window of size 6. The limitations with the BoS (as discussed above) are directly associated with the implementation in cloud.

Yin et al. [129] present the usage of Hidden Markov Model (HMM) for anomaly detection model. Earlier approaches were tedious and time-consuming as application programs are constantly updated; so building a profile for all of them is very difficult. Authors mentioned that temporal signatures are stable, and hence modeling the temporal behavior will improve the effectiveness of IDS. Therefore, the authors used HMM to learn the temporal behavior of normal sequences of the database. HMM consumes more time for training. As the baseline database contains only normal sequences, there are chances of classifying the evolving normal traces as abnormal.

Motivated from the state-based approaches, some researchers apply state transition approaches for detecting attacks in cloud. Alarifi and Wolthusen [117] have used HMM to generate more accurate normal behavior of programs that is based on probability calculations of system calls. The learning of HMM is based on Baum-Welch algorithm. KVM Hypervisor is used in their implementation. HMM takes the collected VM system calls as input and generates the state transition diagram in which each transition is presented by the probability of reaching to other states, and probability of producing next input symbol (system call). Here they have used different scenarios in training. In one scenario, they considered all possible system calls. In another scenario, only IOCTL system calls are considered. They tested the accuracy by DoS attacks patterns and found that first scenario provides 100% detection rate with 5.66% false positive whereas only IOCTL considerations lead to only 83% detection rate.

There are some limitations with the HMM. It is a gradient-based method and may converge to a local optimum [130]. The training time of HMM is also very high which is not desirable in the cloud environment where retraining of a model is desired regularly as the normal user behavior keeps on changing. Hence, recalculation of the probability distribution for the new system calls behavior will consume most of the time in learning. HMM also requires prior information of the total number of states which is not very easy to determine. It may involve lots of experimental tasks which will again consume time.

2.3.2 Hypervisor based Intrusion Detection System

Hypervisor-based IDS monitors a set of TVMs from Hypervisor by leveraging the concept of introspection functions. It is deployed at the

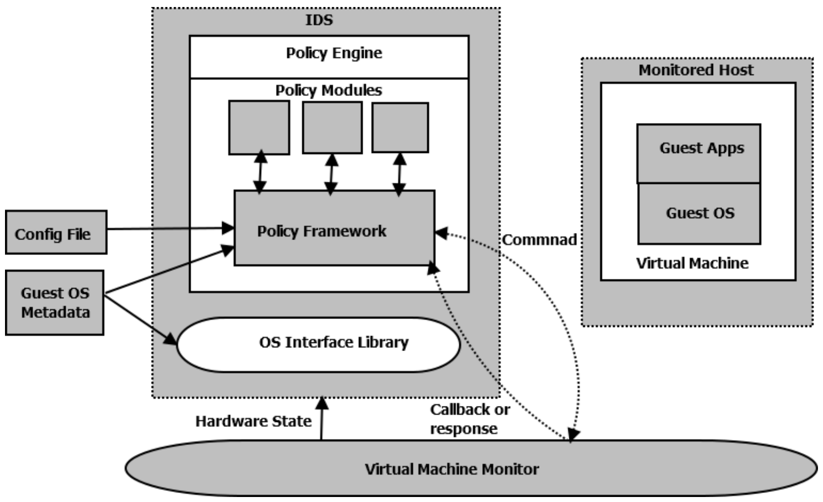


FIGURE 2.9: Virtual machine introspection based IDS architecture [123]

VMM-layer and is dependent on the Hypervisor. Researchers have proposed Hypervisor-based IDS for detecting intrusions at VMM/Hypevisor in the virtualization environment. These IDS instances run individually at each VMM and are controlled by host machine administrator. The command and control by cloud administrator is essential in cloud environment leading to work on architectural aspects of approaches for cloud. However, a few researchers have proposed Hypervisor-based IDS for cloud, in last few years, applicable to IaaS cloud environment. The details of some of the Hypervisor-based IDS are discussed.

Garfinkel [123] propose VMI-IDS architecture called Livewire, deployed at VMware [131] workstation as VMM. VMI-IDS is deployed in a VMM hosted server which makes it more secure from outside attackers. VMM provides an interface to VMI-IDS to talk to it in terms of various commands such as inspection, monitor and administrative commands and to configure intrusion detection policies. It is divided into two parts: OS Interface Library and Policy Engine. OS Library contains the detailed information of the state of VMs (application, processes, operating system). Policy engine consists of the specific policies of the different VMs. Events from the VMM interface and OS interface library are notified to the policy engine as shown in Figure 2.9. It supports both signature techniques. The policy engine needs to update time to time for new attack signatures. The technique may fail to detect unseen variants of attacks.

Kourai and Chiba [132] propose a VMI-based framework where the introspecting component is positioned inside a second VM running on the same host where first VM is running. The author named the introspected VM as VM server. Mirroring software is used to receive the frames sent to/from the introspected VM. Authors have used three techniques. In first technique, a special tap device is used inside each VM IDS; VMM forwards the traffic generated by the server VM to VM-IDS for further analysis. The second technique is Inter-VM Disk Mounting. The server VM file system is mounted to a shadow file system inside the VM-IDS in read-only mode for file integrity checking. The third technique is Inter-VM Process Mapping where a shadow process is created inside VM-IDS corresponding to each VM server process. The approaches lack in providing the mechanism for detailed behavioral analysis of programs running on monitored VM.

Payne et al. [78] propose XenAccess, which is a VMI library for Xen that provides virtual memory introspection and virtual disk monitoring from the Hypervisor. It provides easy access to DomU memory from Dom0 memory by creating a semantic-aware abstraction of DomU memory. A series of introspection functions are supported by XenAccess to provide monitoring of DomU memory. To begin with, it first calls xa_init() to initialize the xa object that holds the information used in introspection process. After this, three access functions are used. xa_access_virtual_address() is used which takes kernel virtual address as input and returns the memory page holding that address that is done via PT lookup. xa_access_kernel_symbol() function is used to convert kernel symbol to the virtual address. The conversion is performed using system.map file associated kernel from DomU. This file contains a table of symbols and addresses. xa_access_user_virtual_address() provides access to user space memory. The extended framework of XenAccess is called LibVMI [122] which provides enhanced introspection functions which supports multiple Hypervisor (such as KVM Hypervisor [133]) and multiple guest OS. LibVMI provides the interface commands which are easy to be integrated with security applications.

VMWatcher [70] provides the prototype implementation of the guest view casting technique, used for extracting the VM memory information from Hypervisor. The guest view casting for a guest OS depends on knowledge of the related guest OS drivers. The commercial systems such as Windows do not offer any such information. For Linux, `System.map` file is used by authors which is offered by Linux distributions. For Windows, VMWatcher performs the full scan of memory and identify the addresses by looking for certain signatures that are unique to kernel level data structure which are of interest. For Windows, it is being tested for Windows XP version. It uses specific signature(i.e. 0x03001b0000000000), to identify potential process instances in the Windows XP raw memory. The method of extracting the details of Windows kernel data structure details is very specific to kernel version and requires the knowledge of signatures. Secondly, it only checks the presence of processes in VM memory and limited in doing the behavior analysis of processes.

Dinaburg et al. [75] developed an Ether, a dynamic malware analysis tool, which uses the hardware virtualization extension such as Intel-VT support to provide transparent monitoring of malware samples from outside the VM. Ether traps the execution of modified pages by trapping each memory write attempt through write-protected shadow memory pages and dumps them to detect and extract dynamically generated codes. Ether traces the system calls invoked by program for analysis. Ether supports both software interrupt and fast system call mechanism of modern processors to trace the execution of analysis target. In fast system call mechanism, Ether monitors the SYSENTER and uses SYSENTER_EIP_MSR register to cause page faults and trap the execution of programs. In software interrupt, 02E interrupt is used for syscall execution. Ether changes the IDT entry for this interrupt to point to the non present page. A VM_EXIT because of page fault will indicate Ether about system call execution. Ether is restricted to Windows XP Guest only and depends on the very older version of Hypervisor i.e. Xen 3.1.0.

Payne et al. [134] propose a VMI framework named as Lares in which the security application is split into two VMs. Guest VM is the monitored machine where the user applications and services run. One part of security application consists of hooks and a specially crafted trampoline code that are placed in the guest VM. These hooks can be redirections or jumps placed inside program code or another mechanism to pass the control of program execution. Hooks are used to intercept the event at guest VM and trampoline code (an indirect jump vector) is used to pass the event signals by hooks to the Hypervisor. Hypervisor passes the events to security VM. The Hypervisor is provided with inter-VM communication functionality for passing events. The security VM consists of the core part of a security tool that does the analysis and makes decisions. The limitation with the approach is that if a guest OS kernel is compromized, the security hooks will be also under control of attackers and may send wrong notifications.

Benninger et al. [91] propose Maitland, a light weight approach to VM introspection technique, which exploits paravirtualization in cloud. Maitland follows a distributed architecture which comprises a set of loadable kernel modules one for each guest VM and other for a privileged VM. Privilege domain also contains cyber security analysis tool. Maitland uses two approaches (i) It keeps track of dirty bit flag status of memory page associated with process page table entry. Memory updates are notified by MMU update interrupt signal to OS. Hypervisor intercepts them and sends for analysis before passing to OS (ii) When a page fault interrupt comes because of NX (non executable bit set to 1) of an page; Hypervisor will check whether CR2 register contains a stack pointer of an untrusted process. Hypervisor sends snapshot of memory of suspicious processes which made an unsuccessful attempt of execution to security analysis tool. Maitland does not provide any detection mechanism and relies on the existing pattern matching tools for doing the analysis. Secondly, the security modules running in VM may get exposed to attackers and hence require strong security measures.

Xenini [69] in an anomaly based IDS framework for virtualization environment. System call tracing is carried out in paravirtualized machines with Xen Hypervisor. In Xenini, syscall interrupts (080) are trapped, and the control is passed to XenIDS before passing control to the guest kernel which is running in Dom0. System call number and process id (PID) are intercepted from %eax register. Xenini is a patch to Hypervisor responsible for the stealthy gathering of the program behavior running in VMs. Xenini communicates to XenIDS running in Dom0 via event channels using libxc interface. Event channel notifies XenIDS for the new interception data present in Xenini's buffer. On receiving an alert; libxc, an API interface is used to read data from Xenini's buffer and transfer to XenIDS. After processing the data, control transfers to the guest VM. For critical operations, Xenini waits for the answer from XenIDS before resuming execution of guest application. The detection mechanism of XenIDS is based on STIDE [115] technique and has been validated over UNM dataset. It introduces an overhead of 56%. The drawbacks associated with STIDE are inherent with XenIDS.

Nitro [135] is a VMI-based system call tracing framework which uses the hardware-based technique for collecting traces of system calls from Hypervisor in virtualization environment. It supports all three types of the mechanism provided by Intel x86 architecture: interrupt based system calls, SYSCALL/SYSRET based system calls, and SYSENTER based system calls. In interrupt-based system call implementation, user interrupts are forced to cause system interrupts and are trapped by Intel VT-X extensions. If the general protection fault is natural, guest OS resumes. However, if it is because of user interrupt (for natural interrupts, interrupt number below 32), Nitro data collection engine collects the system call information. In syscall based system call implementation, Nitro enforces interrupts on execution of SYSCALL or SYSRET instructions and collects the system calls. In SYSENTER based system all implementation, whenever SYSENTER is executed, the program switches to kernel mode at address stored in SYSENTER_CS_MSR. Nitro causes page fault by storing the original value and loading the CS register with NULL value. An attempt to load NULL value causes a general protection exception which can be trapped at Hypervisor, and necessary information can be collected from the Nitro data collection. It supports Linux and Windows guest. Ether introduces a performance degradation between 554% greater than that of Nitro. Nitro is a system call tracing technique can be integrated with security tools implemented on KVM Hypervisor.

Arshad et al. [119] propose an intrusion severity analysis approach for cloud environment, which is implemented at Dom0 of the Hypervisor. They have considered seven security requirements in their model: guest OS integrity, work state integrity, DoS, zombie protection, malicious resource exhaustion, platform attacks and backdoor protection. All the system calls are mapped according to the security requirements using REMUS ([136]) classification. Intrusion detection module uses misuse detection approach based on known attack patterns to detect suspicious system calls running in the VM. If a system call is found suspicious, it is transferred to severity analysis module (SAM) which uses anomaly detection technique (using Decision Tree C4.5) to identify severity of system call. SAM consults with a profile engine to obtain VM specific information such as security characteristics of VM and evaluates the severity of system call. The technique provides an average detection rate of 90.7954% for self generated dataset. The abstract description of technique lacks in providing the technical details of how the system calls are extracted at the Hypervisor and what features are processed for machine learning. The detection rate can be improved by applying the ensemble classifier approaches.

Wu et al. [93] propose ShadowContext, an intrusion prevention strategy for virtualization environment. It is based on system call redirection in which certain selected system calls of monitored programs are executed in Dom0 of the Hypervisor. It protects system calls from in-guest malware attacks. Some security modules of ShadowContext also run at the VMM. ShadowContext does not generate any alarm signal, as it is an intrusion prevention strategy. ShadowContext fetches the details of selected system calls of monitored programs and executes them at Dom0. It maintains their ordering in the execution sequence of a program. However, a piece of code is injected in a dummy monitored process (used for communication with Dom0) which runs at the monitored machine. The security of this code is important and depends on hardware-assisted features such as EPT protection. Moreover, ShadowContext can behave abnormally if the security manager fails to design the system call redirection properly. A redirected system call can produce unexpected results and can even crash the guest OS. It is implemented on KVM (version is not specified).

2.3.3 Network-based Intrusion Detection System

Network-based IDS examines the packets passing through the physical/virtual network interfaces. They are designed to target the network attacks in cloud such as denial of service, scanning, spoofing etc. Since, NIDSes are independent of operating system; which makes them extremely portable to be deployed at any layer (TVM/VMM/Network). However, the network points, connecting multiple servers or TVMs are most suitable location to detect cloud network (physical/virtual) attacks. Below, the details of some of the Network-based IDS are discussed:

Signature-based techniques have been used for detecting intrusions in cloud. Roschke et al. [82] propose VM integrated IDS which consists of various IDS sensor VMs and IDS Management unit as shown in Figure 2.10. IDS Management unit consists of Event Gatherer, Event Database, Analysis Component and IDS Remote Controller. This unit

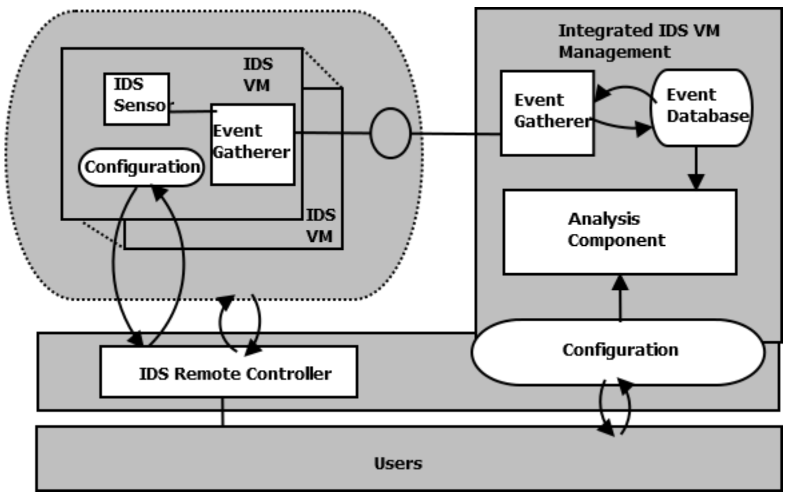


FIGURE 2.10: VM-Integrated IDS [82]

is remotely connected to various sensors. The central IDS management will gather alerts from various virtual components of the user machine and converts them into common format named as Intrusion Detection Message Exchange Format (IDMEF) and stores them in the event database. It further analyzes and correlates various alerts for detecting the presence of any malicious behavior. Users can configure the IDS by the Remote Controller which is a part of IDS VM Management unit. They take actions such as dropping or ignoring packets. The technique may fail if an attacker makes a small variation in the attack pattern without changing the semantics. The detection system also requires regular update of the signature database.

Motivated by the use of machine learning algorithms in misuse detection, some authors have applied these methods in cloud environment as well for learning the behavior of cloud network traffic. Li et al. [86] propose a distributed IDS system based on Back Propagation (BP) ANN based technique in a cloud environment. IDS sensors are deployed at each TVM, are trained using ANN algorithm over a large dataset of VM traffic (normal and anomalous) to learn about the VM profile. The trained ANN does the analysis over network traffic for anomaly detection. It raises alarms for malicious activities. ANN provides good accuracy of 99.7% in 6 min 35 sec. of training time with 7 nodes model using KDD99 dataset. IDS sensors can be compromized easily from TVM. However, KDD99 is very older dataset and does not represent the currently evolving attacks.

Mazzariello et al. [137] propose an centralized architecture for attack detection in cloud environment. An NIDS tool (SNORT) is deployed near to the cluster control for detecting the flooding attacks in cloud. The experiment is done on Eucalyptus Cloud. The centralized deployment are prone to single point of failure and are not much efficient. Moreover, having all analysis at a single node leads to the server overloading, leading to degradation in system performance. Authors have also suggested to use distributed deployment strategy in which individual NIDS components are placed at each physical and virtual machine.

Kumar et al. [111] has propose an intrusion detection approach in a cloud environment based on state-based analysis that makes use of Hidden Markov technique to model the transitions of user behavior over a long span of time. IDS instances are distributed at each TVM and detect the intrusions based on the behavior probability of user actions. HMM develops the data seeking profile of users that acts as a baseline profile and is factored into three profiles: low, middle and high. The high profile is the one that matches with the baseline model of the user. Middle profile refers to the current profile of user that partially matches with baseline profile of the user. If patterns have very low probability of matching, they will correspond to the low profile. All matching is done based on a certain threshold value set for each profile. In this way, even if a user hacks username and password, his/her profile will have low value. An IDS will be configured based on the profile matching the behavior of currently observed patterns. There are some limitations with the HMM. It is a gradient-based method and may converge to a local optimum [130].

Lee et al. [138] propose a method that binds the users to different security groups based on the anomaly score and accordingly applies the security policies to differentiate them. When a user requests access to the cloud system, AAA (Authentication, Authorization and Accounting) module is used to calculate the anomaly score of user and based on the score, a suitable IDS is chosen. AAA module requests the host to assign guest OS image with the selected IDS for the user. Users information, transactions and system logs are stored in the database. The database periodically communicates with the AAA and host OS. Private data of the users is stored in storage centers. Security levels are categories into three types: high, medium and low. High level applies the strong security rules with signature based to Intrusion IDS.

Medium level applies pattern of all known attacks to IDS. Low level applies known pattern of chosen malicious attacks to IDS. Thus resource consumption varies from strong to low level. A suitable IDS is chosen on the basis of the security requirements of the user which enables the efficient usage of cloud resources. It will save computational cost of the system since for a system with low security need, a full functional IDS need not to be deployed. Risk points are given to different users based on the anomaly score. A suitable IDS is chosen based on the risk points. This will increase the speed of detection. The IDS is completely under control of the virtual machine users. A nefarious user can manipulate the security policies, leading to a threat to the cloud infrastructure. Secondly, maintaining signature based IDS with different levels of security requirement leads to extra overhead incurred in updating each of the IDS instance.

Gul et al. [71] propose the centralized detection model for distributed cloud architecture. Each IDS instance is deployed outside of the VM servers at the network points such as routers, switches, gateway etc. They have addressed the issue of huge data processing by cloud servers. Authors also addressed the transparency of Cloud IDS to users. A third party monitoring service is deployed outside of the Cloud Service Provider (CSP) infrastructure which receives the alerts from the IDS and informs users about the attacks on their machine. The third party service also provides expert advice to the Cloud service provider about the mis-configurations and loop holes in the IDS. It needs a trusted management system on the third party authority for control and configuration of IDS. The spoofed traffic with false virtual IP and MAC can not be recognized from the network points, leading to high risk of DoS attacks.

Srinivasan et al. [139] propose eCloudIDS to identify various anomalies in the cloud VMs. The two key components of the system are uX-Engine subsystem (Tier 1) and sX-Engine subsystem (Tier 2). A Cloud Instance Monitor (CIM) subsystem observes the actions performed by user on a user specified application directory and informs H-log-H component for immediate logging. An Audit Log Preprocessor (ALP) extracts the information from H-log-H and pre-processes it to make it compatible with uX-Engine subsystem which is a learning subsystem and uses unsupervised classification algorithm particularly self organizing map (SOM) (Kohonen, 1990) as a first tier detection to detect the normal behavior, abnormality or special permission. It communicates with Permission Recorder to verify the abnormality. The technical description of second tier sX-Engine is based on any existing supervised classification technique. The unsupervised learning approach applied by eCloudIDS provides low accuracy of 89% detection rate with 2% false negative rate and 9% false positive rate. The technique will fail for evolving normal behavior and also against spoofing attacks.

Modi et al. [90] improve the detection reliability of the signature based NIDS by adding one more analysis step (anomaly detection module) after signature based module as shown in Figure 2.11. The NIDS instances are deployed at all cloud regions, i.e. all TVMs and cloud servers. Snort is used as a signature based IDS, which uses a database of known attack signatures. Decision tree algorithm is used to classify intrusions based on the anomaly detection approach. Their intrusion detection framework pre-processes the packets and passes them to Snort. Snort uses the knowledge base for matching them against packet patterns. If Snort signals legitimate behavior, the packets are passed to anomaly detection module. In other case, an alarm is raised. The anomaly detection module checks the patterns against the decision tree rules and decides whether a packet belongs to intrusion class or legitimate class. For intrusive behavior, it raises the alarm. All alerts are collected at a central log. The other databases are updated if central log is updated. The authhor used a well known dataset (KDD99) which exhibits different behavior other than advanced network traffic and hence not satisfactory. It provides an accuracy of 84.31% with false positive rate (FPR) of 4.81% for NSL-KDD99 dataset and 96.71% accuracy with FPR of 1.91% for KDD99 dataset. The anomaly detection module is an additional step after Snort. As all the benign traffic is forwarded and the malicious traffic is marked bad, this cannot reduce the false positive rate of Snort (only the false negative).

Modi et al. [140] in their further work improve signature-based NIDS by integrating the apriori algorithm with Snort signature database to increase the accuracy and effectiveness of the NIDS. Their motivation was to detect attacks and derivatives of attack. Aprioi module takes the partially known rules and support threshold from the signature database as an input. It then generates the new possible rules and also updates the signature database. The derivatives of the attack can also be detected by Snort. The implementation is done using a very

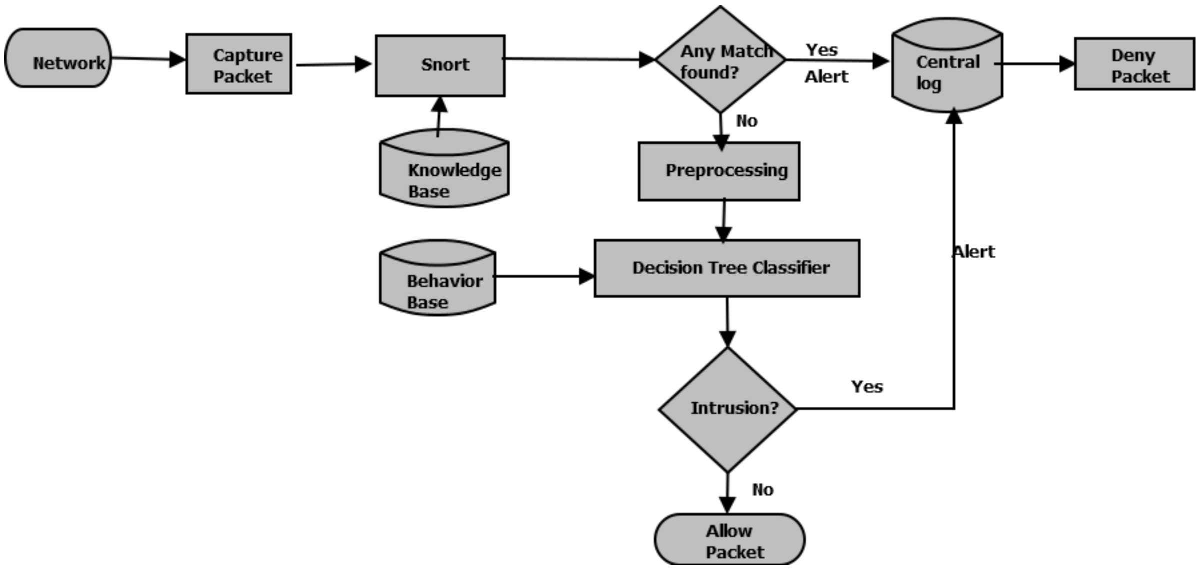


FIGURE 2.11: Execution flow of Cloud-based NIDS (signature and machine learning)[90]

old dataset (KDD'99) which exhibits different behavior other than real traffic and hence is not satisfactory.

Lin et al. [141] propose a NIDS which is deployed at the Dom0 of VMM. The detection rules in NIDS are configured according to the OSes and services executing in the VMs running above VMM. VM information is obtained from the operating system kernel map in the Hypervisor. Services are identified, and rules are updated dynamically. Contents of the memory region of the VM are used to refer to the VM information. In Win32, TIB (Thread Information Block) and PEB (Process Environment Block) can be used to refer to the OS version. In Windows, kernel establishes EPROCESS data structure and MTOM data structure to refer to processes and information related to those processes. The model is implemented in the virtual platform of Linux 3.2.1, WinXP and Win7. Snort is used for network intrusion detection. The system will add significant overhead at the centralized IDS. It is because of the time complexity that is associated with the rule-matching mechanism that will lead to high computational cost at the time of detection. Next, the lack of network introspection feature at VMM leads to other advanced network attacks, originated from TVM.

Tupakula et al. [142] propose an IDS architecture named as VICTOR in IaaS cloud environment as shown in Figure 2.12. VICTOR uses Intrusion Detection Engine (IDE) together with some other components such as packet differentiator, Operating System Library and Repository

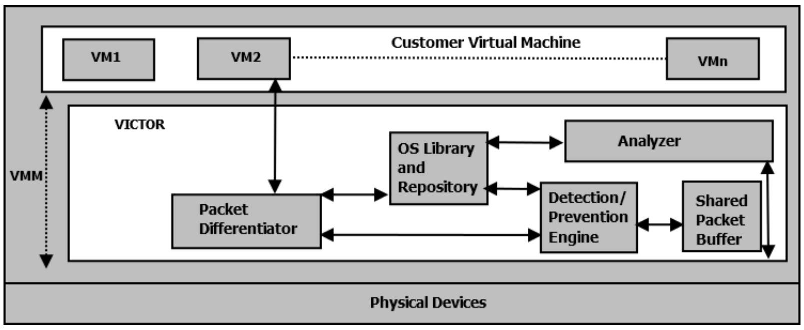


FIGURE 2.12: VICTOR: Basic cloud security architecture [142]

(OSLR), Analyzer and Shared Packet Buffer to make the IDE compatible in a virtual environment. The model is integrated into VMM or host OS. Packet Differentiator receives packets from VMs. Details of the entity (process, application, virtual machine, OS) are updated in the OSLR. It also checks for the correct source address and forwards to IDE. OSLR library contains the detailed information of configurations, details of resources, OS and applications of each VM. OSLR also verifies information reported by packet differentiator. If hidden processes have generated some packets, OSLR raises an alert to IDE for further analysis. IDE matches the packet for known patterns (signature matching) and then for the legitimate pattern by anomaly module. Anomaly module applies machine learning techniques over OSLR to update the behavior of VM. Suspicious packets pass to analyzer and shared packet buffer. Legitimate packets are sent to destination. Detection of malicious entity is done at a fine granular level to avoid denial of service attack. The analyzer determines the legitimacy of the packet with the help of OSLR. Malicious entities are isolated, and new attack signatures are identified, and the attack signature database is updated. Shared packet buffer stores detailed information of suspicious VM. The limitations with the approach is that it is not integrated with advanced memory introspection and network introspection functions, needed to extract detailed VM state information.

Watson et al. [143] propose a Cloud Resilience architecture which is composed of individual instances of a security component known as Cloud Resilience Manager (CRM). A CRM performs anomaly detection at each local node using Support Vector Machine (SVM). One component of CRM is responsible for recovery process and other for coordination between other instances. Authors have tested the architecture with KVM based deployment for detecting attacks Kelihos, Zeus and

DoS attacks and achieved over 90% accuracy. The technique provides low accuracy of attack detection and has been suitable for KVM-based cloud deployments.

Pandeeswari and Kumar [126] present the use of machine learning for intrusion detection at the Hypervisor. Their technique is based on virtual network traffic analysis that is collected at the Hypervisor in normal and anomalous scenarios. The detection mechanism is based on the integration of Fuzzy C-mean Clustering with ANN to learn the input training instances. The collected instances are used as a training database for the classifier. Fuzzy clustering algorithm creates cluster subsets based on membership value. ANN algorithm then trains each sub-cluster which are combined by aggregation module. The technique is found to perform well for capturing the anomalous traffic generated by VM and provides an average detection rate of 97.55% with 3.77% average false alarm rate for detecting intrusions. However, it involves extensive training due to the complexity of the algorithms. The validation is done using KDD'99, a very older dataset. Hence, it is difficult to assess the performance of system for current real time attacks.

2.3.4 Distributed Intrusion Detection System

A Distributed IDS consists of various IDS instances (TVM based IDS, Hypervisor-based IDS and/or Network-based IDS) over the large cloud network, deployed at different locations in cloud. These IDS instances communicate with the central cloud administrator or each other to detect attacks in cloud. The distributed approach is helpful to detect and analyze the attacks collectively.

Che et al. [144] propose a distributed security architecture which uses state based approach for detecting intrusion in cloud. The detection component is distributed among the monitored machines and analyzes the multiple log files generated as per user actions. A sequence of user actions are recorded and stored in log files. The logs are analyzed to find the main motive behind the user actions. The main idea behind the approach is that that attacks such as buffer overflow, scanning and password guessing etc. leave certain traces in the logs. The audit logs can be monitored to identify the attack plan. Event-correlation is then applied to find the correlation between user events. The analysis leads to detection of certain attack patterns. Based on the report, a suitable response is generated to handle them. The accuracy of the approach is highly dependent on the correlation methods which may generate more false alarms if logs are subverted. Secondly, the detection of certain patterns is based on the attack signature matching. This leads to the need of continuous updation of the attack signatures for detecting evolving attacks or variations in attacks.

Motivated from enumeration based approaches, some researchers applied similar approach in the cloud. For example, Gupta and Kumar [84] propose intrusion detection technique based on system call analysis named as Immediate System Call Sequence (ISCS) for detecting attacks in a cloud environment. ISCS instances are distributed to each TVM and centrally controlled by administrator. The approach does not use any learning based system instead it creates a database of system calls structured in a key-value pair format. A key represents a unique system call name whereas value represents an immediate sequence of system calls following it during program execution. The programs to be monitored are listed in the configuration file as specified by cloud admin. Cloud admin performs the program-wide detection by matching the baseline ISCS snapshot with individual ISCS snapshot. Any mismatch from baseline database corresponds to anomalous sequence. ISCS achieves 98% accuracy for intrusion detection. The approach lacks in using the statistical techniques to learn the normal behavior of the system. If a normal sequence does not appear in the database, it will be flagged as anomalous. The chances of such situations occurring are high in huge networks where user' behavior keeps on changing. In this case, this would be inappropriate to say that the instance is anomalous if it does not conform to any match and will result in false positives.

Gupta and Kumar [73] propose an approach named as Malicious System Call execution Detection (MSCD) based on program cum system-wide detection. The approach creates a program-wide MSCD database at each VM and system-wide MSCD database at cloud manager. Program-wide detection matches the current individual MSCD snapshot of each client VM with baseline MSCD snapshot at client VM. System-wide detection matches the current ISCS snapshots of client VMs with MSCD snapshots at cloud admin. The approach provides reliability even if IDS daemon at client VM is subverted, and intrusions can be detected at the system level. MSCD algorithm is tested for validation over UNM (University of New Mexico) sendmail dataset and achieves an accuracy of 80% with 3% false negative rate. The approach may face technical challenges in keeping replicas of VM snapshots and prone to IDS subversion by modern malwares.

Varadharajan et al. [145] propose an integrated security architecture which integrates the intrusion detection techniques with access control policies and trusted attestation techniques. The security components are deployed at Dom0 of VMM and communicate with each other to detect the intrusions. Security tool runs at the virtualization layer of VM hosted server and is completely controlled by cloud administrator. Physical server contains the trusted platform module (TPM) whereas each Virtual Machine Monitor (VMM) is equipped with Intrusion Detection Engine (IDE), Access Control Module (ACM) and Decision Evaluation Engine (DEE). IDE is responsible for detecting intrusions. ACM contains the access control policies for each VM and DEE makes security decisions. The IDE is one of core part of the cloud security framework which applies the signature matching and anomaly detection techniques to detect intrusions. The analysis done by detection engine can be extended for doing the detailed investigation on suspicious processes using system call analysis.

In the above distributed architectures, IDS instances run over individual tenant VMs (TVMs) or each VMM but are configured and controlled by a Cloud Controller Server (CCS). The IDS instances use the resources of VM or VMM (where they are deployed) for their operation and have no control mechanism for tenants. The cloud administrator is responsible for creating policies, controlling the IDSes and responding to alerts. He/she also manages how an IDS instance will behave for a specific tenant VM. It has good visibility of the host machine as it is deployed at each tenant VM. The Cloud administrator (CA) can specify how a tenant VM IDS instance should behave, which makes it achieve better performance than the centralized IDS. CCS regularly gets alerts from individual tenant VMs about the attacks that have happened and acts as a central point for log management.

Some of the distributed architecture also provide a collaborative mechanism in which the IDS instances communicate with each other and update about attack information. The information is correlated collectively to detect distributed attacks.

Lo et al. [74] propose a cooperative intrusion detection framework based on distributed architecture. An IDS is deployed in each Cloud

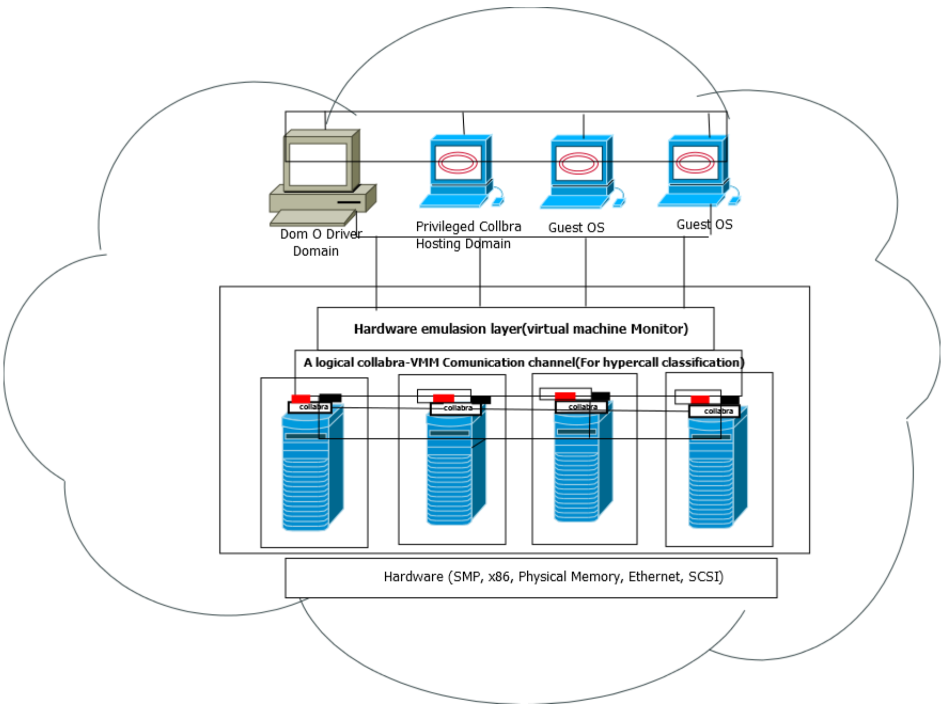

Dom O Driver Domain
Privileged Collbra Hosting Domain
Guest OS
Guest OS
Hardware emulasion layer(virtual machine Monitor)
A logical collabra-VMM Comunication channel(For hypercall classification)
collabra
collabra
collabra
collabra
Hardware (SMP, x86, Physical Memory, Ethernet, SCSI)

FIGURE 2.13: Security architecture of Collabora [72]

Computing region. Snort based IDS is implemented integrated with other components such as alert clustering, threshold computation and comparison, intrusion response, blocking and cooperative operation. A block table is maintained along with signature database containing information about packets to be blocked. Snort first checks the packet against block table and then against known signatures. If packet is abnormal than it is forwarded to alert clustering which determines the severity of packet based on threshold value. If the severity score is high, the packets are dropped and others are accepted. Intrusion response component blocks the malicious packet and an alert is sent to other IDSes. It decides the majority vote based on a threshold. If majority vote is high, it adds a new blocking rule in the block table and otherwise discards the alerts as false. The requirement for additional block table is not clear. The approach also faces the challenges of IDS subversion and is prone to signature-manipulation attacks.

Bharadwaja et al. [72] propose a distributed security architecture, called Collabra as shown in Figure 2.13. It is a Hypervisor-based IDS to detect anomalies in virtualized environment. Collabra is integrated with each VMM and acts as a filtering layer between VMM and Dom0 in the guest VM network. It checks the integrity of the hypercalls. It provides a collaborative detection mechanism to prevent attacks at

VMM because of multiple hypercall requests from multiple VMs in the network. Admin version of Collabra runs alongside Dom0. Each Collabra instance of VMM communicates with another Collabra instance through a logical control channel (LCC). Whenever a guest VM issues a security critical event, it is screened by Collabra admin for any authentication and integrity checks. There are two key security components: hypercall integrity check and hypercall origin check. Each hypercall is cross-checked against the Message Authentication Code (MAC) and specified policy for the call against the cryptographic repository. If the call is not known, it is cross checked with other Collabra instances running in the virtualization network for classification. The classification is based on their origin check. This module checks the origin based on the privilege of VM location and the application that causes the hypercall. A legitimate call passes the module whereas calls made from unknown sites are marked as untrusted and are communicated with other instances. The model theoretical and implementation is not discussed. The technical details such as how the MAC is generated, how the known call sites are maintained and how an anomalous score is calculated for each hypercall is not explained. Moreover, maintaining a logical channel adds overhead to the Hypervisor and increases the attacking surface.

Kholidy et al. [146] propose hierarchical and autonomous cloud based intrusion detection system to secure VMs and back-end servers. IDS is placed in the VM management server, and it sends the alarm with associated risk impact factor to the Controller. The key components of the framework are Event Collector, Event Correlator, Event Analyzer (NIDS analyzer, HIDS analyzer, DDSGA analyzer) and Controller. Event Collector collects host based and network based events from various sensors and correlates host and network events to observe the user behavior in several VMs. Event Analyzers detect the host and network events, based on the probability of user actions leading to attacks and communicates to the Controller for appropriate actions. The autonomous Controller provides the most appropriate response to protect the cloud environment. Traditional HIDS are limited in their capability to detect advanced attacks such as evasion based attacks at virtualization-layer in cloud environment.

Haddad et al. [147] propose a collaborative intrusion detection framework (C-NIDS) where IDS sensors are deployed at each cloud compute server at each VMM in cloud to detect the network attacks. The architecture uses SNORT as a prime security tool to detect attacks in cloud. Once the attack signature is found in the packet flow; an alarm is sent to cloud admin. The normal traffic again passes through the SVM classifier for detection of any abnormality in the traffic. This additional analysis does not reduce the false alarms generated by the SNORT. The collaborate about each other to update about the attacks. However, no verification of updates is provided, leading to more false alarms.

2.4 Research Challenges

There are various security concerns in cloud as discussed in the previous sections. However, our work in this thesis is focused on intrusion detection in cloud environment. Based on the exhaustive literature survey on intrusion detection in cloud, we have identified some research challenges associated with intrusion detection in cloud.

• **Detection of malware attacks:** Signature-matching and static analysis techniques can be evaded by obfuscation and encryption techniques. Dynamic analysis overcomes this limitation. The existing system call analysis approaches for cloud such as Bag of System calls (BoS) [67][148] and Immediate System Call Sequence Detection (ISCS) [73][84] have some limitations associated with them. In BoS, the ordering of system calls is lost which is very important for attack pattern identification. Though, ISCS retains the ordering, it will fail for longer or infinite length traces. Moreover, these approaches assume to have a full knowledge of the normal execution behavior of programs. If a sequence does not appear in the baseline database, it will be flagged as anomalous. The chances of such a situation happening are high in large network systems, where users' behavior keeps on changing with time. In this case, it would be inappropriate to classify such instances as anomalous, if they do not conform to a match, which can result in high false positives. Secondly, they do not apply the statistical-learning-based techniques which is very important for generalizing the program behavior.

• **Subversion of security monitor:** An attacker can check the presence of security processes running at TVM and can try to disable it.

For ex., malware such as Torpig and Conficker disable the security tools and other security-critical services such as auto-update, error reporting and Windows Defender services [101]. Some of the malware hide their presence from the security monitor running at TVM. In addition, advanced malware can refrain from executing if they detect the presence of security components at the TVM. Time-based, processor feature based and exception-based evasions are some examples of evasive activities as discussed in Section 2.1.2. The existing IDS approaches [84] [91] [117] [148] are limited in their capability to detect such kind of attacks in cloud environment. A CSP must provide an efficient security service to detect stealth malware at VMM-layer.

• **Detection of Network attacks within virtulization-layer:** In cloud environment, there can be multiple VMs connected over a virtual switch and create a virtual network. The traditional NIDS and existing frameworks, that deploy IDS at central controller/cluster node/ cloud physical server [71] [87] [143] [147] are limited in their capability to detect network attacks at virtualization-layer in cloud environment. These security solutions fail to detect VM attacks targeted from one TVM to another TVM on the same physical server, as the traffic never passes through physical network. The situation becomes more complex if the attack packets are forged by virtual IP/MAC address. Most of the cloud security frameworks apply traditional signature-based NIDS at TVM-layer to detect network intrusions [82] [138] [83] [139]. Signature matching techniques require regular updates. A small variation in attack pattern can evade the security tool. In addition, applying the signature matching at VMM-layer [141] [149] may impose a significant overhead to the system. It is because of the time complexity that is associated with the rule-matching mechanism that will lead to high computational cost at the time of detection. Some recent works [126] apply machine learning for detecting network intrusion at VMM-layer. However, they are not integrated with the network introspection functions. The evaluation is based on a very older dataset KDD99 [150], estimation of the real performance of the system is difficult to assess.

• **Virtual Machine Introspection (VMI) enabled security:** We identified that not much work has been done in the direction to provide VMI-based security solutions for cloud. The existing security solutions have some limitations associated with them which make them less suitable in a multi-tenant cloud environment For example, VMST [92] requires that the OS of the security VM (monitoring VM) must match with the OS of the TVM being introspected, which is not practically feasible in a cloud. ShadowContext [93] is an attack prevention strategy based on system call redirection approach. An improperly configured system call redirection module can crash the operating system kernel. Maitland [91] provides introspection in cloud but it itself does not provide any detection approach and relies on existing signature matching approach. There exist some VMI libraries such as LibVMI [122] and XenAccess [78] which provide the VM state information but they do not provide any detection mechanisms. Some of the introspection approaches [70] provide the limited introspection functions and are not sufficient for intrusion detection applications. Xenini-IDS [69] is proposed for virtualization environment which does the behavior analysis using very older STIDE approach [115]. STIDE makes use of string features and is prone to string manipulation attacks. Moreover, its detection mechanism only rely on all normal sequences which will produce more false alarms in dynamic cloud environment. Therefore, there is a need to provide a more efficient, VMI enabled security framework which is compatible to work for cloud deployment and is integrated with efficient detection mechanism.

• **Feasibility of same security solution at all layers:** Distributed frameworks overcome some drawbacks of existing IDS frameworks [128] [117] [144] [91] [93] [86] that were not centrally controlled and hence prone to attacks from nefarious users. However, it is being identified that distributed TVM-layer approaches cannot be applied directly at VMM-layer. This is because Hypervisor/VMM can view the guest operating system's information as raw bits and bytes. At the VMM-layer, high-level semantics of TVM is not known. For example, the information about a TVM such as processes, data structures, files and OS abstraction etc. are not recognized from Hypervisor. In the same way, distributed Network-layer approaches can not view the guest-specific information from outside. Hence, the practicability of the security frameworks [74] [90] [87] [84] [147] [151], which claim to deploy the same security solution at all regions in cloud sounds less feasible and inefficient.

- **Efficient security architecture:** Majority of IDS proposals for cloud [86] [141] [83] [146] [126] perform network traffic analysis to detect various types of attacks using network traces. The network traffic analysis can be helpful to detect network attacks. However, it is less accurate in detecting low-frequency attacks such as rootkits, hidden malicious processes, malware etc. just based on the examination of their network statistics. Some of the researchers [84] [148] worked in the direction of process analysis at the TVM-layer of cloud. However, again only process analysis is not sufficient for detecting network attacks. The IDS which does network traffic analysis or process analysis is not robust and efficient. Hence, a security architecture which incorporates both traffic monitoring and process monitoring is required.
- **Robust layered-security architecture:** Distributed IDS which are centrally controlled by cloud administrator or which cooperate with each other for attack detection are robust than other types of IDS. However, the existing distributed security architectures support distributed guest-based IDS [84] or distributed VMM-based IDS [72] or distributed network-based IDS [74][87]. These IDSes detect the attacks at specific-layer. There is a need to provide a more robust security architecture which covers all three-layers of cloud i.e. TVM-layer, VMM-layer and Network-layer, addressing the design limitations at individual layer to facilitate the detection of both malware and network detection in cloud.
- **Policy management for IDS:** Existing techniques lack maintenance of reliable policies which in turn lead to deterioration of the efficiency of security tools. Cloud administrator needs to set up various policies for an IDS where ever it is installed and also update it from time to time. Moreover, the routing path for a VM communicating with non-co-resident VMs needs to be established by the cloud admin based on the location of nearby IDS (if IDS is installed at cloud network points) whenever a new flow request comes from a VM. The manual update is time-consuming and can be prone to errors. The automation of such policies should be systematic.
- **Limitations with conventional IDS:** Some solutions [90] for cloud-based IDS integrate the conventional signature matching (Snort) with anomaly detection (machine learning). The anomaly detection module is an additional step after Snort. As all the benign traffic is forwarded and the malicious traffic is marked bad, this cannot reduce the false

positive rate of Snort (only the false negative). As, there is no verification of an alert, it does not reduce the false alarms, either for Snort or for the anomaly module. Some cloud-IDS [148] apply conventional host intrusion detection approach [128] in cloud. The techniques may sound good when implemented at the TVM as the required information (such as network traffic or system calls) can be gathered at the VM without the need for additional expertise. However, conventional techniques when implemented at the VMM, require additional tools or mechanisms to obtain a high-level view of TVMs.

• **Software assisted solutions for improving Hypervisor security:** Some low-level attacks can tamper security functions of Hypervisor and can manipulate the fields of kernel-data structure of compromised Hypervisor and host OS. Researchers have proposed security architectures to improve security of the Hypervisor. However, most solutions [152] [153] are hardware-specific and require modification to the hardware design such as CPU, MMU etc. and hence are less flexible to adapt. Furthermore, there is a need to address the security of the security components themselves when implementing them in software.

We can conclude that there is need to provide a more robust and efficient security architecture to deal with major security challenges identified. To overcome the limitations with traditional approaches, CloudHedge incorporate VMI approaches in the security proposals to detect attack at the VMM-layer of cloud environment. Introspection based IDS provides a high confidence barrier between attacker and security monitor than traditional IDS. Efficient dynamic analysis approaches which are based on n-gram analysis and system call graph based analysis are integrated with VMI and Machine learning based approaches to detect different types of attacks such as hidden process detection, program subversion detection and evasion attack detection, etc. To address the limitations with NIDS in detecting attacks at virtualization-layer, CloudHedge provides Behavior based Network Intrusion Detection Approach for detecting network intrusions at both VMM-layer and Network-layer in cloud environment. The third line of defense also provides security from spoofing attacks by leveraging the concepts of network introspection and hypervisor libraries such as Xenstore.

To address the limitations with existing security architectures, CloudHedge provides three-levels of security check for intrusion detection which makes it more robust than other cloud security approaches. The first line of defense provides the detection of attacks at TVM-layer and provides efficient TVM-based monitoring solution. The second-line of defense provides the detection of attacks at VMM-layer by providing a program-semantic aware VMI based monitoring solution. The third-line of defense provides the detection of attacks at both network and VMM-layer. The loop holes in the existing cloud security approaches are identified which gave us a direction for providing a more robust and efficient security approaches. The proposed TVM-level security approach is efficient and applicable to traditional physical hosts and all cloud deployments such as SaaS, PaaS and IaaS.

2.5 Conclusion

An exhaustive literature survey is carried out which covers how the traditional security solutions are being used for cloud security. It also discusses about the virtualization-specific security approaches, proposed for cloud environment. The contributions of the chapter can be summarized as: An threat model is proposed based on target cloud component. Various attack surfaces related to the cloud environment are presented. Classification of intrusion detection mechanisms in cloud environment is proposed. A classification of Virtual Machine Introspection techniques (VMI) is proposed. The categorization of key IDS security proposals have been provided with discussion of their pros and cons. The detailed analysis of techniques provide readers a strong and coherent view on security solutions proposed so far. Research gaps have been identified in the existing cloud security approaches.

We can conclude that there is need to provide a more robust and efficient security architecture to deal with major security threats identified. We also identified the loop holes in the existing cloud security approaches which gave us a direction for providing a more robust and efficient security solution for cloud. VMI approaches are more specialized intrusion detection approaches developed to work in virtualized environment. There is need to incorporate VMI approaches in the security proposals for cloud environment. Introspection based IDS provides a high confidence barrier between attacker and security monitor than traditional IDS. To address these challenges, we propose a security framework which incorporates the introspection approaches and provides three lines of defense to detect specific set of attacks. The first-line of defense provides the detection of attacks at TVM-layer by providing an efficient guest-based monitoring solution. The second-line of defense provides the detection of attacks at VMM-layer by proving a program-semantic aware VMI based monitoring solution. The third-line of defense provides the detection of attacks at both network and VMM-layer by providing network traffic monitoring solution integrated with network introspection concepts. This makes our security proposal more efficient, robust and VMI-based when compared to solutions which apply conventional security approaches at all regions in cloud to detect any abnormal activities. The description of proposed security architecture with information about all three line of defenses is presented in the Chapter 3, in detail.

Chapter 3

CloudHedge: Intrusion Detection Framework for Cloud Environment

This chapter describes the proposed security framework, CloudHedge for Intrusion detection in cloud environment. CloudHedge provides various security solutions to detect intrusions at different security-critical positions in cloud such as tenant virtual machine (TVM), virtual machine monitor (VMM) and cloud network server. The security solutions have been divided into three types: TVM-based (deployed at TVM-Layer), Hypervisor-based (deployed at VMM-Layer) and Network-based (deployed at Network and VMM-Layer). Each of the security solution is provided by one of the CloudHedge sub IDS instances namely Malicious System Call Sequence Detection (MSCSD), VM Introspection based Malware Detection (VIMD) and Malicious Network Packet Detection (MNPD). The instances are distributed, centrally configured and controlled by cloud administrator.

3.1 Introduction

We propose a robust, efficient and VMI-based distributed security architecture, called **CloudHedge** for detecting intrusions in cloud environment. It addresses the limitations with existing cloud security proposals as discussed in Chapter 2. CloudHedge exhibits monitoring capability of both program behavior and network traffic based on introspection of VM memory and network features. CloudHedge detects intrusions in cloud environment by providing different-levels of security. The CloudHedge places the monitoring tool at various security-critical positions such as tenant virtual machine (TVM), hypervisor/virtual machine monitor (VMM) and cloud network server. It provides different intrusion detection strategies for both network and malware attack detection in cloud and claim that same security solution may not be efficient and viable to be applied at various regions in the cloud. It is because of the semantic gap problem at the virtualization environment and limitations associated with the security solutions. Semantic gap problem refers to interpreting the low-level bits and bytes of a guest OS into a high-level semantics. Furthermore, there are different design choices which security researchers have to make while designing solutions at various layers in cloud. The existing solutions at one particular layer are not complete and hence incapable to deal with advanced attacks. Some of these attacks hide their presence on detection of the security tool or even try to evade the security tool running in the same virtual memory in which the tenant applications are running. We propose a comprehensive intrusion detection framework called CloudHedge which provides efficient security solutions at TVM, VMM and Network-layer against different types of attacks in the cloud. CloudHedge exhibits the following unique qualities:

∗ It performs three-lines of defense which makes our architecture more robust to attacks which may bypass the basic security provided by a cloud administrator.

∗ The first-line of defense is provided a program behavior monitoring based TVM-layer security solution without requiring any complex functions to gain insight into TVM. The monitoring tool has got access to full contextual information of the monitored machine.

∗ The second-line of defense is provided by a introspection-assisted program behavior monitoring based VMM-layer security approaches with the support of introspection libraries. Two different detection mechanisms are provided for doing program behavior analysis at hypervisor.

∗ The third-line of defense is provided by a network monitoring approach for malicious network packet detection in cloud at both Network-layer and VMM-layer.

∗ There are various complexities and privileges associated with each layer in cloud which makes the single security solution infeasible to be applied at multiple locations. The architecture and design of CloudHedge overcomes the gap of existing security approaches (NovelNIDS [90], C-NIDS [147]) which apply single security solution at various regions/layers in cloud.

∗ It incorporates VMI approaches at hypervisor for providing the high-level view of processes running in TVM which is a way ahead of other IDS approaches (ISCS [84], HMM-IDS[117]) which do process monitoring at TVM-layer. Moreover, these existing frameworks do not support both memory and network introspection for attack detection in cloud.

∗ It supports both process monitoring and network monitoring functions which makes it superior than other approaches (Xenini-IDS [69], Signature-IDS [93], FingurePrint-IDS [154], Shadow-Context [93]) which rely only on one type of analysis for various types of attack identification.

∗ It can detect various malware attacks such as privileged program modification, hidden processes, rootkits and various other evasion based stealthy malware attacks, virtual IP spoofing and virtual MAC spoofing attacks and other networking attacks by running the individual CloudHedge instances at different positions.

∗ It utilizes machine learning approaches for learning the intrusion profiles of monitored TVMs. Machine learning techniques classify processes based on learned patterns and hence can help to reduce false positives which occur with rigid sequence matching with the normal traces.

CloudHedge is robust solution since the attacks which are bypassed by one layer can be detected at subsequent layer. Moreover, it performs the monitoring of both program behavior and network traffic with support of introspection functions. It detects the suspicious behavior of the applications running in the TVM by monitoring the virtual machine both from inside and outside. This makes it efficient when compared to other approaches which only do program analysis or network traffic analysis, either from inside TVM or outside TVM. CloudHedge is adaptable as it employs the machine learning approaches to provide a learning based detection model which can be retrained to learn the features of new attacks in future. CloudHedge is distributed in nature as individual CloudHedge sub IDS instances are executed at TVM-layer, VMM-layer and Network-layer but are centrally configured, managed and controlled by cloud administrator.

Cloud administrator monitors the individual sub IDS instances, running in each layer and responds against alerts reported by IDS instances. He/she takes decision based on the logs generated by the IDS-instances and knowledge about the individual TVM and applications installed. He/she has the highest privileges to stop/restart/resume TVM or applications or processes, reported as malicious by individual sub IDS instances of CloudHedge. The proposed framework is an effective security solution to detect malicious activities in cloud.

3.2 Security Architecture

A CSP is responsible for securing the cloud infrastructure from malicious entities which target the virtual domains running in cloud. The existing security proposals have some limitations, as discussed before. Hence, we propose a robust security architecture incorporating efficient approaches for providing the intrusion detection at three different layers in cloud. CloudHedge is intended to detect attacks at various security-critical positions in cloud environment. The three lines of defense and security architecture for deployment of CloudHedge sub IDS instances is discussed in subsequent subsections.

3.2.1 Lines of Defense

CloudHedge offers both basic and advanced security functions to detect malware and network attacks at different layers of cloud. To provide a robust security architecture, CloudHedge provides three-lines of defense for detecting intrusions in cloud environment using its three sub IDS instances. Each of the three modules are motivated to solve the research gaps identified in existing security solutions, as discussed in Chapter 2. We have categorized the security solutions of CloudHedge into three types: (i) TVM-based security solution (ii) Hypervisor-based security solution (iii) Network-based security solution. The brief description of detection capabilities of each of the security solution provided by CloudHedge is described below:

(i) TVM-based security solution: TVM-based sub IDS instance runs inside the individual TVM and examines specific guest-based actions as following: *What applications are installed and are currently running in virtual machine?, What files are being processed?, What regions of the memory are being accessed by applications in terms of their system call interaction with the guest operating system?, etc.* It performs the analysis of system calls and programs of monitored TVM with good detection efficiency. Since, it has direct access to all contextual information of monitored VM where the detection agent is running. TVM-based security tools have the greatest visibility into the monitored host. They are installed at TVM-layer and can be controlled either by the tenants or cloud administrator, depending on the tenant's security requirements as stated at the time of registration in terms of SLA. However, TVM-based security solutions completely depend on the trustworthiness of the guest operating system and may not provide a comprehensive security from advanced attacks. Once guest OS kernel is compromized, an attacker can disable the security tool running in the virtual machine. The proposed security approach for TVM-based monitoring provides first-line of defense and detects the intrusion at TVM-layer such as malicious modifications of the privileged programs, based on program behavior analysis .

(ii) Hypervisor-based security solution: Hypervisor-based based sub IDS instance is configured to monitor a specific set of guest virtual machines for malicious actions from the hypervisor/VMM. For example, it examines: *What application behavior is malicious?, Is there any hidden malware (rootkits) running in the virtual machine?, Are all the critical-security processes such as auto-update and auto-scan running in the virtual machine?.* The main concern with the Hypervisor-based tools is to detect the intrusions which may not get detected by the guest-monitoring tools. It also monitors the interaction between applications and operating system; however it cannot access all the TVM-layer contextual information directly. It leverages the VM introspection (VMI) approaches that was lacking in traditional IDS approaches. The hypervisor/VMM can not view the high-level semantics of the guest operating system's information

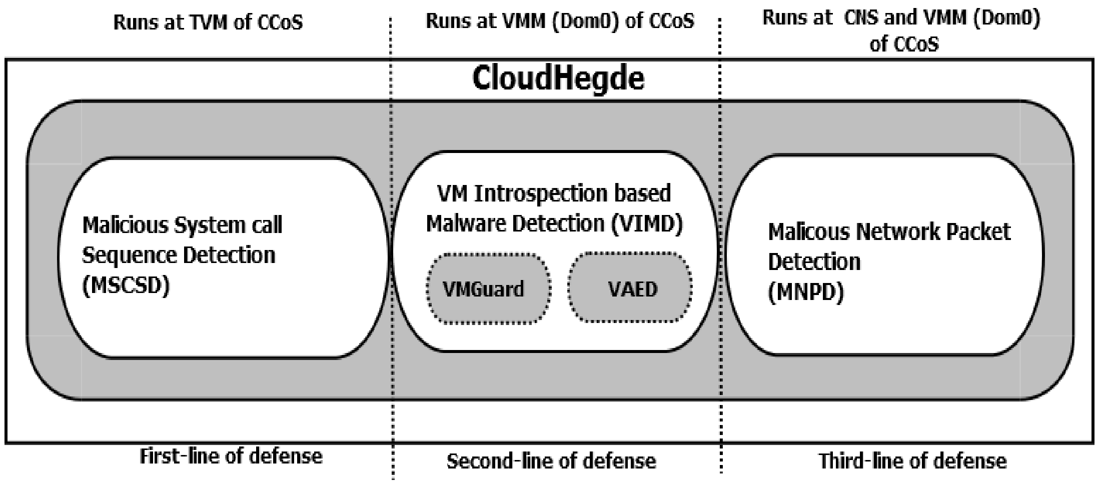

Runs at TVM of CCoS
Runs at VMM (Dom0) of CCoS
Runs at CNS and VMM (Dom0) of CCoS
CloudHegde
Malicious System call Sequence Detection (MSCSD)
VM Introspection based Malware Detection (VIMD)
VMGuard
VAED
Malicous Network Packet Detection (MNPD)
First-line of defense
Second-line of defense
Third-line of defense

FIGURE 3.1: Abstract Composition of CloudHedge instances monitoring at three-layers of cloud

as discussed in Section 3.1. VMI libraries provide possible ways to extract the high-level semantics of the machine from hypervisor. The information can be used for the intrusion detection approaches by the security practitioner. However, the applicability of the introspection approach for the hypervisor and its integration with an efficient detection mechanism is again a challenging task for intrusion detection in cloud. The proposed security approach for Hypervisor-based (out-of-the-guest) monitoring detects the intrusions at VMM-layer such as hidden processes, malware, subverted program modification attacks, evasion based attacks (time-based, processor feature based, exception-based). This is done by performing the program behavior analysis with memory introspection.

(iii) Network-based security solution: Network-based sub IDS instance examines the network traffic passing through a specific network segment for detecting the malicious network packets in the network. For example, it examines: *Is there any malicious network flow coming to or going from a monitored machine? Is the traffic coming from a machine is actually coming from the claimed sender's machine?, etc.* It is independent of the operating system over which it is installed and uses the network data as a primary source of information, rather than operating system logs as used by guest monitoring approaches. This makes the network-based systems extremely portable and hence can be deployed at various cloud layers such as TVM-layer, VMM-layer and Network-layer. In the proposed solution, the IDS sensors are

distributed at Network-layer and VMM-layer, out of the reach of the tenants. The configuration, control and monitoring of each sub IDS instance is governed centrally by cloud administrator. However, tenants may have their own NIDS tool installed running in their virtual machine, totally under the control of the tenants. This situation would occur if tenants do not opt for security from the cloud service provider. This proposed security approach for network traffic monitoring is a composite module to detect network intrusions both at Network-layer and VMM-layer such as SYN flooding, UDP flooding, scanning, etc., based on network behavior analysis with network introspection.

CloudHedge is a composition of three sub IDS instances which provide security solutions based on guest monitoring (TVM-based IDS), out-of-the-guest monitoring (Hypervisor-based IDS) and network traffic monitoring (Network-based IDS) respectively to deal with the attack detection at three different line of defense as discussed earlier as shown in Figure 3.1. Each of the sub IDS instance has different security functionalities to support the attack detection at various regions in cloud, covering all three layers. However, they all together form a single IDS instance (CloudHedge), which is monitored by cloud administrator. Each of the IDS sub instance provides one of the category of security solution provisioned by CloudHedge.

The three sub IDS instances are as follows:

- **Sub-IDS-1:** Malicious System Call Sequence Detection (MSCSD)
- **Sub-IDS-2:** VM Introspection based Malware Detection (VIMD)
- **Sub-IDS-3:** Malicious Network Packet Detection (MNPD)

MSCSD is a TVM-based IDS which provides TVM-layer security solution. VIMD is a Hypervisor-based IDS which provides VMM-layer security solution. VIMD provides two different detection strategies for malware detection. A cloud administrator can employ any of them or both at hypervisor layer in cloud. The first detection mechanism in the second-line of defense is VMI-assisted malware detection approach (called VMGuard) based on system call sequence analysis with memory introspection capabilities. The other detection mechanism is VMI-assisted evasive malware detection approach (called VAED) based on system call transition analysis with memory introspection capabilities. The third sub IDS instance, MNPD provides network traffic monitoring based security solution at both network and hypervisor-layer with network introspection capabilities.

All the sub IDS instances are part of CloudHedge. CloudHedge is an efficient framework as each of the IDS sub instance has been designed to detect intrusions at its different locations (inside the TVM or outside the TVM) and found to perform well when compared to other solutions. The different locations of CloudHedge have got different security-levels. IDS running in TVM are less secure than IDS running outside the TVM (at VMM and/or Network layer). Hence, even if the first-line of defense mechanism is breached, the second and third-line of defense mechanism would still be running actively at their locations. For example, malware attacks bypassed by MSCSD can be detected from outside the TVM by VIMD. However, all attacks cannot be detected by analyzing the program behavior. These attacks includes the ones which do not harm the programs installed in monitored machines but unnecessarily keep the system's resources busy (eg. Denial of Service (DoS) and scanning attacks). Some other attacks may run malicious programs to generate a flood of packets. The detection of these attacks becomes more difficult by security tools if the flooding is done with spoofed address. Therefore, network traffic monitoring is as important as process-monitoring for attack detection in cloud. CloudHedge provides two security-levels in the third-line of defense from network intrusions. It performs the network-trace analysis at the Network-layer, providing primary security from attacks. It also supports the traffic validation at the VMM-layer using network introspection to check the legitimacy of the monitored TVM. It then carries out network-trace analysis using machine learning techniques at VMM-layer, providing the secondary security from attacks.

3.2.2 Deployment of CloudHedge in Cloud

The proposed security approach adds security functionalities at the TVM-layer and VMM-layer of Cloud Computer Server (CCoS) and Network-layer of Cloud Network Server (CNS) as shown in Figure 3.2. The implementation set up is based on Xen VMM for hosting the TVMs. We have considered the cloud architecture based on OpenStack [16], a leading global cloud management software. Openstack is a collection of open source cloud components used which is popular for developing cloud platform for public, private and hybrid cloud [21]. Various companies such as RackSpace public cloud [25], ELASTX

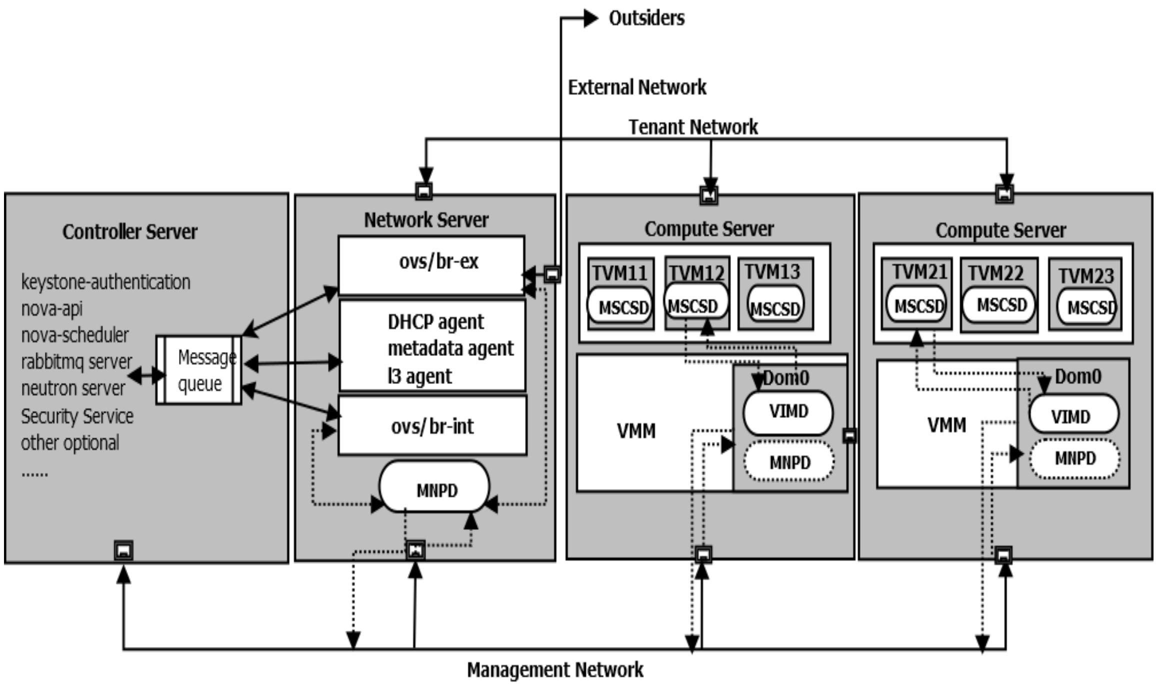


FIGURE 3.2: A security design architecture and deployment of various sub IDS instances in cloud

OpenStack:IaaS [27], Dualtec [28], AgileCLOUD [29], offer OpenStack powered public cloud services, as discussed Chapter 1. Xen is an open source hypervisor supported by commercial CSPs such as Amazon and Citrix for hosting TVMs. The privileged domain (Dom0) of Xen is used to configure, monitor and control the TVMs which are referred as DomUs (untrusted domains). However, cloud administrator enforces strict policies at VMM to restrict DomU users from accessing Dom0. Each sub IDS instance of CloudHedge runs at different locations in cloud. One sub IDS instance of CloudHedge runs at TVM to deal with basic malware attacks and provides the first-line of defense from attacks. Another instance of CloudHedge runs at Dom0 of VMM-layer to deal with advanced malware attacks and provides the second-line of defense. The third instance of CloudHedge runs at both Dom0 of CCoS and host OS of CNS to deal with network attacks and provides third-line of defense. CloudHedge performs the analysis on process logs and network logs extracted from TVM memory and virtual switch. If TVM is found to be suspicious, alert signals are sent to cloud administrator with details of logs generated by CloudHedge.

CloudHedge is a distributed deployment approach where each of the sub

IDS instance responds to cloud administrator on detection of the malicious activity. The overall architecture of the cloud has been taken into consideration. Each of the CloudHedge instance runs individually at TVM, VMM and/or network server in the cloud but they are configured and controlled by the cloud administrator at Cloud Controller Server (CCS). A tenant member has no control over the configuration, monitoring and control of CloudHedge instances. As the sub IDS instances are distributed in nature, they do not share the same VM resources for their operations, reducing the resource overhead at individual tenant TVM. Moreover, the IDS-instances which are deployed outside the TVM, reduce the fear for IDS subversion at individual TVM.

A tenant member may not be aware about the IDS instances running in the cloud as he/she has no control over any of the IDS sub instance. The cloud administrator is only responsible for creating policies and responding to alerts generated by sub IDS instances. A summary of attack statistics gathered from various locations is made available to cloud security team by cloud administrator so that the attacks can be further analyzed with the expertise knowledge. The frequency of attacks, associated TVM and applications responsible for causing attacks are identified. A suitable response is initiated by cloud administrator. CCS acts as a central point for log management, gathering attack statistics, creating and modifying security policies etc. This helps the cloud administrator to identify how the current security solutions can be enhanced to reduce the attacks on cloud. The brief summary of each of the sub IDS module with their functionality is described in subsequent sections.

3.3 Malicious System Call Sequence Detection (MSCSD)

The aim of Malicious System Call Sequence Detection (MSCSD) is to provide a security technique that is applicable to traditional physical hosts and all cloud deployments such as SaaS, PaaS and IaaS. A CSP can apply the MSCSD at TVM-layer to provide the first-line of defense from basic malware attacks which generally disclose their behavior on execution. MSCSD is based on the runtime behavioral analysis of the programs, running in the tenant VM. The conceptual

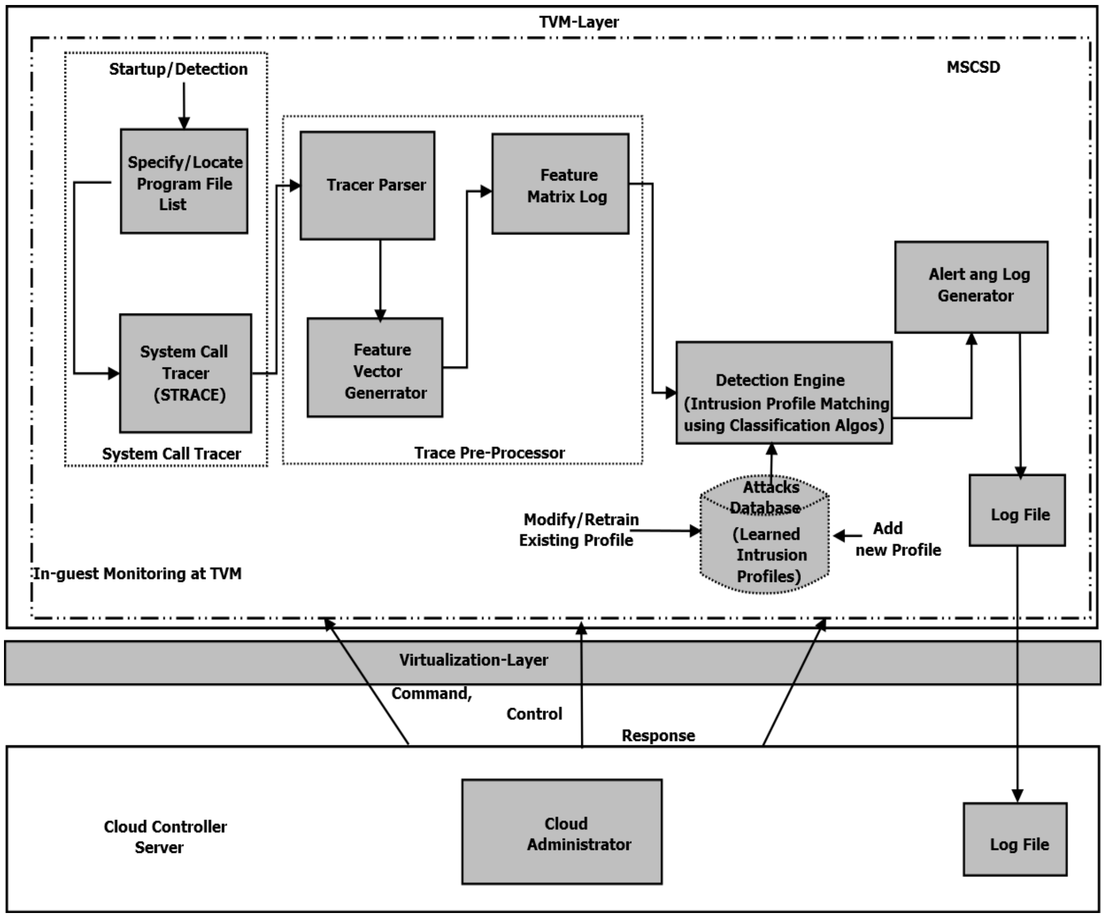

TVM-Layer
MSCSD
Startup/Detection
Specify/Locate Program File List
System Call Tracer (STRACE)
System Call Tracer
Tracer Parser
Feature Matrix Log
Feature Vector Generrator
Trace Pre-Processor
Alert ang Log Generator
Detection Engine (Intrusion Profile Matching using Classification Algos)
Attacks Database (Learned Intrusion Profiles)
Modify/Retrain Existing Profile
Add new Profile
Log File
In-guest Monitoring at TVM
Virtualization-Layer
Command,
Control
Response
Cloud Controller Server
Cloud Administrator
Log File

FIGURE 3.3: Conceptual diagram for in-guest monitoring by MSCSD in cloud environment

diagram of MSCSD is shown in Figure 3.3. In initial phase, MSCSD extracts the execution traces of monitored programs (located in program file list) in form of system call logs using system call tracer. Trace pre-processor then parses the tracing logs and extracts the system call sequences (traces) to generate the features. MSCSD provides a feature extraction approach, called 'Bag of n-grams (BonG)'. It finds out the sequence structure of the various short sequences of same size, called n-grams. It then converts the traces into a numeric feature vector $< c_1, c_2, c_3, c_4, c_k >$. Each entry c in feature vector represents the occurrences of individual short sub-sequences in the trace.

BonG considers both frequency and structure of various short sequence of system calls patterns of each trace. It is therefore successful in maintaining the ordering of the subsequent system calls within each subsequence. The extracted features are stored in Feature Matrix Log (FML) which is passed to detection engine (DE). In learning phase of DE, machine learning technique (Decision Tree C 4.5) is applied to learn the behavior patterns of feature vectors of observed system call patterns. In detection phase of DE, the trained model is used to analyze the behavior of running processes. If any suspicious activity is detected, alerts are sent to cloud administrator. The key advantage with MSCSD is that it improves the accuracy while maintaining the ordering of system calls. MSCSD observes the run-time behavior of the programs, hence it is free from anti-detection techniques such as obfuscation and encryption. MSCSD has been validated with University of New Mexico (UNM) [155] datasets. It provides various advantages over existing dynamic analysis approaches proposed for cloud. MSCSD is applicable to traditional physical hosts and virtual machines running in cloud environments. However, the TVM-layer technique will fail to detect advanced malware attacks which hide their behavior after sensing the presence of security tool. There is need to provide a more strong line of defense in cloud.

3.4 VM Introspection based Malware Detection (VIMD)

The goal of VM Introspection based Malware Detection (VIMD) is to detect attacks at the VMM-layer, providing second-line of defense in cloud. A cloud administrator can control and monitor the VIMD sub IDS instance from Dom0 of VMM, which helps to prevent IDS subversion from untrusted domains (TVMs). VIMD provides both primary and secondary security checks. It operates in three phases: memory introspection phase, behavior analysis phase and logging & alerting phase. In memory introspection phase, primary security check is performed to ensure whether any of the hidden malicious processes are running in TVM. It also ensures whether all security-critical processes are running properly on TVM. It then extracts the execution traces of running processes by using the kernel debugging based VM introspection mechanism [124]. The introspection mechanism employed by VIMD uses the software break point injection at guest OS kernel function. However, the break points are hidden using Extended Page Table (EPT) mechanism. Deng et al. [156] have proved the effectiveness of the combined use of EPT protection with break points to hide from

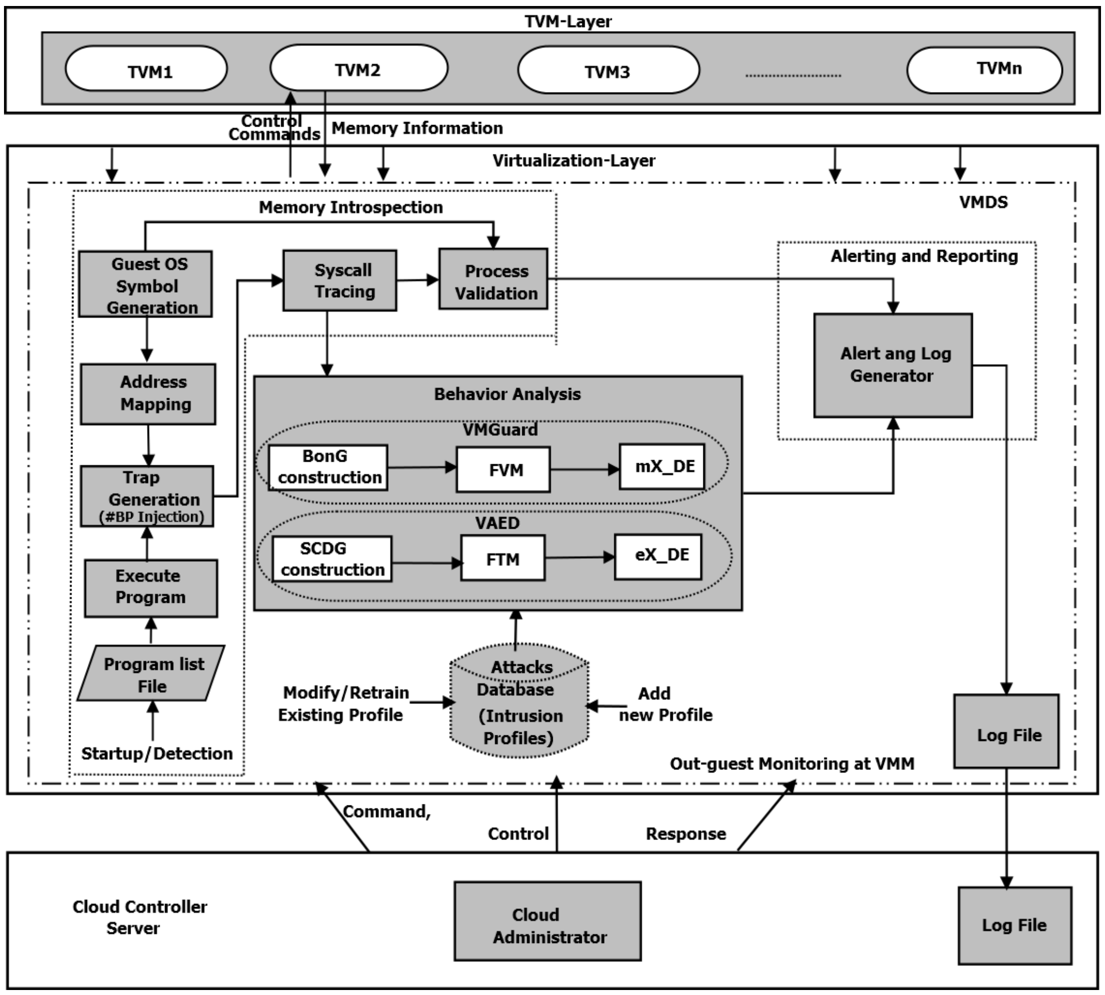


FIGURE 3.4: Conceptual diagram for out-of-the-guest monitoring by VIMD in cloud environment

advanced anti-debugging techniques. VIMD utilizes the Rekall memory forensic framework [157] to obtain the details of guest OS kernel symbols and their address locations as these details are not provided by commercialized operating systems. On executing the programs, a detailed behavioral log of all the processes are obtained which are passed to behavior analysis phase. In behavior analysis phase, the secondary security check is performed to detect the malicious processes running in monitored TVM. VIMD provides two detection mechanism named as VMGuard and VAED which capture and detect the program semantic behavior of different malware attacks at the VMM-layer. The conceptual diagram of VIMD is shown in Figure 3.4. The description of each of the detection mechanism is given below:

3.4.1 VMGuard: VMI-assisted Malware Detection Approach based on System Call Sequence Analysis at the Hypervisor

The first detection mechanism, called as VMGuard, is a security solution at VMM-layer in cloud. VMGuard is designed to detect the malicious activities of TVMs which can be bypassed by MSCSD. VMGuard is one of the core detection components of VIMD. It performs the behavior analysis on the extracted traces obtained from memory introspection phase of VIMD. VMGuard improves the BonG feature representation by integrating it with with text mining approach for feature selection particularly Term Frequency-Inverse Document Frequency (TF-IDF) and ensemble learning technique. BonG generates the feature vectors based on the frequency distribution of all possible unique n-gram values with respect to each class. VMGuard applies the text mining approach with generated feature vectors to improve the discriminative power of n-grams. It considers two major factors: Frequency and Rarity. Frequency refers to "how frequent an n-gram is in each trace?" and Rarity refers to "how rare an n-gram is in a collection of traces?". This is measured as a TF-IDF score for each n-gram.

VMGuard considers this score as feature selection criteria to generate the feature vector matrix (FVM). The selected features (stored in FVM) are passed to the mX_DetectionEngine (mX_DE). In learning phase, mX_DE uses Random Forest classifier to learn the behavior of attacks. Random Forest trains the multiple decision trees and applies bagging to aggregate their result to provide the combined output. The trained model is used to detect the malicious patterns in the detection phase. Alert signals are generated and sent to cloud administrator on detection of any suspicious activity. VMGuard has been validated with University of New Mexico (UNM) [155]) dataset. VMGuard is well suited to provide secondary security functions in virtualization based cloud environment.

3.4.2 VAED: VMI-Assisted Evasion Detection Approach based on System Call Transition Analysis at Hypervisor

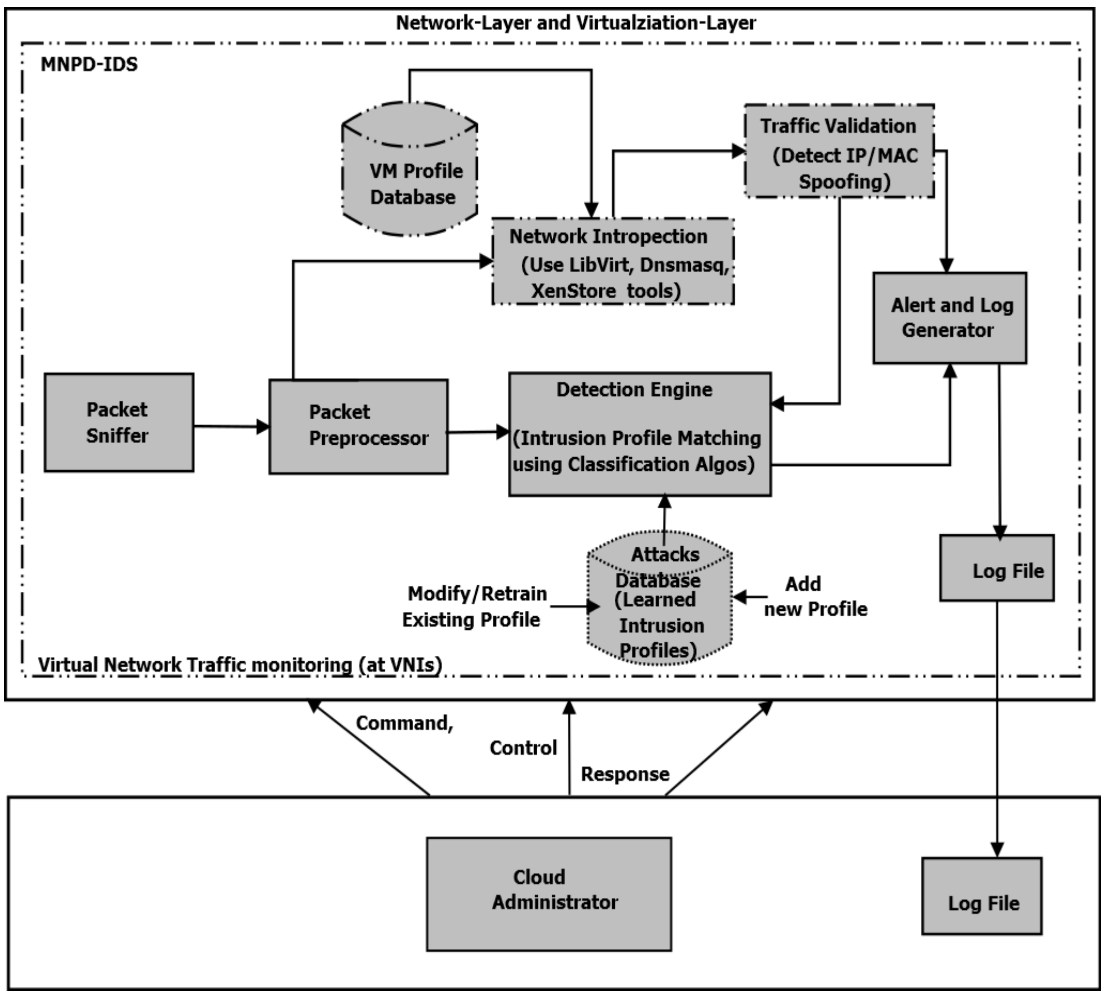
The second detection mechanism, called as VAED, is a security solution at VMM-layer in cloud which can also be used by CSP to provide the second-line of defense from advanced evasive malware attacks. The main objective of VAED is to provide a program semantic based VMI-assisted evasion detection mechanism for detecting stealthy evasion-based malware attacks from VMM-layer in cloud environment. Once the execution traces are obtained from memory introspection phase, VAED constructs the program semantics in form of the System Call Dependency Graphs (SCDGs) to analyze the semantics in different execution paths of the programs installed on TVM. SCDG represents the system call transitions in different execution paths of the program. The semantics are based on the ordered sequence of system calls with the analysis of transition probabilities from one system call to other possible system call. The transition probabilities are calculated by using the Markov Chain property which is basically a derived form of 2-gram model which extends the frequency model and uses the probability model to represent the system transitions in a more efficient way. Each of the SCDG is stored in form of adjacency matrix which stores all possible transition probabilities for each transition. It forms a basic building block in forming feature transition matrix (FTM).

VAED employs the feature selection method, particularly Information Gain Ratio (IGR) to select the important system call transitions. Now eX_DetectionEngine (eX_DE) learns the behavior of attacks. It applies an ensemble learning approach in which different classifiers are trained for the FTM and weighted voting scheme is used as a fusion rule to fuse the diverse classifiers results. The fused results are used to predict the malware class for intrusive processes in the detection phase. The fused model is used as a baseline information to test the semantic (run time) behavior of programs running in monitored VM. If any suspicious activity is detected by eX_DE, alerts are generated and sent to cloud administrator. VAED is also well suited to provide secondary-line of defense in virtualization based cloud environment. It has been validated with evasive attack dataset obtained from University of California [158].

VIMD is deployed at virtualization-layer in cloud and monitored by CSP to provide the secondary-line of defense from both basic and advanced malware (evasive malware) attacks. The proposed approach is superior to other approaches, as on one hand, it is difficult for an attacker to subvert the security monitor from TVM, while on the other hand, it provides efficient techniques for attack detection at hypervisor. It has been validated with different datasets and results seem to be promising.

3.5 Malicious Network Packet Detection (MNPD)

The goal of Malicious Network Packet Detection (MNPD) is to provide a network traffic monitoring based malicious network packet detection approach in cloud. It monitors the TVMs from outside the monitored machine at both Network and VMM-layer in cloud environment and provides the third-line of defense from attacks. MNPD is configured to listen on virtual network interfaces (VNIs) of Cloud Network Server (CNS) and Cloud Compute Server (CCoS). MNPD performs the behavioral analysis of network traffic at CNS; providing the primary security from network intrusions at Network-layer. MNPD also provides VM network introspection to gain the VM related information using open source tools such as LibVirt, XenStore, dnsmasq server from Dom0 of hypervisor of CCoS. The information is later used to perform traffic validation at virtualization layer of CCoS to detect spoofing attacks originated from monitored TVM. However, an attack can be targeted using correct Source IP and/or MAC address. Hence, non-spoofed packets are further analyzed using behavior analysis of network traffic to detect any abnormality in the virtual traffic; providing secondary security from intrusions at virtualization-layer. The traffic is captured using the packet sniffer tools. Packet pre-processor converts the captured packets (.pcap files) into standard format (.CSV or .ARFF file) which contains various traffic features. Relevant features are extracted using the ensemble of feature selection methods (chi square and Recursive Feature Elimination) and stored in Optimal Feature Statistics Matrix (OFSM). MNPD performs behavior analysis of traffic at both the servers using detection engine (DE). DE applied Random Forest Classifier (RF) to learn and



Network-Layer and Virtualziation-Layer
MNPD-IDS
VM Profile Database
Traffic Validation (Detect IP/MAC Spoofing)
Network Intropection (Use LibVirt, Dnsmasq, XenStore tools)
Alert and Log Generator
Packet Sniffer
Packet Preprocessor
Detection Engine (Intrusion Profile Matching using Classification Algos)
Attacks Database (Learned Intrusion Profiles)
Modify/Retrain Existing Profile
Add new Profile
Log File
Virtual Network Traffic monitoring (at VNIs)
Command,
Control
Response
Cloud Administrator
Log File

FIGURE 3.5: Conceptual diagram for network traffic monitoring by MNPD in cloud environment

detect network attacks such as DoS, Scanning etc. On detection of any suspicious activity, alert signals are generated and sent to cloud administrator. MNPD does not involve overhead incurred in monitoring extensive memory writes or instruction-level traces. It is a more secure solution to detect network attacks which never pass through physical interface and hence not detected by traditional IDS. The proposed approach has been validated with latest datasets (UNSW-NB and ITOC) and results seem to be promising. The conceptual diagram of MNPD is shown in Figure 3.5.

3.6 Conclusion

A robust security architecture, *CloudHedge* has been proposed for detecting intrusions in cloud environment which provides three lines of defense to detect both malware and network attacks at various regions in cloud. The limitations in the existing security solutions have been well investigated and handled by CloudHedge. Our security architecture provides three-lines of defense which make our architecture more efficient and robust when compared to other cloud security approaches. The design of CloudHedge is distributed and motivated from the fact that a centralized IDS becomes a bottleneck when number of TVMs in the cloud host increase and also when the security tool uses centralized resources. If all TVMs are monitored at the central location i.e. at Cloud Controller Server (CCS), it adds a big overhead to the cloud administrator which results in inefficiency, privacy and scalability issues.

The tenants who are more concerned about their privacy can opt for TVM-layer security solutions as tenant data will never go out of their machine since security tool runs inside the monitored machine. Tenants who are more concerned about security and at the same time who do not want to compromise much on privacy can also opt for higher layer line of defenses i.e. second and/or third. The privacy concerns are clarified between tenant and cloud service provider (CSP) in form of service level agreement (SLA) at the time of registration. The more the flexibility a tenant provides to CSP for accessing its applications, memory and CPU states; the more the security a CSP can provide to tenants by using various layers of security functions provided by CloudHedge. We have worked on following aspects:

* Security approach for performing TVM-based monitoring at TVM-layer (first-line of defense)
* Security approaches for performing Hypervisor-based monitoring at VMM-layer (second-line of defense)
* Security approach for performing Network-based monitoring at Network and VMM-layer (third-line of defense).

CloudHedge is designed to deal with attacks at different cloud layers. Each of the security solution addresses the complexities associated with each layer and applies the suitable security functions at the monitoring layer. The security solutions have also been categorized based on security-layers which makes the CSP to assign the specific security solution based on the tenants demands. However, there are some limitations associated with CloudHedge. Currently, the support of combining and co-relating the alert information obtained from various layers is not supported by CloudHedge. Each instance can individually detect a set of attacks. The fusion of their output can be helpful to detect the distributed attacks in cloud. In future, a mechanism will be provided to correlate the information obtained from various sensors, deployed at different layers to improve its efficiency and attack detection power.

In the subsequent chapters i.e. Chapter 4 (Malicious System Call Sequence Detection), Chapter 5 (VM Introspection based Malware Detection) and Chapter 6 (Malicious Network Packet Detection), the design and implementation of individual sub IDS instances of CloudHedge is discussed in more detail.

Chapter 4

Malicious System Call Sequence Detection (MSCSD)

This chapter describes the design and implementation of 'Malicious System Call Sequence Detection (MSCSD)', one of the sub IDS instance of CloudHedge that is based on the run-time behavioral analysis (dynamic analysis) of the programs at the Tenant Virtual Machine (TVM)-layer. MSCSD sub IDS instances are distributed in each monitored TVM, configured and controlled by cloud administrator. The security architecture of MSCSD along with various detection components is described in detail.

4.1 Introduction

The flexibility and scalability in cloud services has opened door for attackers. Any vulnerability present in cloud, can allow the attacker to gain illegal privileges of tenant virtual machine (TVM) users. A malicious user can install advanced malware programs and gain higher access privileges (guest OS kernel privilege). A compromised guest OS kernel can call malicious drivers and perform malicious actions. Once a virtual machine is fully compromised, an attacker can try to launch further attacks on other TVMs. A compromised virtual machine is a big threat to cloud infrastructure as it can bypass the security of other TVMs. It could further lead to monetary disputes between cloud service provider (CSP) and legitimate VM users.

Some researchers [68] [91] [123] used signature matching techniques as a detection mechanism in their security architecture which maintains a database of attack signatures. These techniques require a regular update of the signature database. An attacker can be successful by forming attacks patterns which can bypass the security mechanism of signature based tools. Dynamic analysis is one of the intrusion detection approaches to learn the run time behavior of the programs running in VMs. Some researchers [67][84] applied dynamic analysis based intrusion detection techniques in cloud which maintain a baseline profile of normal behavior of tenant's applications. Any deviation from baseline profile, generates an alarm to cloud admin. This will result in more false positives for evolving new behavior of users in dynamic cloud environment. Moreover, without applying a learning based mechanism, these approaches require a huge storage to store the training database for the monitored programs.

For example, Gupta et al. [84] have proposed enumeration-based approach, called Immediate System Call Sequence (ISCS) approach in cloud environment in which a baseline database is created for all monitored programs in a virtual machine in the form of key-value pairs. A key refers to the system call name and value refers to the subsequent sequence of system calls following it. The technique will fail for longer or infinite length traces. Further, Alarifi et al. [67] and abed et al. [148] have implemented frequency-based approach, called 'Bag of system calls (BoS)' [128] approach for detecting intrusions in cloud. The technique leads to reduction in storage as traces are converted to a numeric feature vector which contains frequency count of each system call but the ordering of information is lost. The ordering of system calls is very important for attack behavior identification. Therefore, we need an intermediate solution which will preserve the characteristics of both enumeration-based and frequency-based approaches.

In TVM-based monitoring, the detection mechanism runs inside the monitored TVM. Advanced security techniques such as Virtual Machine Introspection (VMI) [122] can be used for virtualization and cloud environments which is helpful in introspecting the TVM memory from outside at the hypervisor, called as Hypervisor-based monitoring or out-of-the-guest monitoring. However, TVM-based monitoring has some advantages over hypervisor-based monitoring as discussed below:

∗ TVM-based security tools have excellent visibility into the monitored host and have the highest attack detection efficiency.

∗ The techniques are applicable for all the cloud deployments including IaaS, PaaS, SaaS.

∗ CSPs can apply TVM-based security tools without any modification in the hypervisor or hardware design.

∗ This is the only way for privacy concerned tenants who do not want to disclose their information for securing their VMs from different types of attacks.

∗ Even if the tenants opt for Hypervisor-based security solutions, TVM-layer security will always be needed as a basic security solution because of lesser complexity and higher privacy.

Therefore, TVM-based security tools are better suited to provide first-line of security defense. These type of IDS make use of In-VM security approaches which run inside the TVM. Out-of-the-VM security approaches for malware detection, run outside the VM at a secure location in hypervisor and used by Hypervisor-based IDS. These details of these approaches are discussed in Chapter 5.

The aim of our work is to develop a more efficient security technique that is applicable to traditional physical hosts and all cloud deployments such as SaaS, PaaS and IaaS. We propose 'Malicious System Call Sequence Detection (MSCSD)' approach that is based on the run-time behavioral analysis (dynamic analysis) of the programs, running individually in the tenant VMs and centrally controlled and coordinated by cloud administrator. Dynamic analysis of a process involves capturing the run-time behavior of a program in the form of a sequence of system calls (trace) executed by it. System calls reveal how the process is interacting with the operating system, which is key in identifying how malicious processes perform attacks. Although a number of different malicious programs exist which misuse different types of vulnerabilities (such as stack-based buffer overflow), there is always a sequence of specific system calls which is/are often common in executing the exploits. System calls can be invoked by both programs in the user space as well as by privileged system code.

MSCSD extracts and analyses the execution traces of the monitored programs. It considers various short sequence of system call patterns, called n-grams (same size) of each trace and therefore successful in maintaining the ordering of the subsequent system calls within each sub-sequence. Each trace is converted to a numeric feature vector where each entry of feature vector represents the occurrences of unique n-grams in the trace. In next phase, machine learning technique (Decision Tree [159]) is applied to learn the behavior patterns of observed system call sequences which will serve as baseline profile for future test instances. MSCSD is successful in improving the accuracy and reducing the storage requirement while maintaining the ordering of system calls. MSCSD is applicable to traditional physical hosts and as TVM-based security tools for virtualization and cloud environment, monitored by cloud administrator. Tenants have no access to any of the MSCSD detection components and its configuration files. CSP can adopt MSCSD as basic security mechanism to deal with attacks at TVM-layer.

4.2 MSCSD: Security Design

MSCSD as a sub IDS instance of CloudHedge, is designed to detect the malicious system call sequences by monitoring the behavior of programs for any malicious actions. MSCSD stands out from the existing approaches [67][148] for program behavior monitoring as it follows a distributed detection which is controlled and configured by the cloud administrator at the centralized server. This solution answers some of the research gaps identified and discussed in Chapter 2. The key characteristics are described along with design choices, below:

i. MSCSD is a TVM-based security tool which is hypervisor independent and centrally controlled by cloud administrator to detect malicious execution of programs at the tenant VMs.
ii. The deployment of MSCSD sub IDS daemons is distributed in nature as the IDS instances run independently in all the monitored TVMs. All the sub IDS daemons are centrally configured and controlled by cloud administrator. It reduces the bottleneck situations which can occur for the security architectures where the whole monitoring is done at the centralized cloud server.
iii. It performs the run time behavior analysis of programs at the monitored machine which makes it robust from obfuscation and encryption based attacks where an attacker changes the code in a way to

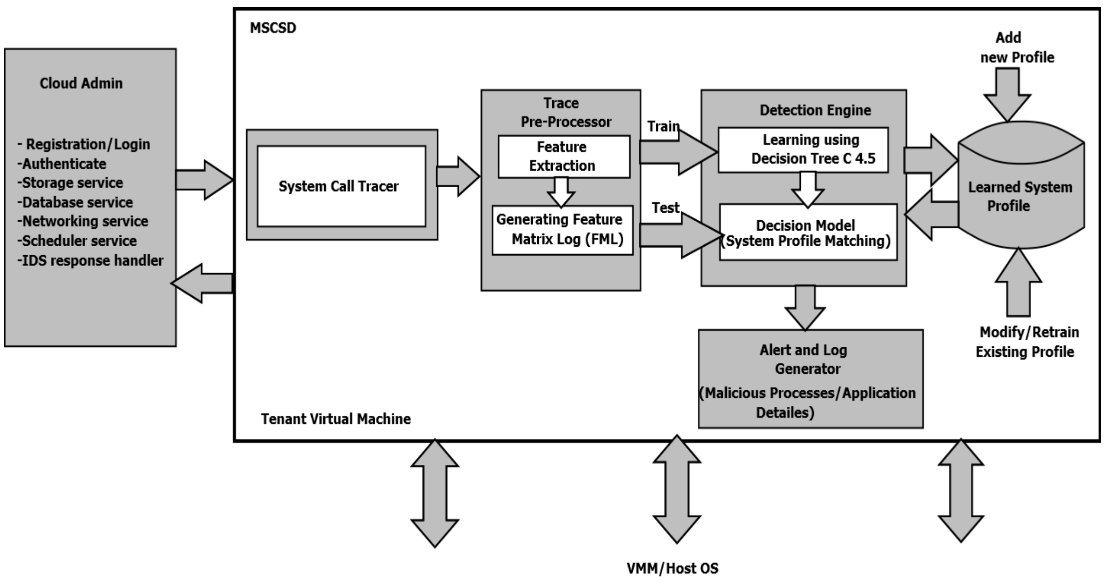


Figure 4.1: Security architecture of MSCSD

thwart the detection schemes, specially schemes that are based on static and signature based analysis.

iv. The solution is better than other solutions where the individual IDS sub-daemons are distributed among VMs without having any central control and reporting. The sub IDS daemons in this situation can be controlled by tenant users. They can directly update the security policies which make the system insecure from nefarious users which can subvert and stop the security policies without getting noticed by cloud admin.

v. The proposed solution does not make use of any complex introspection mechanism for trapping the system call sequences. It performs the analysis based on the frequency count of various ordered subsequence of system calls to detect an intrusion. This increases the detection accuracy and resource efficiency, which is an important requirement for cloud environments.

vi. It takes the advantage of both misuse detection and anomaly detection as it is based on the analysis of both intrusive and normal sequences which makes it better than well known anomaly detection approaches [67] [148] [154] proposed for cloud. The use of machine learning techniques removes the need of storing all possible system call sequences as needed by existing approaches such as key-value pair [84].

At a high-level, there are four detection components: System Call Tracer, Trace Pre-Processor, Detection Engine and Alert and Log Generator as shown in the security design of MSCSD, in Figure 4.1. All these components of MSCSD execute in the monitored TVM. System Call Tracer retrieves the system call logs of monitored programs running in the TVMs. It is configured to locate the monitored programs from the file specified by cloud administrator. Trace Pre-Processor parses the logs and extracts the system call sequences (traces). Each trace is processed by a sliding window and short system call sub-sequences (n-grams) are generated. Numeric feature vectors are generated and stored in feature matrix log (FML) for all traces. FML is given as a input to Detection Engine. Detection Engine learns the behavior of the programs by using a machine learning algorithm to train itself for the supplied behavioral pattern. The monitored programs are checked by trained classifier (Decision Model) for their similarity and dissimilarity from the learned behavior. An alert is generated by alert and log generator (ALG) to cloud admin if suspicious behavior is detected . All the IDS daemons, are centrally controlled by cloud administrator at the cloud controller server which is responsible for taking further actions after receiving the alert from MSCSD tool running in TVM. The execution flow of various security functions of these components is shown in Figure 4.2. Various detection components of MSCSD are explained in the following subsections.

4.2.1 System Call Tracer

System Call Tracer (SCT) extracts the run-time behavior of the monitored program in the form of sequences of system calls invoked by each of them. The MSCSD sub IDS instance attaches a system call tracer (`strace`) to all the running processes to create the individual system call trace log for each of them. The `strace` utility produces the traces of monitored programs. Monitored programs can either be decided by the tenant member and communicated to cloud admin at the time of registration or automatically taken from program file of monitored machine. The `strace` of monitored program produces a long list of system calls, called a trace. For example, let us consider a system call trace corresponding to a user utility program of linux as shown in Figure 4.3. The sequence of open(), read() and write() system calls, executed in the

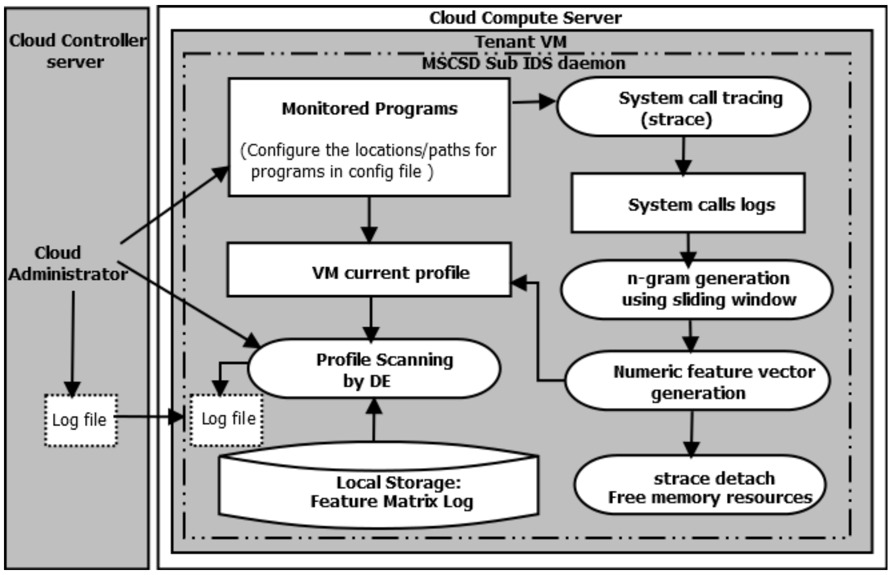
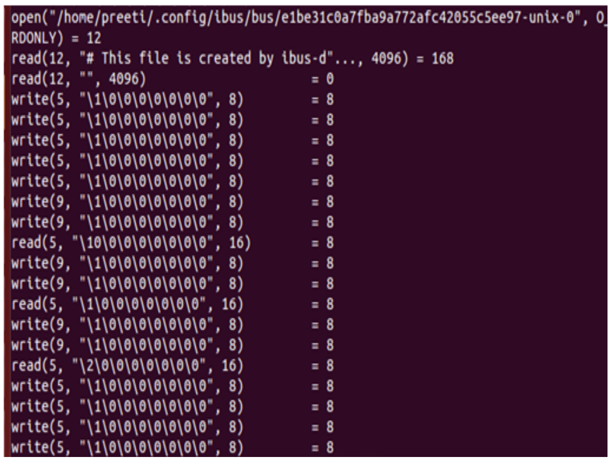

Cloud Controller server
Cloud Administrator
Log file
Cloud Compute Server
Tenant VM
MSCSD Sub IDS daemon
Monitored Programs
(Configure the locations/paths for programs in config file)
System call tracing (strace)
System calls logs
VM current profile
n-gram generation using sliding window
Profile Scanning by DE
Numeric feature vector generation
Log file
Local Storage: Feature Matrix Log
strace detach Free memory resources

Figure 4.2: Execution flow of various security modules of MSCSD



```
open("/home/preeti/.config/ibus/bus/e1be31c0a7fba9a772afc42055c5ee97-unix-0", O_
RDONLY) = 12
read(12, "# This file is created by ibus-d"..., 4096) = 168
read(12, "", 4096)                      = 0
write(5, "\1\0\0\0\0\0\0\0", 8)         = 8
write(5, "\1\0\0\0\0\0\0\0", 8)         = 8
write(5, "\1\0\0\0\0\0\0\0", 8)         = 8
write(5, "\1\0\0\0\0\0\0\0", 8)         = 8
write(5, "\1\0\0\0\0\0\0\0", 8)         = 8
write(9, "\1\0\0\0\0\0\0\0", 8)         = 8
write(9, "\1\0\0\0\0\0\0\0", 8)         = 8
read(5, "\10\0\0\0\0\0\0\0", 16)        = 8
write(9, "\1\0\0\0\0\0\0\0", 8)         = 8
write(9, "\1\0\0\0\0\0\0\0", 8)         = 8
read(5, "\1\0\0\0\0\0\0\0", 16)         = 8
write(9, "\1\0\0\0\0\0\0\0", 8)         = 8
write(9, "\1\0\0\0\0\0\0\0", 8)         = 8
read(5, "\2\0\0\0\0\0\0\0", 16)         = 8
write(5, "\1\0\0\0\0\0\0\0", 8)         = 8
write(5, "\1\0\0\0\0\0\0\0", 8)         = 8
write(5, "\1\0\0\0\0\0\0\0", 8)         = 8
write(5, "\1\0\0\0\0\0\0\0", 8)         = 8
```

Figure 4.3: strace of a VM user utility Program

normal execution scenario of the a user utility program, can be seen. Each trace log is stored in form of the two columns where first column represents the process id and second column represents the ordered sequence of system calls. All the trace logs of the monitored programs are merged to create a behavior log which is later processed by other components to represent the TVM profile.

System call tracing works in two stages, described below:

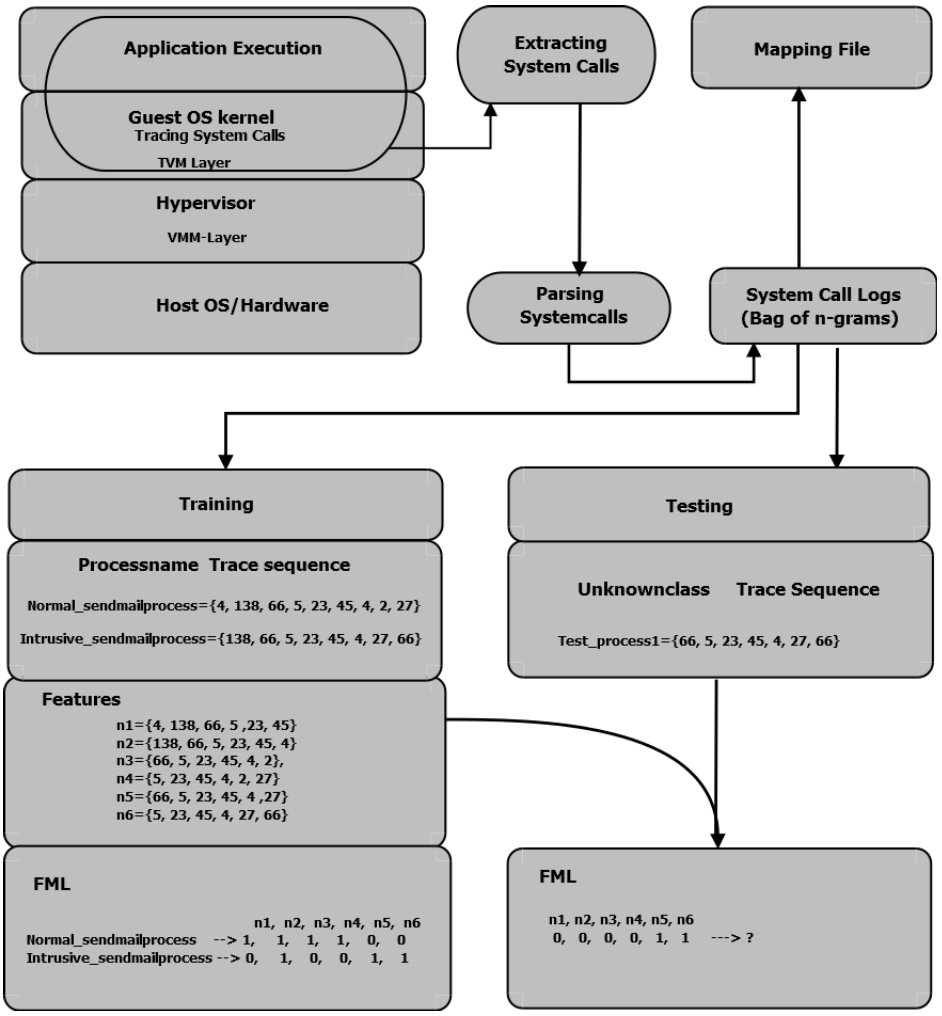
∗ **Startup:** On receiving the startup command, MSCSD executes the programs mentioned in the file of list of programs and performs the execution tracing on the running processes as explained above. The system call logs are transferred to the next detection component i.e. Trace Pre-Processor for extracting the features from the run time behavior of the programs. This command is generally fired by the cloud administrator whenever a new tenant VM is allocated to the user.

∗ **Update:** On receiving the update command, MSCSD again runs the `strace` for the newly updated programs and obtains behavior logs of system call sequences. The command is generally fired by the cloud administrator when a new set of programs are installed by tenant which needs security. The initial file of list of programs is also updated to incorporate the changes requested by the tenants. The output is supplied to next phase in the similar manner as in the startup phase to do the feature extraction from logs and obtaining the behavior of newly installed programs.
The two phases are very important for behavior extraction which is analyzed for the intrusion detection in later stages. A behavioral log file (Log1) is prepared from each monitored VM. This represents various possible behaviors of monitored programs in the TVM.

4.2.2 Trace Pre-Processor

After generating program behavioral logs, pre-processing of each trace is carried out to obtain a large collection of n-gram sequences for each trace (present in Log1) by Trace Pre-Processor (TPP). TPP extracts features inform of n-gram sequences. An n-gram is a short sequence of system calls obtained by considering a sliding window of size n and gradually shifting the window by 1. Each shift of the window provides a unique n-gram sequence. MSCSD considers a sliding window with n=6 which is proved to be best size based on analysis carried out with various length of n-gram sequences by Warrender et. al [116]. For example, consider a system call trace of a user utility program as shown in Figure 4.3. The n-grams generated by traversing the window by 1 will be as follows:
n_1=open(), read() read(), write (), write (), write ();
n_2= read() read(), write (), write (), write (), write();
n_3= read(), write (), write (), write (), write(), write();
n_4= write (), write (), write (), write(), write(), write() and so on. All n-grams of a trace are stored in the Log2.

Lets say X_i is a feature vector of trace i, represented by $X_i = \{n_1, n_2, n_3, n_4, ...n_m\}$. X_i is a set of n-grams which is the



Application Execution
Guest OS kernel
Tracing System Calls
TVM Layer
Hypervisor
VMM-Layer
Host OS/Hardware
Extracting System Calls
Mapping File
Parsing Systemcalls
System Call Logs (Bag of n-grams)
Training
Processname Trace sequence
Normal_sendmailprocess={4, 138, 66, 5, 23, 45, 4, 2, 27}
Intrusive_sendmailprocess={138, 66, 5, 23, 45, 4, 27, 66}
Features
n1={4, 138, 66, 5 ,23, 45}
n2={138, 66, 5, 23, 45, 4}
n3={66, 5, 23, 45, 4, 2},
n4={5, 23, 45, 4, 2, 27}
n5={66, 5, 23, 45, 4 ,27}
n6={5, 23, 45, 4, 27, 66}
FML
n1, n2, n3, n4, n5, n6
Normal_sendmailprocess --> 1, 1, 1, 1, 0, 0
Intrusive_sendmailprocess --> 0, 1, 0, 0, 1, 1
Testing
Unknownclass Trace Sequence
Test_process1={66, 5, 23, 45, 4, 27, 66}
FML
n1, n2, n3, n4, n5, n6
0, 0, 0, 0, 1, 1 ---> ?

FIGURE 4.4: Description of the Bag of n-grams representation

output obtained after data preprocessing where m is total number of n-gram obtained after processing trace. Each n-gram is represented by $n_j =< s_1 s_2 s_3 s_4 s_5 s_6 >$ where $1 <= j <= m$, a substring of system call. After feature extraction, feature vector for the trace is generated. All features in the form of n-grams are converted into a numeric vector represented by $X'_i =< c_1, c_2, c_3, c_4, c_k >$ where each c value represents the total number of occurrences of each unique n-gram value in the trace and k is total unique n-grams obtained. After feature extraction and pre-processing, each feature vector (named as Feature_Vector) is written to the Feature Matrix Log (FML) with the labeled name of trace (benign or intrusive). The Feature_Vector entries are flushed and next trace is pre-processed to obtain next Feature_Vector in similar way. The process is repeated for all traces present in the Log1. Two file are maintained for each machine: FML

and mapping file. Final FML presents the feature vector matrix which contains the features for all the traces. A mapping file represents the unique n-grams executed at the machine by the monitored programs. To better understand the idea behind 'Bag of n-grams (BonG)', lets consider two traces of sendmail process in two execution scenarios: intrusive and benign. The Normal_sendmailprocess= {*write*→sethostid→sstk→open→setuid→setgid→write→fork→ alarm}
Intrusive_sendmailprocess= {*sethostid*→sstk→open→setuid→setgid→write→ alarm→sstk} .

For minimizing space and calculations, we have replaced each unique system call by a unique integer number. The n-grams obtained by following the above procedure of BonG, are shown in Figure 4.4. The actual values of each unique n-gram sample for sendmail process is given in Table 4.1. It can be observed that n_1, n_2, n_3 and n_4 are appearing once in the normal trace sequence of sendmail process. The n-gram n_2 is also appearing in intrusive trace, hence we can infer that there can be some similar system call sequences/patterns appearing in both intrusive and normal traces. However, n_5 and n_6 are appearing only in intrusive trace which can be the signatures for the attack patterns, hence important for detection. We have also taken a random trace: {sstk→open→setuid→setgid→write→alarm→ sstk}.

It can be seen that it contains the short system call patterns (i.e. n_5 and n_6; shown in Table 4.1) which appears in intrusive trace. If n_5 and n_6 appear rarely in normal traces and mostly in the intrusive traces, then the chances of the random trace to be classified as intrusive is high. Therefore, the frequency-based model of BonG approach plays a very important role in the classification of normal and intrusive processes. The Feature Matrix Log (FML) obtained by processing all traces, forms a baseline database and represents the behavior of the machine in the different execution scenarios. The obtained output is used to train the classifier as described in the next phase. For example, the obtained feature vector corresponding to a process named as 'ps' is discussed in Section 4.3 (refer Figure 4.6).

Table 4.1: Sample of unique n-grams of sendmail process

Feature variable	**N-grams representation**	**Actual n-grams**
n1	[4, 138, 66, 5, 23, 45]	[write, sethostid, sstk, open, setuid, setgid]
n2	[138, 66, 5, 23, 45, 4]	[sethostid, sstk, open, setuid, setgid, write]
n3	[66, 5, 23, 45, 4, 2]	[sstk, open, setuid, setgid, write, fork]
n4	[5, 23, 45, 4, 2, 27]	[open, setuid, setgid, write, fork, alarm]
n5	[66, 5, 23, 45, 4, 27]	[sstk, open, setuid, setgid, write, alarm]
n6	[5, 23, 45, 4, 27, 66]	[open, setuid, setgid, write, alarm, sstk]

4.2.3 Detection Engine

Detection Engine (DE) is one of the main components of MSCSD which is responsible for detecting the intrusions in the monitored tenant VM. There are two execution modes for the DE to operate: Learning mode and Detection mode.

Learning mode: DE is executed in this mode, once cloud administrator fires the `configure` command. In learning mode, the three components of MSCSD such as SCT, TPP and DE operate over a set of known normal and maliciously modified processes (intrusive), in an offline fashion. A TVM may contain user programs such as sendmail, login, perl script, JavaScript, xlock and other executables or system program such as ls, ping, dir etc. FML is generated which provides the run-time behavioral statistics of the monitored programs in different execution scenarios. The proposed approach is specifically designed for attacks against TVMs. The working of MSCSD in learning mode is shown in Algorithm 1. The workflow defining its specific actions are as follows:

∗ Monitored programs are executed in different execution scenarios and the execution traces are obtained. A behavior log is prepared for the machine, called as FML. Each row of FML is represented by <X'i, $L >$ where $L\varepsilon\{0,1\} >$. L is label for each feature vector $< X_i' >$. Here, 0 is representing malicious class and 1 is representing benign class. FML records the features of the programs using the Bag of n-grams approach.

∗ MSCSD now executes a machine learning algorithm: Decision Tree C 4.5 [159] to learn the run-time behavior of programs (stored in FML). It represents the behavioral profile of monitored VM which is used to detect the malicious processes in future.

∗ The trained classifier acts as baseline profile of the machine. The trained classification object is stored in the local storage which represents the normal and intrusive profile of the monitored programs. The stored information is later used for matching the abnormality in the running processes with the pre-compiled profiles.

Detection mode: Detection engine is executed in this mode once cloud

Algorithm 1: Algorithm for Learning the Behavior of benign and intrusive samples of monitored programs

```
Result: VM1 Profile, VM2 Profile...VMn Profile
Profile_Generation_Module();
for $VM_i$=1 to $VM_i$ <=n do
    Log1(Traces)=strace(monitored programs);
    for Each Trace i in Log1 do
        Label=Extract_name($Trace_i$);    // Each program has many labeled benign and
        intrusive traces
        while Trace i ≠ NULL do
            Feature_Vector=Trace_Preprocessor($Trace_i$);
        end
        Feature_Matrix_Log=Write(Label,Feature_Vector);
    end
    clf=DecisionTreeClassifier();
    X=Feature_Matrix_Log [ ,1:n-1] ;   // All rows from 2nd(index 1) to n (index n-1)
    columns
    Y=Feature_Matrix_Log[0] ;   // First column contains target class labels (index
    0)
    $Decision\_Model_{VMi}$=clf.fit(X, Y);
end
```

Algorithm 2: Algorithm for Detecting Maliciously Modified Monitored Programs

```
Result: Alert Logs
Detection_Module()
for $VM_i$=1 to $VM_i$ <=n do
    Test_Log1(Traces)=strace(monitored programs)
    for Each Trace i in Test_Log1 do
        while Trace i ≠ NULL do
            Feature_Vector=Trace_Preprocessor($Trace_i$)
        end
        Test_FML=Write(Feature_Vector)
    end
    Pred=Load($Decision\_Model_{VMi}$);
    Labels=pred.predict(Test_Feature_Matrix_Log)
    if Lables.find('intrusive') then
        Generate_Alert("Suspicious Activity Detected")–>CloudAdmin
    else
        No alarm Raised;
    end
    CloudAdmin–>Analysis(Log);
    CloudAdmin–>Respond();
end
```

administrator fires the `detection` command. In this mode, all the components of MSCSD such as SCT, TPP, DE and ALG operate over the monitored processes of TVM, in an online fashion. Whenever a tenant VM is allocated to a user, a program list file of monitored programs which is configured by cloud administrator is shared with MSCSD sub IDS as part of IDS deployment and initialization. Similar to learning, Test_FML is generated which provides the run-time behavioral statistics of the monitored programs. However, the class label for the monitored process are not known and are determined by the MSCSD. The working of MSCSD in detection mode is shown in Algorithm 2. The workflow defining its specific actions for attack detection are as follows:

∗ On receiving the detection command from cloud administrator, MSCSD sub IDS instance invokes SCT for each running process to fetch execution traces of the processes running on the virtual machine. Monitored programs are executed and the execution traces are obtained. A behavior log is prepared for the machine, called as Test_FML. Unlike learning mode, each row of Test_FML is represented by <X'i> where each x'_i is a feature vector for the monitored trace. The testing FML represents the features of the programs in terms of the bag of n-grams which is further processed for analysis.

∗ MSCSD now loads the trained classifier to analyze the running processes and detect any suspicious system call pattern occurring in the traces. The run-time behavior of the monitored program is processed by the trained classifier which acts as a decision model. The decision model perform the statistical calculations over the Test_FML to classify the instances in one of the attack classes based on the learning of the various types of execution traces.

∗ The behavioral semantics of monitored programs are detected by detection engine. On detection of any suspicious system call pattern, detection engine classifies a process as intrusive. It also notifies alert and log generator.

In cloud environment, tenant user behavior may get changed over a period of time, hence cloud administrator needs to verify the TVM from time to time for the legitimate programs. We assume that new normal behavior is differentiated from malicious one by cloud admin based on the expertise knowledge and information available about tenant activity logs. A classifier is retrained for the updated dataset over a time period.

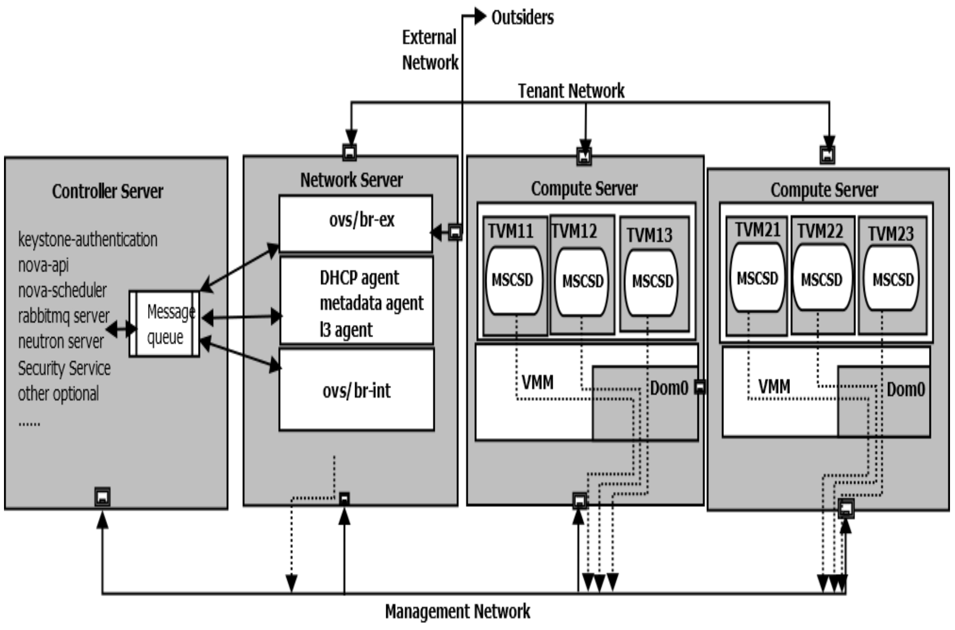


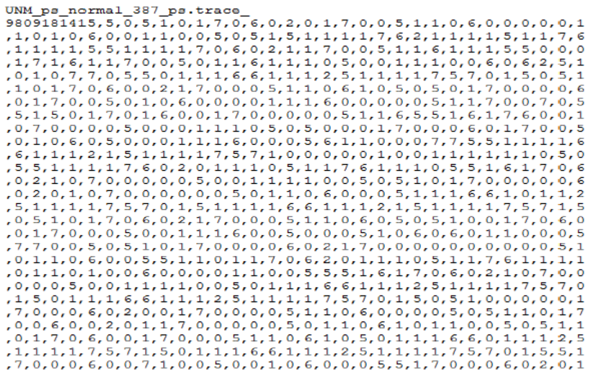
FIGURE 4.5: Deployment of of MSCSD instances in cloud environment

4.2.4 Alert and Log Generator

Alert and Log Generator (ALG) generates an alert signal if any of the test instance (monitored processes running in TVM) have been flagged as anomalous by Decision Model. It generates the log files with the information about the process name, process ID, system call logs, domain ID and domain name of the TVM generating the suspicious patterns. On receiving the alerts log report, cloud administrator can take further actions such as terminating the application generating malicious behavior, terminating the TVM from where the alert has been received or isolating the machine from the cloud environment. All the alerts are reported to the centralized cloud server i.e. cloud controller as shown in Figure 4.5 which is handled by cloud administrator.

4.3 Experiments

In this section, we first describe the dataset and its pre-processing in detail. Next, the analysis on results of classification is discussed. Lastly, we compare our work with existing related work. The prototype implementation of MSCSD has been done on a Linux machine with 16 GB RAM, 1 TB HDD, Core i7 processor and Ubuntu 15.10 Host OS, Core i7 processor, Xen 4.6 VMM and 1 Guest OS Ubuntu 14.04. Python 2.7.10



```
UNM_ps_normal_387_ps.trace_
9809181415,5,0,5,1,0,1,7,0,6,0,2,0,1,7,0,0,5,1,1,0,6,0,0,0,0,0,1
,1,0,1,0,6,0,0,1,1,0,0,5,0,5,1,5,1,1,1,1,7,6,2,1,1,1,1,5,1,1,7,6
,1,1,1,1,5,5,1,1,1,1,7,0,6,0,2,1,1,7,0,0,5,1,1,6,1,1,1,5,5,0,0,0
,1,7,1,6,1,1,7,0,0,5,0,1,1,6,1,1,1,0,5,0,0,1,1,1,0,0,6,0,6,2,5,1
,0,1,0,7,7,0,5,5,0,1,1,1,6,6,1,1,1,2,5,1,1,1,1,7,5,7,0,1,5,0,5,1
,1,0,1,7,0,6,0,0,2,1,7,0,0,0,5,1,1,0,6,1,0,5,0,5,0,1,7,0,0,0,0,6
,0,1,7,0,0,5,0,1,0,6,0,0,0,0,1,1,1,6,0,0,0,0,0,5,1,1,7,0,0,7,0,5
,5,1,5,0,1,7,0,1,6,0,0,1,7,0,0,0,0,0,5,1,1,6,5,5,1,6,1,7,6,0,0,1
,0,7,0,0,0,0,5,0,0,0,1,1,1,0,5,0,5,0,0,0,1,7,0,0,0,6,0,1,7,0,0,5
,0,1,0,6,0,5,0,0,0,1,1,1,6,0,0,0,5,6,1,1,0,0,0,7,7,5,5,1,1,1,1,6
,6,1,1,1,2,1,5,1,1,1,1,7,5,7,1,0,0,0,0,0,1,0,0,1,1,1,1,1,1,0,5,0
,5,5,1,1,1,1,7,6,0,2,0,1,1,1,0,5,1,1,7,6,1,1,1,0,5,5,1,6,1,7,0,6
,0,2,1,0,7,0,0,0,0,0,5,0,0,1,1,1,1,0,0,5,0,5,1,0,1,7,0,0,0,0,0,6
,0,2,0,1,0,7,0,0,0,0,0,0,5,0,1,1,0,6,0,0,0,5,1,1,1,6,6,1,0,1,1,2
,5,1,1,1,1,7,5,7,0,1,5,1,1,1,1,6,6,1,1,1,2,1,5,1,1,1,1,7,5,7,1,5
,0,5,1,0,1,7,0,6,0,2,1,7,0,0,0,5,1,1,0,6,0,5,0,5,1,0,0,1,7,0,6,0
,0,1,7,0,0,0,5,0,0,1,1,1,6,0,0,5,0,0,0,5,1,0,6,0,6,0,1,1,0,0,0,5
,7,7,0,0,5,0,5,1,0,1,7,0,0,0,0,6,0,2,1,7,0,0,0,0,0,0,0,0,0,0,5,1
,0,1,1,0,6,0,0,5,5,1,1,0,1,1,7,0,6,2,0,1,1,1,0,5,1,1,7,6,1,1,1,1
,0,1,1,0,1,0,0,6,0,0,0,0,1,1,0,0,5,5,5,1,6,1,7,0,6,0,2,1,0,7,0,0
,0,0,0,5,0,0,1,1,1,1,0,0,5,0,1,1,1,6,6,1,1,1,2,5,1,1,1,1,7,5,7,0
,1,5,0,1,1,1,6,6,1,1,1,2,5,1,1,1,1,7,5,7,0,1,5,0,5,1,0,0,0,0,0,1
,7,0,0,0,6,0,2,0,0,1,7,0,0,0,0,5,1,1,0,6,0,0,0,0,5,0,5,1,1,0,1,7
,0,0,6,0,0,2,0,1,1,7,0,0,0,0,0,5,0,1,1,0,6,1,0,1,1,0,0,5,0,5,1,1
,0,1,7,0,6,0,0,1,7,0,0,0,5,1,1,0,6,1,0,5,0,1,1,1,6,6,0,1,1,1,2,5
,1,1,1,1,7,5,7,1,5,0,1,1,1,6,6,1,1,1,2,5,1,1,1,1,7,5,7,0,1,5,5,1
,7,0,0,0,6,0,0,7,1,0,0,5,0,0,1,0,6,0,0,0,5,5,1,7,0,0,0,6,0,2,0,1
```

Figure 4.6: Feature set corresponding to UNM ps normal process

has been used for programming. There is no publicly available standard cloud dataset to validate the detection approaches proposed for cloud. Generating self generated cloud dataset requires huge resources for setting up real cloud environment and it also raises a question of data validation which has not been known to anyone. MSCSD is validated over the well known University of New Maxico (UNM) datasets [155]. This dataset is also being used by researchers [84][73] working in cloud security and is publicly available provided by University of New Mexico.

Table 4.2: Details of UNM Dataset

Process	Number of Intrusive traces	Number of Normal traces	Number of system calls in Normal Traces
CERT synthetic sendmail	23	294	182,901
UNM synthetic lpr	1001	4298	2,027,468
UNM Live named	2	27	9230, 572
UNM xlock (mixed)	2	72	16, 937, 816
UNM Live login	9	2	8, 894
UNM Live ps	26	24	6, 144
UNM Live inetd	31	3	541
UNM Live stide	105	13,726	15, 618, 237
UNM Synthetic sendmail	10	7	2076

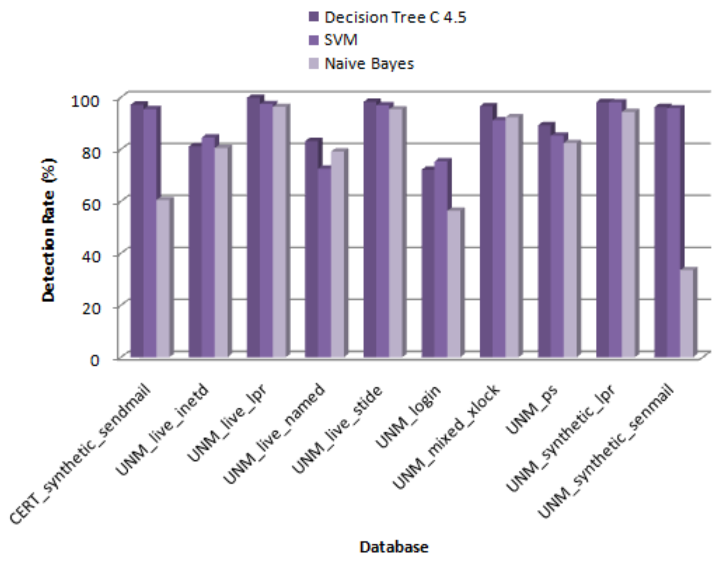


FIGURE 4.7: Performance comparison of different classifiers

4.3.1 Preparing the Dataset and Feature-Extraction

The detailed description of the UNM datasets is given in Table 4.2. Each UNM dataset represents the system call traces of privileged programs such as sendmail, ps, login etc. The normal and intrusive execution traces of each program is stored in separate dataset. Each dataset consists of trace files having two column entries. First column represents the PID of trace and second column represents the system call executed by the process. Data pre-processing is carried out to extract the features from each dataset by applying 'Bag of n-grams (BonG)' approach explained in section 4.2.2. The collected n-gram's frequency count corresponding to different traces of a dataset are stored in a common file named as FML. The sample feature vector of UNM ps normal process is shown in Figure 4.6 where first entry is PID name and rest entries are corresponding to frequency count of n_0, n_1 n_3....n_m n-grams respectively. The system call sequence corresponding to each n-gram, is stored in a mapping file where each n-gram entry in mapping file points to a short system call sequence.

4.3.2 Results and Discussion

In this section, we have evaluated the performance of the detection mechanism of MSCSD for detecting the malicious processes. The system call traces for each UNM dataset are pre-processed and are stored in form of bag of n-grams, as discussed above. The classification results of Decision Tree (DT) C 4.5 [159] have been discussed which is used as a learning/decision model for MSCSD. However, we have also considered the performance of other two classifiers mainly Support Vector Machine (SVM) [110] and Naive Bayes (NB) and compared their results with DT [160] over UNM dataset for detecting malicious programs. The comparison is depicted in the form of bar chart as shown in Figure 4.7 which shows the variations in detection rate of different classifiers for different datasets. The results of the Algorithm are shown in Table 4.3. The results show that Decision Tree C 4.5 is performing better than other algorithms in detecting most of the anomalous patterns. This is because of the implicit feature selection power of the tree

TABLE 4.3: Classification rate of different classifiers (Decision Tree C4.5, Neural Network and Naive Bayes)

Database	**DT C4.5**	**SVM**	**NB**
CERT syn. sendmail	97.171	95.421	60.4606
UNM live inetd	81.103	84.521	80.4342
UNM live lpr	99.812	97.365	96.321
UNM live named	83.121	72.521	79.10
UNM live stide	98.230	96.871	95.356
UNM login	72.103	75.325	56.321
UNM mixed xlock	96.542	91.143	92.2973
UNM ps	89.20	85.21	82.44
UNM synthetic lpr	98.11	98.07	94.341
UNM synthetic sendmail	96.213	95.7821	33.5042

TABLE 4.4: performance results of Decision tree C.5

Database	**Detection Rate (%)**	**FPR(%)**
CERT Syn sendmail	97.171	4.29
UNM live inetd	81.103	3.21
UNM live lpr	99.812	2.23
UNM live named	83.121	3.98
UNM live stide	98.230	3.01
UNM login	72.103	6.53
UNM mixed xlock	96.542	1.00
UNM ps	89.20	5.02
UNM synthetic lpr	98.11	1.34
UNM synthetic sendmail	96.213	1.93

based on Information Gain and Entropy. However, results are almost similar to SVM in some case such as UNM live lpr, UNM synthetic lpr and UNM synthetic sendmail. The average results are between 72%-99%. We found that there are variations in the total number of normal and intrusive traces in the datasets and hence there is a variation in the performance for different classifiers. We found that a machine learning algorithm performs better if the available dataset is sufficient and data distribution is balanced.

In addition, it is observed that SVM provides better results even for datasets with a small collection of traces. For example UNM live inetd consists of 31 intrusive traces and 3 normal traces. Total number of n-grams obtained are 235 which is lowest among all datasets. The detection rate obtained by SVM is 84.521%, DT C 4.5 is 81.103% and Naive Bayes is 80.4342%. For some datasets, large collection of n-grams are obtained such as UNM synthetic sendmail, there were 12351 unique n-grams found. Decision Tree is providing best results (97.171%) in this case. Time taken by SVM is longer than other classifiers for larger datasets. Naive bayes is fastest among all classifiers but the detection rate is not good for most of the cases. This is because of probability distribution done by Naive bayes is based on the assumption that features are independent to each other which may not be true for all cases.

It is also observed that if performance of a classifier is good in detecting one type of intrusion; it may not provide same accuracy in detecting other intrusions. For example, SVM is providing good results in detecting UNM_synthetic_lpr with detection rate 98.07% whereas detection rates of SVM for UNM ps and UNM login intrusions are not good (75%-85%). This is because of the patterns gathered during the normal execution of the processes are not sufficient enough to learn the behavior. The low frequency occurrence of these attacks may be similar to the normal execution of the system. There is also a difference in the behavioral characteristics of these attacks. Some attacks behave similar to the normal behavior of the system. For example, local privileges obtained without running some malicious script will provide behavior similar to the normal processes until the attacker misuses the system for launching further attacks. Weak authentication mechanism can be breached easily in one or two attempts and does not give any clue about suspicious system call execution clue. Detecting such kind of attacks

are very difficult to achieve. On an average, we found decision tree is performing better than other classifiers for detecting most of the malicious attack patterns.
It is observed that Decision Tree C.5 performs better in compare to other classifiers. It provides the detection rate of 72%-99.8% with false positive rate of 1.00%-6.53%. The false positive rates are high for login process. It may be because sometimes the wrong password attempts by legitimate users can be assumed as malicious attempts for breaking the password. On an average false positive alerts are less for UNM databases. The results of decision tree with false positive rate for all UNM processes are shown in Table 4.4.

MSCSD is compared with existing dynamic analysis based intrusion detection approaches in cloud environment as shown in Table 4.5. Gupta et. al [84] provided Immediate System Call Sequence (ISCS) approach where each feature is represented by a key-value pair. A key is a system call and value represents the list of immediate sequence of system calls following the key. However, they did not consider the length of trace in each 'key-value' pair (no sliding window concept). Hence, less efficient for infinite length traces or very long traces. Alarifi's et al. [67] applied 'Bag of system calls' method which keeps the frequency count of system calls and hence the ordering among system calls is lost. 'MSCSD' maintains the frequency count of each unique n-grams which represents the total occurrences of n-gram in each trace and hence temporal ordering of system calls is maintained within each n-gram sequences. A sliding window of fixed size is used in proposed MSCSD and in 'Bag of system calls' approach which generates the short attack patterns to observe. In addition, 'Bag of system calls' approach is less robust to attacks as an attacker may fool the detection mechanism by generating malicious system calls occurring with same frequency count as in normal system calls but in malicious order. ISCS approach may fail for longer and infinite length traces which make it less robust to detect attacks which delay the execution of programs and run in loops. However, in MSCSD, it is very difficult to generate short system call patterns with similar frequency count of normal patterns as knowing the different patterns of fixed size is quite difficult to achieve which makes 'MSCSD' more robust than other approaches. There are millions of system calls possible and storing frequency count of all possible system call traces or generating and storing immediate system call patterns is not an efficient

Table 4.5: Comparison between MSCSD technique and existing cloud-IDS techniques based on behavior analysis

Parameters	**MSCSD**	**ISCS [84]**	**BoS [67]**
Technique used	BonG	Key-value pair	Bags of system calls
Nature of Technique	Dynamic Analysis	Dynamic Analysis	Dynamic Analysis
Frequency count of n-grams	Considered	Not Considered	Not Considered
Ordering of System Calls	Considered	Considered	Not Considered
Sliding Window	Applicable	Not Applicable	Applicable
Robustness	High	Low	Low
Machine Learning	Applied	Not Applied	Not Applied
Testing Time	Low	High	Medium
Adaptability	Very High	Low	Low
False Positives	Low	High	High
Storage Requirement	Medium	Very High	High
IDS Placement	VMs	VMs	VMs
Scalable	Yes	No	No
Efficient	More than Gupta's Model	More than Alarifi's Model	Lesser among all
Data Set used for Validation	UNM	UNM	Self Generated
Accuracy	72.103%-99.812%	4.7%-100%	NA for UNM

approach. BoS and ISCS make a baseline dataset of all normal system call sequences which may not be a good approach in dynamic cloud environment, where the tenants behavior evolve over a period time. In such a case, generalizing the behavior will be most suitable approach. MSCSD incorporates the machine learning into the system unlike other approaches which brings the adaptability and scalability in the system. Moreover, ISCS and 'Bag of system calls' perform comparison of test instance with all possible observed normal behavior instances, which will lead to more testing time. The trained detection model of MSCSD, provides negligible testing time of a few seconds.

MSCSD and ISCS are validated with UNM dataset prepared in virtualization environment whereas 'Bag of system calls' is validated with self generated traces of programs in VMs. 'MSCSD' achieves an accuracy of 72.103%-99.812% whereas 'Key-value pair' achieves an accuracy of 4.7%-100% and 'Bag of system calls' achieves an accuracy of 100% for attack detection using self generated small dataset.

4.4 Conclusion

An intrusion detection approach, 'Malicious System Call Sequence Detection (MSCSD)' has been proposed for detecting intrusions on TVMs, running in cloud environment. MSCSD is a distributed in deployment as different MSCSD sub IDS instances are deployed in the monitored TVMs. Each of the instance is completely under the control of the cloud administrator. Tenants do not have access to any of the configuration file of MSCSD as each of them are write protected. MSCSD provides a feature representation approach called as BonG which is helpful in retaining the properties of enumeration-based and frequency-based approaches. It is validated with a well known UNM dataset of privileged programs and provides a good accuracy of 72.103%-99.812%. The approach is applicable to TVM-layer of cloud environment. The use of machine learning brings the adaptability in the system and is helpful to detect the variants of learned attack patterns which is way ahead of other approaches which solely depend on pattern matching approach. It is also secure from string manipulation attacks as it uses the numeric feature vector. It can be applied in cloud environment as a first-line of security defense mechanism.

Chapter 5

VM Introspection based Malware Detection (VIMD)

The chapter describes the design and implementation of VM Introspection based Malware Detection (VIMD), one of the sub IDS instance of CloudHedge. VIMD performs the program behavior monitoring at VMM-layer of cloud environment. The security design of VIMD is described with the explanation about the execution phases and detection components. The two core detection components (i.e. VMGuard and VAED) for behavior analysis, are described in detail in subsequent sections. VMGuard is based on system call sequence analysis whereas VMI-Assisted Evasion Detection (VAED) is based on system call transition analysis. VIMD can detect hidden malware, program subversion attacks and stealth evasion-based attacks.

5.1 Introduction

Signature based and static analysis approaches are also prone to anti-detection techniques such as obfuscation and encryption at Virtual Machine Monitor (VMM)-layer similar to Tenant Virtual Machine (TVM)-layer. Dynamic analysis is one of the popular approaches used to capture the run-time behavior of the programs. TVM-layer security solutions have better visibility to detect intrusions and they are also easy to be implemented. However, there are higher chances of subversion of security monitor as they are deployed on a monitored machine. A Cloud Service Provider (CSP) can apply our previous security solution, MSCSD (discussed in Chapter 4) at TVM-layer to provide the basic security from malware attacks. However, the attacks can be bypassed at TVM-layer if the security tool itself is compromised. Advanced malware can detect the presence of the hypervisor, virtualization environment or security analyzer. It can evade the TVM-layer security solutions as identified in the research gaps, discussed in the Chapter 2. There is a need to provide a more strong line of defense in cloud.

TVM-layer security solutions can not be applied directly at VMM-layer because of the semantic gap problem [92] as a VMM cannot directly access the high-level semantics of a guest machine. The drawbacks associated with existing VMM-layer security solutions have been discussed in Chapter 2. We propose a robust security architecture, called VM Introspection based Malware Detection (VIMD) as one of the sub-IDS instance of CloudHedge which provides second-line of defense from attacks at the VMM-layer. The aim of VIMD is to detect malicious activities happening on TVMs by integrating the run-time behavior analysis approach with VMI functions and machine learning techniques [161] at the VMM/hypervisor level. The security monitor in the VIMD runs in a privilege domain (i.e. Dom0) of the hypervisor. Dom0 is the administrative VM used by CSP to start/stop VMs, change the configuration of TVMs and enforce security policies on TVMs at IaaS-level. Each DomU is an untrusted TVM and does not have privilege to access administrative VM (Dom0). This makes the subversion of VIMD more difficult to achieve. Dom0 is considered to be in the Trusted Computing Base (TCB) of hypervisor. VIMD sub-IDS instances are distributed and deployed at each VMM in cloud which are centrally coordinated and configured by cloud administrator.

VIMD provides both primary and secondary security checks. The primary security check performs the process validation at the hypervisor to identify the hidden processes and presence of security-critical processes on TVM. The secondary security check extracts the execution traces of running processes to generate a detailed behavioral log of all the processes. It then performs the program semantic (run-time) behavior analysis at hypervisor to detect the advanced malware which attach themselves with the privileged programs and evasive malware which change their behavior often to evade the security tool. VIMD employs kernel debugging based VM introspection approach [124] to extract the execution traces of programs, running on monitored TVM from hypervisor. The introspection mechanism employed by VIMD uses the software breakpoint injection at guest OS kernel function. However, the introspection mechanism hides the breakpoints using Extended Page Table (EPT) feature which is explained in detail in Section 5.1.1. The combined use of EPT protection with breakpoints have been proved to be effective to hide from advanced anti-debugging techniques by Deng et al. [156]. VIMD utilizes the Rekall memory forensic framework [157] to obtain the details of guest OS kernel symbols and their address locations as these details are not provided by commercialized operating systems.

Once all the execution traces are collected, their behavior is analyzed at hypervisor using two proposed detection approaches. The first approach, VMGuard constructs the program semantics inform of the frequency distribution of the n-grams in a collection of traces. It applies Bag of n-grams (BonG) feature representation method and integrates it with text mining approach particularly Term Frequency-Inverse Document Frequency (TF-IDF). It leverages the VMI functions along with machine learning approach particularly Random Forest classifier to provide the practicability of BonG at the hypervisor also improve its accuracy. The second approach VAED constructs the program semantics in form of the System Call Dependency Graphs (SCDGs) to analyze the semantics in different execution paths of the programs installed on TVM. SCDGs are integrated with combination of feature selection (Information Gain Ratio) method, VMI functions and fusion of classifiers.

VMGuard considers the structural aspects of each trace. It also derives behavioral semantics by considering the frequency distribution of the system call sequences in intrusive and normal traces. VMGuard is found to perform well to detect attacks which do not depend on system artifacts such as analysis environment, hypervisor, etc. (eg. program subversion attacks). However, VMGuard does not capture the more complex behavioral aspects of programs as the the possibility of occurrence of system call transitions from each system call has not been considered. This is very important in capturing and detecting the complex behavior of evasive malware which may often change their behavior. VAED considers both structural and behavior aspects in different execution paths of the program in form of system call transitions. VAED is found to perform well to detect evasion-based attacks which change their behavior on detection of system artifacts. On detection of suspicious activity, alert signals are generated and sent to cloud administrator. VIMD can be opted by the CSP to provide the second-line of defense from advanced malware attacks.

The proposed VIMD exhibits the following key characteristics:

i. *Secure positioning*: It is difficult to subvert from a compromised TVM as user domains(DomUs) do not have access to Dom0. This makes VIMD more attack resistant from modern attacks which thwart security tool. It also addresses the problem of IDS subversion at the individual TVMs.

ii. *Robustness:* It performs both primary and secondary security checks making the architecture more robust to attacks which may bypass the basic security provided by the cloud administrator. The primary security check detects the hidden malware and verifies the presence of security-critical processes such as auto-update, autoscan in TVM. The Secondary security check performs the behavior analysis on processes to detect suspicious behavior using two proposed detection approaches.

iii. *Fine granularity:* Performing process-level detection at the TVM can help to reduce the effect of denial of service attacks caused by shutting down all processes running in a TVM on detection of any malicious activity. VIMD does the fine-granular monitoring of processes and blocks only those processes which perform maliciously.

iv. *High attack resistance:* It performs out-of-the-guest run-time program behavior analysis which makes VIMD more attack-resistant from anti-detection techniques such as obfuscation and encryption.

v. *Good accuracy:* VIMD is more accurate than other dynamic analysis approaches in cloud (such as ISCS [84], BoS [117]) as it applies text-mining approaches along with system call structure and frequency/probability of system call sequences/transitions.

vi. *Introspection support:* It incorporates VMI approaches at hypervisor for providing the high-level view of processes running in TVM which is way ahead of other IDS approaches which do process monitoring at TVM-layer.

vii. *Adaptability:* It uses a learning model to learn the certain patterns (features) present in process traces and hence, VIMD can easily be adapted to learn the behavior of new malware.

viii. *Detection of stealthy attacks:* It is installed at hypervisor and can directly access the TVM memory by using introspection libraries from outside. It does not perform the analysis inside the virtual machine where the malware is running. Hence, VIMD can detect stealthy attacks which hide their presence from security tools running on TVM.

ix. *Detection of program subversion:* It can also thwart malicious insiders who change the behavior of well-known programs running at a TVM which cause changes to the system call sequences of programs.

x. *Detection of evasion-based attacks:* It also facilitates the detection of evasive malware activities by providing a program semantic aware technique for extracting the behavior. Evasive malware try to change their behavior on detection of the security tool or delay their execution for thwarting the detection.

The discussion on detailed security design of various detection components of VIMD is given in subsequent sections.

5.2 VIMD: Security Design

In this section, first of all, the deployment of VIMD in cloud along with design choices is described. The description about various execution phases and detection components of VIMD is described in subsequent subsections. Detailed description about the core detection components i.e. VMGuard and VAED have been explained with their implementation details in Section 5.3 and 5.4 respectively.

The proposed security approach adds security functionalities at the virtualization-layer of Cloud Compute Server (CCoS). VIMD addresses the limitations identified in existing approaches, as mentioned in research gaps. It is superior to other security frameworks, as on one hand, it is difficult for an attacker to subvert the security monitor from TVM, while on the other hand, it provides efficient techniques for attacks detection at the VMM-layer.

The implementation set up for VIMD uses Xen VMM [10] for hosting the TVMs. Xen is an open source hypervisor supported by commercial CSPs such as Amazon [162] for hosting TVMs. We assume that CSP provides a trusted VMM platform. The privileged domain (Dom0)

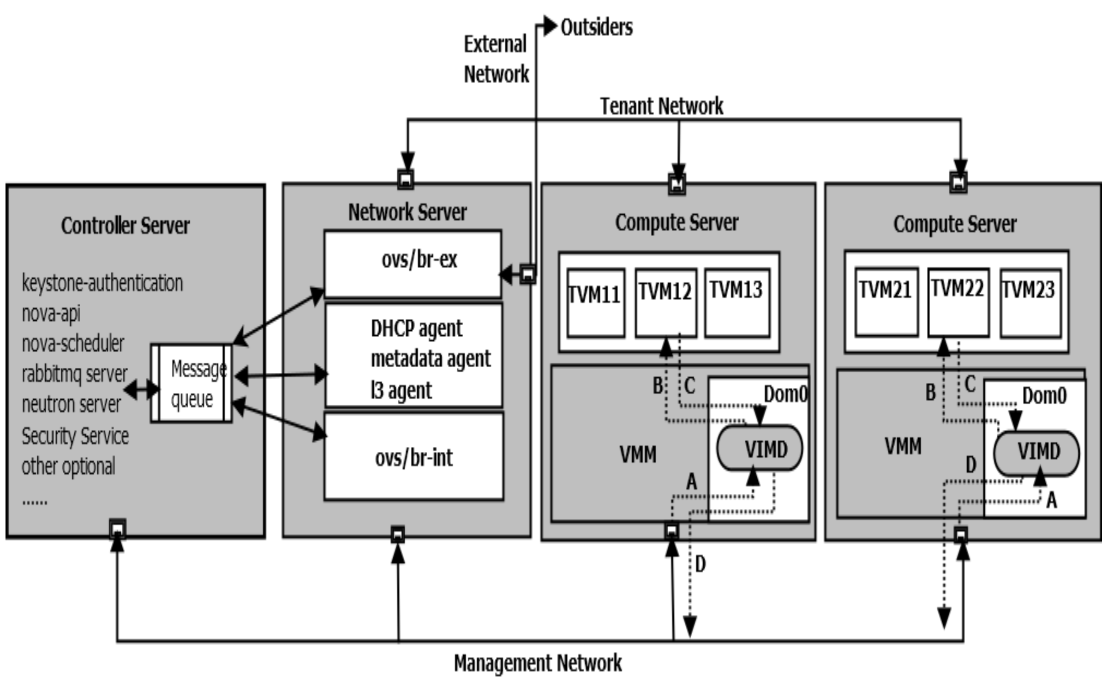


FIGURE 5.1: Secure deployment of VIMD instances in cloud environment

of VMM is used to monitor, control and configure the TVMs which are referred as DomUs (untrusted domains). Cloud administrator enforces strict policies at VMM to restrict DomU users from accessing Dom0. Hence, we propose the deployment of VIMD instances at the privileged domain (Dom0) of hypervisor on CCoS which performs the security check on TVMs from outside. It is under the control of cloud administrator and requires that cloud administrator has access to application specific information of monitored TVMs. Such an access is granted from VMM in CCoS. This privacy concern is clarified between CSP and tenant users at the time of registration in form of SLA. For example, Google says that it reserves the right to review the tenants applications and data [163] and users sign this agreement at the time of registration. A cloud administrator triggers VIMD from management network to perform the security check on a monitored TVM (line A). VIMD starts executing and performs memory introspection from Dom0 (line B). VIMD performs the analysis on process logs extracted from TVM memory(line C). If TVM is found to be suspicious, an alert signal is sent to cloud administrator with details of logs generated by VIMD (line D) as shown by dotted lines in Figure 5.1.

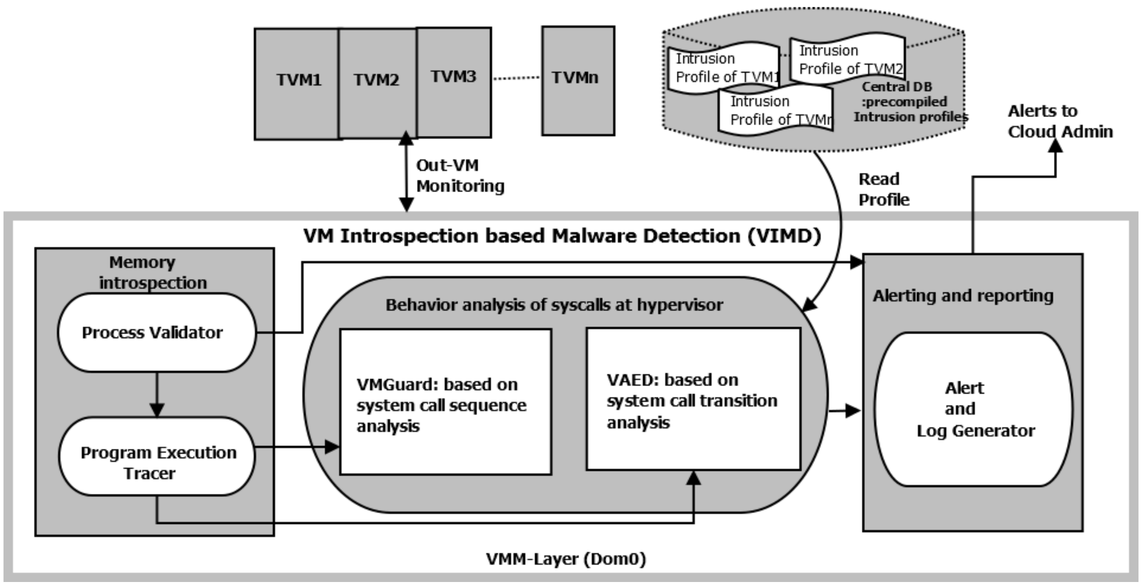
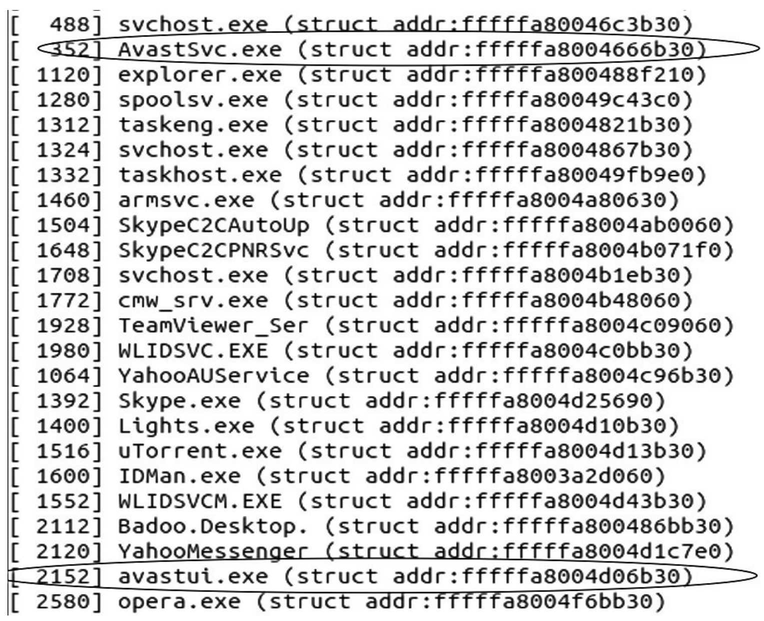

TVM1
TVM2
TVM3
TVMn
Intrusion Profile of TVM1
Intrusion Profile of TVM2
Intrusion Profile of TVMn
Central DB :precompiled Intrusion profiles
Alerts to Cloud Admin
Out-VM Monitoring
Read Profile
VM Introspection based Malware Detection (VIMD)
Memory introspection
Process Validator
Program Execution Tracer
Behavior analysis of syscalls at hypervisor
VMGuard: based on system call sequence analysis
VAED: based on system call transition analysis
Alerting and reporting
Alert and Log Generator
VMM-Layer (Dom0)

Figure 5.2: Security architecture for VIMD

Broadly, the execution of VIMD have been classified into three security phases (i) Memory introspection, (ii) Behavior analysis of syscalls at hypervisor and (iii) Alerting and reporting phase. At a high-level, there are total four detection components: Process Validator, Program Execution Tracer, VMGuard or VAED and Alert & Log Generator a shown in Figure 5.2. Each one of them is associated with one of the three key security phases for doing the attack detection at the hypervisor. In memory introspection phase, VIMD performs the basic security check to detect the presence of hidden processes or attacks which disable security processes (such as virus tools) running in TVM using Process Validator. VIMD then extracts the run time behavior of programs in terms of sequence of system calls using Program Execution Tracer. In behavior analysis phase, VIMD performs the behavior analysis using two detection components: VMGuard and VAED, as discussed briefly in Section 5.1. In alerting and reporting phase, an Alert and Log Generator is invoked to create logs and send alerts to the cloud administrator with detailed report. Below, we will discuss, each of the security phase and associated detection components in detail. VMGuard and VAED are discussed in detail in Section 5.3 and Section 5.4 respectively.

5.2.1 Memory Introspection

VIMD provides prime security functions in Memory Introspection (MI) phase to introspect the VM memory from Dom0 of VMM (a protected



```
[  488] svchost.exe (struct addr:fffffa80046c3b30)
[  352] AvastSvc.exe (struct addr:fffffa8004666b30)
[ 1120] explorer.exe (struct addr:fffffa800488f210)
[ 1280] spoolsv.exe (struct addr:fffffa80049c43c0)
[ 1312] taskeng.exe (struct addr:fffffa8004821b30)
[ 1324] svchost.exe (struct addr:fffffa8004867b30)
[ 1332] taskhost.exe (struct addr:fffffa80049fb9e0)
[ 1460] armsvc.exe (struct addr:fffffa8004a80630)
[ 1504] SkypeC2CAutoUp (struct addr:fffffa8004ab0060)
[ 1648] SkypeC2CPNRSvc (struct addr:fffffa8004b071f0)
[ 1708] svchost.exe (struct addr:fffffa8004b1eb30)
[ 1772] cmw_srv.exe (struct addr:fffffa8004b48060)
[ 1928] TeamViewer_Ser (struct addr:fffffa8004c09060)
[ 1980] WLIDSVC.EXE (struct addr:fffffa8004c0bb30)
[ 1064] YahooAUService (struct addr:fffffa8004c96b30)
[ 1392] Skype.exe (struct addr:fffffa8004d25690)
[ 1400] Lights.exe (struct addr:fffffa8004d10b30)
[ 1516] uTorrent.exe (struct addr:fffffa8004d13b30)
[ 1600] IDMan.exe (struct addr:fffffa8003a2d060)
[ 1552] WLIDSVCM.EXE (struct addr:fffffa8004d43b30)
[ 2112] Badoo.Desktop. (struct addr:fffffa800486bb30)
[ 2120] YahooMessenger (struct addr:fffffa8004d1c7e0)
[ 2152] avastui.exe (struct addr:fffffa8004d06b30)
[ 2580] opera.exe (struct addr:fffffa8004f6bb30)
```

FIGURE 5.3: A sample output of active processes running in a TVM, introspected by PV component from hypervisor

location) and do primary security analysis. VM introspection provides the high-level view of the system from privileged domain which is helpful to facilitate strong security measures. The two detection components; Process Validator and Program Execution Tracer are used to provide primary security check.

5.2.1.1 Process Validator

Some malware use rootkit technology to hide their presence from guest OS. Afterwards malware may disable the security-critical processes such as auto-update, auto-scan running in the TVM. Attacks such as conficker and torpig can disable the security tool itself [101]. Some other malware can maliciously modify the privileged programs and execute with the same name and privileges of victim program. These attacks are called as program-subversion attacks. In addition, advanced malware, called evasive malware (described in Chapter 2), can even try to change their behavior on detection of the virtualization and analysis environment. A malware can even attempt to evade the detection tool. VIMD aims to detect these attacks at the VMM-layer. VIMD invokes Process Validator (PV) to map the processes actually running in monitored VM from a trusted-view. It then performs process validation as a primary security check to provide basic security against attacks which target the virtual domains running in cloud. It has the following objectives:

∗ To detect whether any of the security-critical process such as auto-update, auto-scan etc, is disabled in the TVM.

∗ To detect whether there is any hidden process running in the TVM.

PV has two security modules: Virtual Machine Library Repository (VMLR) and Virtual Machine Memory Inspection (VMMI). The VMMI module of the PV detection component invokes the LibVMI API interface [122] to obtain a list of the processes (`Pro_List`) actually running in the TVM. LibVMI is an open source library for extracting the information about running processes such as process name, address location and PID from outside the VM at hypervisor. It is very useful in building the VMI based security systems. LibVMI provides the API for accessing the virtual memory and vCPU registers.

For Windows based TVM, first of all, kernel debug information is obtained using LibVMI win-guid tool. The tool takes the domain name of VM as input and creates the debug file (program database (PDB) file) of kernel (i.e `ntoskrnl.pdb`) with associated GUID information of PDB file which provides the mapping of kernel symbol names with their addresses. After that, Rekall [157] memory forensic framework is used to create the rekall profile of the VM (`win.rekall.json`) from the PDB file. The JSON format created by Rekall is easy to parse and is in machine readable format. To generate rekall profile, Rekall tool uses `fetch_pdb` and `parse_pdb` plugins to fetch and parse the debug information. It creates rekall profile based on GUID of debug file. VIMD configures LibVMI for `win.rekall.json` to get the information about the windows kernel functions and addresses. It does not require any low level details of memory like presence of certain signatures[70]. A memory snapshot of security-related processes for Avast is shown in Figure 5.3. This snapshot is taken from the hypervisor, providing the information of currently running processes at TVM.

For, Linux-based TVM, VIMD configures the LibVMI configuration file to fetch the kernel symbol details from `system.map` file which provides a lookup between symbol name and addresses in memory. The `system.map` is used to obtain symbol table. The page directory is mapped to find the correct page table using the symbol table. The page table is further mapped to find the data page in physical memory which belongs to a process. Finally a list of processes running in virtual memory of VM are returned to PV. This method is a standard way to fetch the details of the kernel data structure and is applicable to different version of OS.

The VMMI module of the PV detection component makes a connection with guest OS from VMM and invokes utilities within guest OS such as `ps` (for UNIX like VM) or `tasklist.exe / ps.exe` (for Windows VM) to generate the process list. It maintains `TVM_Report` for the monitored machine, taken from the untrusted-view (guest-view). A TVM_report represents a detailed list of the running processes, reported by a TVM. The information in `TVM_Report` is updated automatically at regular time intervals. A `TVM_Report` may not provide the list of processes which are controlled by an attacker and hence, it cannot be trusted. However, VMMI can access the actual VM-state information using the VMI functionality from the hypervisor and hence, the list of processes produced by it, can be trusted. The security operation of the PV component is now discussed. First of all, it executes the VMLR module to obtain the latest `TVM_Report` of the monitored TVMs. It then runs the VMMI module to get the `Pro_List` of monitored TVMs. PV analyzes both the results for the presence of any suspicious process: There are two cases:

∗ *Case 1:* PV scans `Pro_List` for the presence of security-critical processes. If any of the security-critical processes are not found in the `Pro_List`, then the TVM is considered to be compromised.

∗ *Case 2:* If security processes are found to be running in the TVM, `Pro_List` is compared with the `TVM_Report` for the presence of hidden processes. If there exist any variation in the list of processes between the `Pro_List` and the list of processes present in the `TVM_Report`, then the TVM is considered to be compromised.

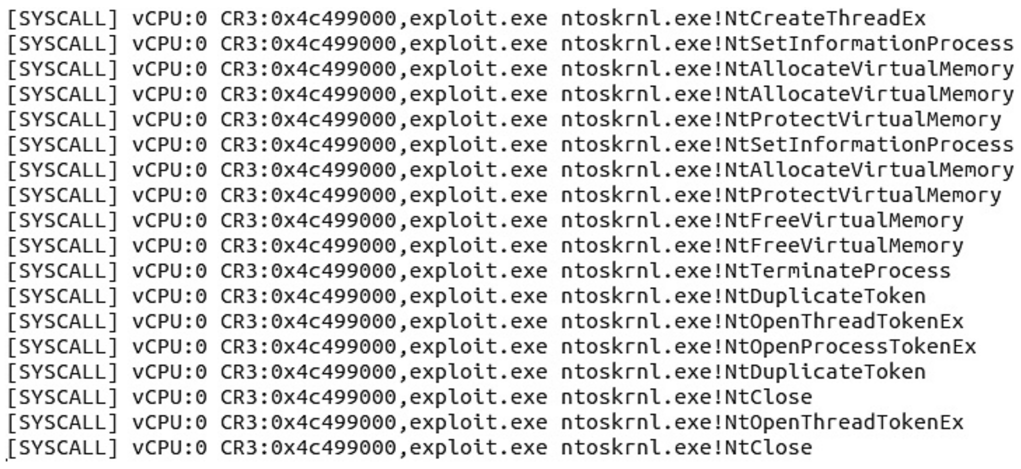
In both cases, it may happen that a TVM had been just booted or some malicious programs just terminated at the time when VMLR or VMMI were executed. In such cases, there would be variations in both the outputs. To reduce such false positives and false negatives, the VMLR queries a TVM multiple times to obtain a stable list of processes (`TVM_Report`) in successive runs. Similarly, the VMMI performs the memory mapping of a TVM in multiple runs.

If a TVM fails to pass the primary security check, it is assumed to be compromised and an alert is generated to the administrator with detailed reports of running processes that are found to be suspicious in the TVM. If TVM passes the primary security check, then VIMD performs the secondary security check. It extracts the execution traces of programs using advanced memory introspection and performs the behavior analysis of processes using two proposed detection mechanism explained in subsequent sections.

5.2.1.2 Program Execution Tracer

VIMD provides advanced memory introspection using Program Execution Tracer (PET) to extract run time behavior of processes executing in TVM. PET generates a behavior log which provides useful information related to running processes in the TVM which is further analyzed by other security components. There are some methods to trap the execution of processes such as generating a page fault exception (Nitro [135]) or generating a software debug exception using breakpoint injection (DRAKVUF [124]). The later method is used by VIMD for collecting the system call trace sequences at the Xen hypervisor. The former method can be applied for program execution tracing at KVM hypervisor. The breakpoint injection method is OS agnostic; it would work with any operating system running in the VM. Breakpoints are opcodes (eg. int 0XCC) inserted in the beginning of the kernel functions to be trapped. Once executed, it generates the software debug exception to vCPU. A vCPU is configured to call VM EXIT() operation in the exception handler.

A VM_EXIT transfers control to hypervisor which forwards the event to Dom0 where the PET detection component is running. It then performs the continuous read operation of the memory from the trapped location which generates the full trace execution of process. The other key aspect in PET is knowing the addresses of kernel functions, need to be trapped. This is done by using kernel-debugging tool called as Rekall [157]. Rekall supports the plugins for multiple operating systems. Rekall provides a standard way to retrieve the complete information about specific kernel version and is much flexible than traditional signature scanning mechanism [70]. Rekall parses the debug data (uses PDB file (`ntoskrnl.pdb`) for Windows kernel



```
[SYSCALL] vCPU:0 CR3:0x4c499000,exploit.exe ntoskrnl.exe!NtCreateThreadEx
[SYSCALL] vCPU:0 CR3:0x4c499000,exploit.exe ntoskrnl.exe!NtSetInformationProcess
[SYSCALL] vCPU:0 CR3:0x4c499000,exploit.exe ntoskrnl.exe!NtAllocateVirtualMemory
[SYSCALL] vCPU:0 CR3:0x4c499000,exploit.exe ntoskrnl.exe!NtAllocateVirtualMemory
[SYSCALL] vCPU:0 CR3:0x4c499000,exploit.exe ntoskrnl.exe!NtProtectVirtualMemory
[SYSCALL] vCPU:0 CR3:0x4c499000,exploit.exe ntoskrnl.exe!NtSetInformationProcess
[SYSCALL] vCPU:0 CR3:0x4c499000,exploit.exe ntoskrnl.exe!NtAllocateVirtualMemory
[SYSCALL] vCPU:0 CR3:0x4c499000,exploit.exe ntoskrnl.exe!NtProtectVirtualMemory
[SYSCALL] vCPU:0 CR3:0x4c499000,exploit.exe ntoskrnl.exe!NtFreeVirtualMemory
[SYSCALL] vCPU:0 CR3:0x4c499000,exploit.exe ntoskrnl.exe!NtFreeVirtualMemory
[SYSCALL] vCPU:0 CR3:0x4c499000,exploit.exe ntoskrnl.exe!NtTerminateProcess
[SYSCALL] vCPU:0 CR3:0x4c499000,exploit.exe ntoskrnl.exe!NtDuplicateToken
[SYSCALL] vCPU:0 CR3:0x4c499000,exploit.exe ntoskrnl.exe!NtOpenThreadTokenEx
[SYSCALL] vCPU:0 CR3:0x4c499000,exploit.exe ntoskrnl.exe!NtOpenProcessTokenEx
[SYSCALL] vCPU:0 CR3:0x4c499000,exploit.exe ntoskrnl.exe!NtDuplicateToken
[SYSCALL] vCPU:0 CR3:0x4c499000,exploit.exe ntoskrnl.exe!NtClose
[SYSCALL] vCPU:0 CR3:0x4c499000,exploit.exe ntoskrnl.exe!NtOpenThreadTokenEx
[SYSCALL] vCPU:0 CR3:0x4c499000,exploit.exe ntoskrnl.exe!NtClose
```

FIGURE 5.4: A sample output of PET component

and DWARF file with System.map file for Linux kernel) to establish a map of internal kernel functions. It generates the rekall profile for monitored machine (ex. `Windows-sp1.rekall.json` for windows kernel and `Linux.rekall.json` for linux kernel). Using these profiles, the introspection mechanism knows about the address details of the kernel functions to be trapped. VIMD executes PET to get the traces of benign and malware binaries. A sample execution trace output of a malware, extracted from hypervisor is shown in Figure 5.4. This shows the detailed information of a process including the sequence of system calls structure with process name and the address locations in CR3 register for each system call invocation.

One of the important aspect in execution tracing is the fear of detection of breakpoints at the guest VM by malware. The breakpoints can be easily detected by malware in the traditional security environment where the security code executes without the interruption or involvement of hypervisor. VIMD performs the analysis at the virtualization-layer of cloud where multiple VMs are controlled and monitored by hypervisor. The introspection mechanism used by VIMD makes use of altp2m feature of Xen hypervisor to hide the presence of breakpoints from guest VM applications. Xens altp2m is an extension of p2m, called as alternate p2m. The p2m feature refers to the translation of the guest physical memory address (GPA) to host machine physical memory address (MPA). The p2m functionality in Xen is used to partition the real memory of the machine between Xen and the VMs over it. This also ensures that one VM cannot access the memory of the other

VM without permission and is implemented by the Extended Page Table (EPT) hardware-support mechanism. This mechanism in hardware terminology is called as hardware-assisted paging mechanism in which a single EPT is maintained by hypervisor for each of the guest VM. The altp2m is an advanced feature which allows Xen to create multiple EPTs (multiple views) of a guest VM [164].

The EPT-layer is very helpful for the stealthy monitoring from the hypervisor. This advanced feature enables the introspection mechanism to create a shadow copy for each memory page where the breakpoint is going to be written. The breakpoints are written only in the shadow copy of page. Guest VM accesses the guest physical memory in a normal way. It is actually executing the shadow copy which is a trapped memory page. During execution, whenever a breakpoint is hit; the modified guest view is switched to the unaltered guest view for a single instruction (read access) and after that it is again switched back to shadow copy view. This procedure hides the presence of the traps from the guest VMs. Thus two guest views are created for monitored guest using the altp2m feature. Each of the page in altered view (trapped view) is marked as execute-only page which prevents the read access on the execute-only page. A program trying to scan the memory will induce an EPT violation which is handled by switching the memory views. The read access is performed in the unaltered view only. The functionality is integrated with the introspection mechanism employed by VIMD. It now enters in the behavior analysis phase and calls the other detection components to perform the detailed investigation of processes.

5.2.2 Behavior Analysis of Syscalls at the Hypervisor

One of the core detection phase of VIMD is performing the behavior analysis of syscalls at the hypervisor and providing the secondary defense from advanced attacks. The system calls extracted from earlier phase are analyzed in detail for fetching the behavior semantics of programs. There exist some methods that can be used for analysis such as look-ahead pair based approach [85], STIDE approach [115] and frequency-based approach [128]. Look-ahead pair approach considers the ordering of system calls. In this approach, each feature is represented by two values $< look, ahead >$. A ‘Look’ represents a system call and ‘ahead’ represents the immediate sequence of system calls. The approach maintains a structural property of system call sequences. However, it fails for infinite length traces. Moreover such approaches are subjected to more false alarms, because of rigid sequence matching technique [115]. An attacker can insert many legitimate sequences in the program trace to thwart detection. STIDE approach maintains the short sequences of system call patterns of fixed size and is also based on the pattern matching approach. A rare pattern may occur because of sudden misuse behavior of a normal user such as invoking the unwanted function, overflowing the buffer or any unexpected error. The anomalous behavior of a trace cannot ascertain just by mismatches with the baseline database.

The frequency based model considers the count of each unique system call as feature vector. Although, storing the frequency count of each system call reduces the storage requirement. It loses the ordering of system calls which is the major limitation in such a system. Attacker can fool the system by creating similar frequency (as of normal) by injecting legitimate code into the malicious program without changing malicious pattern. Some researchers [67] [148] have applied the frequency method and some researchers [84] [73] have applied the look-ahead method for attack detection application at the TVM-layer in the cloud. These methods are less efficient for attack detection because of the above mentioned issues.

The behavior semantics are very important to detect the attacks which can even attach themselves with the benign processes and run with benign process privileges. There are two core detection components of VIMD which extract the behavior semantics and carry out the analysis on the execution traces, collected from the memory introspection phase, called VMGuard and VAED. VMGuard is based on improved Bag of n-Gram (BonG) analysis whereas VAED is based on System Call Dependency Graph (SCDG) analysis. Each of them analyzes the traces individually using different detection mechanisms for the behavior analysis of processes.

A. VMGuard: VMI-Assisted Malware Detection Based on System Call Sequence Analysis

VMguard is proposed to detect malware attacks at the VMM-layer by capturing the run-time behavior of programs outside the TVM. It analyzes behavior of programs by doing system call sequence analysis. VMGuard provides the integration of BonG (feature representation) method which is based on frequency based n-gram model with text mining approaches for feature selection particularly TF-IDF and ensemble learning approach. VMGuard runs outside the TVM at the Dom0 (privilege domain) of hypervisor and is totally configured, monitored and controlled by cloud administrator. It takes the input from the PET; detection component which runs in memory introspection phase of VIMD. The behavior is analyzed by two sub-detection components of VMGuard: (i) Trace Pre-Processor (TPP) and (ii) mX_DetectionEngine.

TPP pre-processes the collected information to extract useful patterns with high discriminative power. TPP applies BonG with combination of TF-IDF to select the most appropriate n-grams. BonG considers the frequency distribution of all possible unique n-gram sequences in both normal and intrusive traces. It forms the numeric feature vector which takes the count of patterns (n-grams) into consideration and is not subjected to string-manipulation attacks. The integration of BonG with TF-IDF method improves the discriminating power of n-grams. It gives importance to both factors i.e. how frequent a sequence is occurring in a trace? (TF) and How rare a sequence is in a collection of traces? (IDF). Therefore, VMGuard removes less discriminative sequences. The output of the TPP is fed to the mX_DetectionEngine. The mX_DetectionEngine uses the pre-compiled intrusion profile of the TVMs (discussed in Section 5.2.4) and checks the behavior similarity of the currently running processes with the intrusive behavior. The detection engine of VMGuard applies Random Forest classifier to learn and detect the program behavior instead of using a single decision tree like MSCSD. The ensemble classification results of VMGuard are better then single classifier results of MSCSD for detecting the program-modification attacks. The subverted programs change the normal execution sequence of the victim programs. VMGuard is has been validated with University of New Maxico dataset (UNM) [155] of program subversion attacks and found to perform well to detect such attacks. It can be used as a secondary check to detect attacks which have bypassed MSCSD. The detailed description of VMGuard with its implementation is given in Section 5.3.

VMGuard is mainly concerned with the structural aspects of traces. Though, it considers the frequency distribution of sequences in malware and normal classes. It does not consider the possible behavior of execution traces in terms of other possible execution paths of patterns from each system call. Hence, it may fail to detect the complex evasive malware behavior which may change their behavior pattern after some time of execution. A different detection mechanism, called VAED is proposed for detecting the evasive malware which considers the possible system call execution paths for each trace. The detection approach of VAED is based on the probability model of system call transitions. It considers the frequency distribution of each system call transitions (say S1 to S2) with respect to all other transitions from calling site (S1), discussed below.

B. VAED: VMI-Assisted Evasive Malware Detection (VAED): is proposed to detect evasion-based attacks at the VMM-layer by capturing the run-time behavior of programs outside the TVM. VAED provides system call transition analysis to extract the behavioral-semantics of the evasive malware programs. It is difficult to generate signatures (based on system call patterns, opcode sequences or byte code patterns etc.) for evasive malware samples as attacker carefully crafts such malware samples to evade detection. Hence, behavioral analysis is useful, which extracts the run-time behavior of malware patterns. VAED represents the behavior of each process in term of System Call Dependency Graph (SCDG), used to generate the features. Graph analysis approaches overcome the drawback of sequence analysis approaches and capture both structural and behavioral aspects of the programs. This is very important for capturing complex behavior of evasive malware which may often change their behavior. In past few years, system call graph analysis has been applied in different fields like network security, system security and document processing applications. Anderson et al.[165] demonstrated the use Markov Chain property for modeling the behavior of assembly code instruction traces in-form of graphs for detecting attacks in traditional systems. They have also shown the effectiveness of probability based graph model. The system call graph model has not been applied for malware attack detection in cloud. Taking the motivation from this, we applied it for modeling the behavior of evasive malware execution traces in form of SCDG to detect attacks at VMM-layer in cloud environment. It is also configured, monitored and controlled by cloud administrator.

VAED takes the input from the PET; detection component running in memory introspection phase of VIMD. The behavior is analyzed by following sub-detection components: (i) Program Semantics Extractor (PSE) (ii) Feature Transition Matrix Generator (FTMG) (iii) eX_DetectionEngine. PSE generates the system call graphs (inform of adjacency matrix) for each system call trace using Markov Chain property. Each SCDG is stored in a adjacency matrix and pre-processed by FTMG to obtain a feature vector which is appended in a common file. FTMG applies the Information Gain Ratio (IGR) to extract the system call graph paths with maximum information. The output is fed to the eX_DetectionEngine. The eX_DetectionEngine uses the pre-compiled intrusion profile of the TVMs (discussed in Section 5.2.4) and checks the behavior similarity of the currently running processes with the intrusive behavior. The detection engine of VAED applies the fusion of diverse multi classifier scheme to obtain the detection output which is found to perform well when compared to ensemble classifiers. VAED has been validated with the evasive malware dataset [158]. The evasive samples tend to change their behavior. The behavior is captured in multiple runs of the programs from hypervisor. The system call transition analysis done by VAED, is found to perform well for detecting evasive malware running in TVM. The detailed description of VAED with its implementation is given in Section 5.4.

Detection of how the malicious user or system programs or malware by core detection components of VIMD is discussed nest. Suppose that an attacker succeeds in attaching a malware code with a victim program. The malware sample will execute some sequence of system calls which was earlier captured by detection engine during the training phase and hence, would be detected at the time of its execution. For example, the PET component produced a subsequence of a system call trace for a malicious program: NtRaiseException() $\rightarrow$ NtInformationProcess()$\rightarrow$ NtInformationThread() $\rightarrow$ NtRaiseException() $\rightarrow$ NtInformationProcess()$\rightarrow$ NtInformationThread() $\rightarrow$ NtRaiseException()... occurring in repetition and forming an infinite loop. These sub-sequences are more likely to occur in malicious software and are captured by the DE component. The intuition is that processes (whether user-level or system-level processes) can use an abnormal sequence of system calls to gain memory accesses and perform malicious operations to launch an attack is considered. Hence, the technique is able to detect malicious patterns irrespective of whether they emerge in user or system processes.

However, the system call patterns for newly installed applications at a TVM will not match with any malware behavior as they do not perform any malicious action. A malicious action can cause violation in memory access or raise exceptions in a loop, running infinitely many times and crashing an application. There are fewer chances that a new application might display system call sequences not following a pattern consistent with malignant processes.

The detection engine (mX_DetectionEngine / eX_DetectionEngine) generates a notification to Alert and Log Generator upon detection of a suspicious processes which sends alerts to cloud administrator.

5.2.3 Alerting and Reporting

In alerting and reporting phase, an Alert and Log Generator (ALG) component is invoked. Analyzer checks the details of the applications generating the malicious behavior with the help of VMLR. The applications which generate such malicious traces are dynamically isolated. ALG can identify the malicious processes with the help of behavior logs generated by the PET. As the detection approach of VIMD is based on runtime behavior analysis of malware, it enables capturing the actual behavior of malware, which is learned using the machine learning classifiers. The use of machine learning permits detection of the variants of the learned attacks and their classification to a suitable attack class based on the similarity score with the learned attack classes. The malware variants differ in signatures but possess similar behavior characteristics to their original malware class. Signature matching techniques fail to detect such attacks as they are not based on behavior analysis and generate false negative alerts for known malware variants. If a TVM generates a large number of malicious processes, such a TVM is isolated and restored to a previous checkpoint that is benign. The alert generator component sends alerts to the cloud administrator with

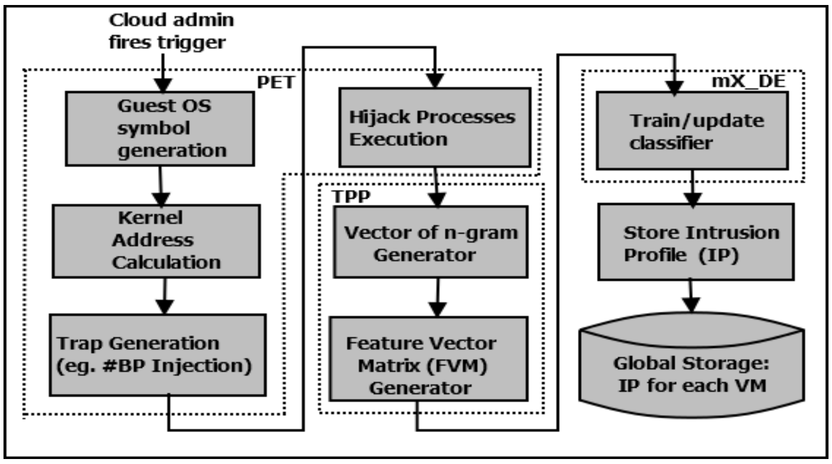

Cloud admin fires trigger
PET
Guest OS symbol generation
Kernel Address Calculation
Trap Generation (eg. #BP Injection)
Hijack Processes Execution
TPP
Vector of n-gram Generator
Feature Vector Matrix (FVM) Generator
mX_DE
Train/update classifier
Store Intrusion Profile (IP)
Global Storage: IP for each VM

FIGURE 5.5: Intrusion Profile generation using VMGuard

a detailed report of malware detected at the TVM and basic default security actions is taken. The cloud administrator can then take further actions to improve the security of the system.

5.2.4 Generation of Intrusion Profile

Intrusion profiles are created by cloud administrator in an offline mode for the monitored virtual machines. The sequence of system calls invoked by a TVM may differ depending on the applications and the guest OS kernel running in the machine. Therefore, different behavioral profiles need to be captured for each TVM in VIMD architecture. VIMD makes use of the existing detection components of for each approach in an offline mode for creating an intrusion profile database for each monitored TVM as shown in Figure 5.5 (in case of VMGuard) and Figure 5.6 (in case of VAED). The process of intrusion profile creation has three main stages namely Trace Execution Phase (TEP), Feature Generation and Pre-Processing phase (FGP) & Classification phase. They are described in detail as follows:

(i) Trace Execution Phase: In this phase, traces of benign and malware files are obtained using the PET detection component. First, a clone TVM of the monitored TVM is created using Logical Volume Manager (LVM)'s copy-on-write (COW) functionality. The original TVM state is saved and a backend/snapshot of the original disk volume is created. The malware files are then injected into the Clone TVM of the monitored TVM from the privilege domain of the hypervisor using libguestfs tool [166] copying files into and out of a virtual machine. We

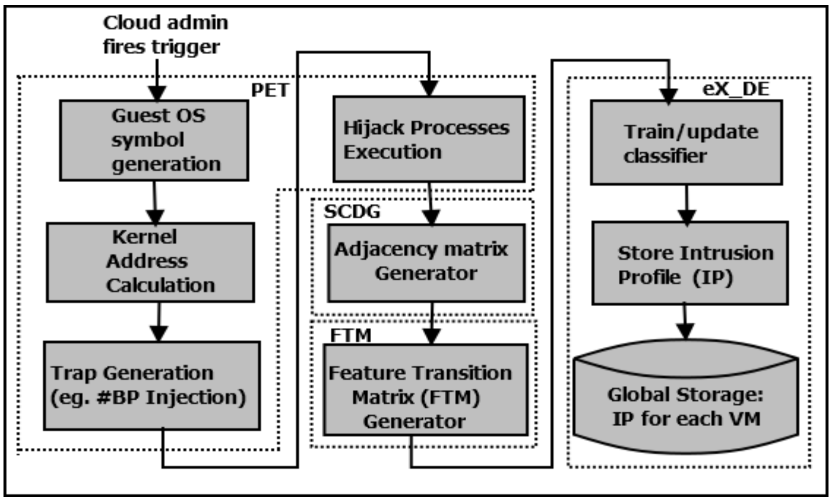

Cloud admin fires trigger
PET
Guest OS symbol generation
Kernel Address Calculation
Trap Generation (eg. #BP Injection)
Hijack Processes Execution
SCDG
Adjacency matrix Generator
FTM
Feature Transition Matrix (FTM) Generator
eX_DE
Train/update classifier
Store Intrusion Profile (IP)
Global Storage: IP for each VM

FIGURE 5.6: Intrusion Profile generation using VAED

have applied this technique in an open-source Xen hypervisor deployment scenario. However, there are other easier methods for sharing files between host and VM machines such as the use of vmtool in VMware ESX-based deployment, but it can lead to risks of information disclosure. The program traces of benign and malicious programs running in a cloned TVM are collected from the hypervisor using the PET tool. The clone TVM is deleted and the original TVM resumes its execution after trace collection.

(ii) Feature Generation and Pre-Processing: In this phase, the existing detection components of behavior analysis phase are used to retrieve the features from the labeled malware samples. VMGuard and VAED, both approaches are different for trace pre-processing. In VMGuard, the first step in pre-processing is retrieving the n-gram for each trace, and the second step is pre-processing the collected n-grams to generate a numeric vector of n-gram values, which represents the frequency count of each of the n-grams appearing in a trace. Next, TF-IDF is applied to select the most appropriate features. The outputs of pre-processing are maintained in the Feature Vector Matrix (FVM).

In case of VAED, the first step is pre-processing each trace to generate system call dependency graph of system calls with labeled transition probabilities for each edge. Each SCDG is stored as an adjacency matrix and pre-processed to obtain the feature vectors. A feature vector represents the transition probabilities of each system call transition appearing in a trace. Next, all feature vectors are combined to form common file. VAED applies IGR to select the most appropriate features.

The output of pre-processing is maintained in the Feature transition Matrix (FTM). Each entry in FVM/FTM is labeled with the intrusion class. Each of the intrusive and benign traces may have some set of common features but the value corresponding to the feature may be different. For example, a malicious file may invoke a large number of open() and read() operations in an abnormal way, which could be an attempt to find some secret value by searching a large number of files. In addition, malicious traces may also have some sequence of system calls of kernel routines which do not appear in any of the benign traces. This could be due to invocation of rare system calls. It is important to consider such behavior of sequences as it plays an important role in distinguishing the behavior of benign and malicious processes. We have used text-mining approaches, described in Section 5.3 and 5.4 to capture important behavior semantics of programs.

(iii) Classification: In this phase, the detection engine is trained for the FVM/FTM values. In VMGuard, we have used a training module of the mX_DetectionEngine, which uses Random Forest (RF) algorithm to learn the behavioural statistics of the malicious and benign programs stored in the FVM. RF provides good accuracy as it involves data distribution, feature selection and parameter tuning of the algorithm. The trained classifier represents the pre-compiled intrusion profile of a monitored TVM, and is stored in the database alongside the VMM.

In VAED, we have used training module of eX_Detection Engine which uses fusion based ensemble classifier to learn the features of FTM. It fuses the results of multi-classifiers to get a common output and obtain a good accuracy to detect detect evasive attacks. The pre-compiled intrusion profiles (PIPs) are stored in common global database storage, alongside the VMM.

5.3 VMGuard: Design and Implementation

VMGuard analyzes the execution traces, obtained from the memory introspection phase of VIMD using two sub-detection components: (i) Trace Pre-Processor (TPP) and (ii) mX_DetectionEngine. TPP pre-processes the collected information to extract useful patterns by extracting the behavior of various system call sequences and

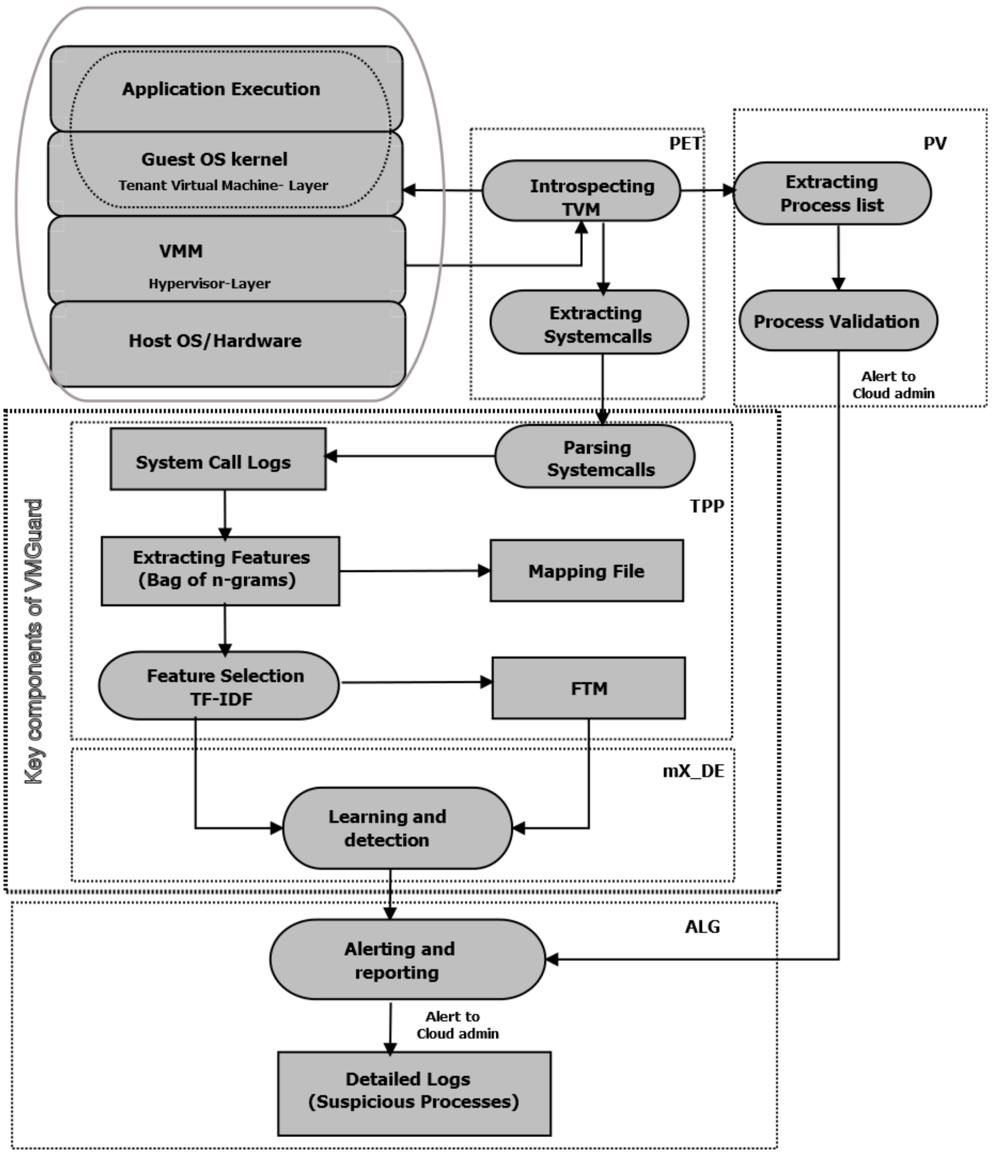


FIGURE 5.7: Execution Flow of various security functions of VIMD with behavior analysis using VMGuard

mX_DetectionEngine applies machine learning to learn and detect the malware behavior. The execution flow of various security functions using VMGuard is given in Figure 5.7. A detailed description of its detection phases and the implementation of VIMD with VMGuard as core detection component is presented in subsequent sections.

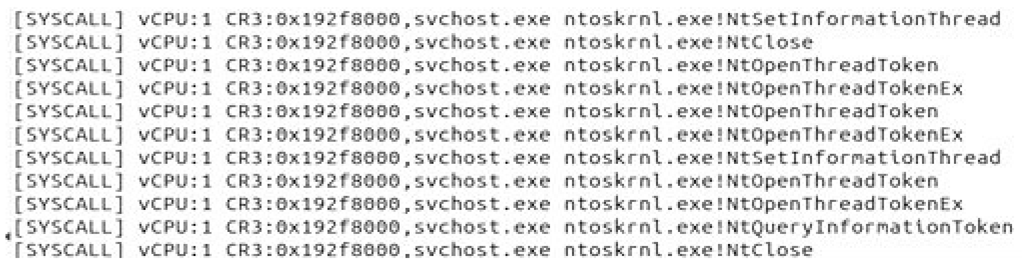
5.3.1 Program-semantic extraction using frequency based n-gram approach integrated with TF-IDF

Initially, all the collected traces, obtained from the memory introspection phase are pre-processed to extract some semantics out of them. This stage uses a detection sub component Trace-pre-processor (TPP) for this purpose. TPP pre-processes the collected traces, extracts the desired features from the traces and forwards them to the detection engine for further analysis. Similar to mining approaches, VMGuard employs a feature extraction method, called 'Bag of n-grams (BonG)' integrated with Term Frequency-Inverse Document Frequency (TF-IDF) for intrusion detection in the cloud. The detailed description of feature extraction using BonG approach has been given in Chapter 4 with example. A brief summary is provided here. However, the details about how BonG has been improved to provide a more discriminative system call patterns from the set of normal and intrusive traces, is given in detail. The steps of the BonG method are briefly explained below:

1. Process each trace i (present in Log_DB) into a feature vector, represented by $X_i = n_1, n_2, n_3, n_4, ...n_m$, as a set of n-grams, where m is the total number of n-grams obtained after processing the trace. Each n-gram is represented by a feature vector $n_j =< s_1s_2s_3s_4s_5s_6 >$ where $1 <= j <= m$, a substring of a system call.

2. Each feature vector is converted into a numeric vector represented by $X_i' =< c_1, c_2, c_3, c_4, c_k >$ where each c value represents the total number of occurrences of each unique n-gram value in the trace and k is the total number of unique n-grams obtained.

A collection of n-gram feature vectors is obtained for the collected traces by following step 1 of the procedure as explained above, and are stored in a Log file (Log2_DB). The numeric vectors of n-grams obtained after step 2 are written in a common feature-vector database file, FV_DB. It represents the frequency count of unique n-grams in each trace. For example, the execution trace of a system process (srvhost.exe), intercepted from hypevisor, is shown in Figure 5.8. The sample of collected



```
[SYSCALL] vCPU:1 CR3:0x192f8000,svchost.exe ntoskrnl.exe!NtSetInformationThread
[SYSCALL] vCPU:1 CR3:0x192f8000,svchost.exe ntoskrnl.exe!NtClose
[SYSCALL] vCPU:1 CR3:0x192f8000,svchost.exe ntoskrnl.exe!NtOpenThreadToken
[SYSCALL] vCPU:1 CR3:0x192f8000,svchost.exe ntoskrnl.exe!NtOpenThreadTokenEx
[SYSCALL] vCPU:1 CR3:0x192f8000,svchost.exe ntoskrnl.exe!NtOpenThreadToken
[SYSCALL] vCPU:1 CR3:0x192f8000,svchost.exe ntoskrnl.exe!NtOpenThreadTokenEx
[SYSCALL] vCPU:1 CR3:0x192f8000,svchost.exe ntoskrnl.exe!NtSetInformationThread
[SYSCALL] vCPU:1 CR3:0x192f8000,svchost.exe ntoskrnl.exe!NtOpenThreadToken
[SYSCALL] vCPU:1 CR3:0x192f8000,svchost.exe ntoskrnl.exe!NtOpenThreadTokenEx
[SYSCALL] vCPU:1 CR3:0x192f8000,svchost.exe ntoskrnl.exe!NtQueryInformationToken
[SYSCALL] vCPU:1 CR3:0x192f8000,svchost.exe ntoskrnl.exe!NtClose
```

FIGURE 5.8: Trace interception of svchost.exe process at VMM by PET component

TABLE 5.1: Sample of n-grams extracted from a system process

	srvhost.exe
n1	NtSetInformationThread(), Ntclose(), NtOpenThreadToken(), NtOpenThreadTokenEx(), NtOpenThreadToken(), NtOpenThreadTokenEx()
n2	Ntclose(), NtOpenThreadToken(), NtOpenThreadTokenEx(), NtOpenThreadToken() , NtOpenThreadTokenEx(), NtSetInformationThread()
n3	NtOpenThreadToken(), NtOpenThreadTokenEx(), NtOpenThreadToken(),NtOpenThreadTokenEx(), NtSetInformationThread(), NtOpenThreadToken()
n4	NtOpenThreadTokenEx(), NtOpenThreadToken(), NtOpenThreadTokenEx(), NtSetInformationThread(), NtOpenThreadToken(), NtOpenThreadTokenEx()

n-grams for the srvhost.exe trace is shown in Table 5.1. The feature vector obtained for srvhost.exe is $< normal, 2, 3, 5, 1, ... >$.

The key component in trace pre-processing is extracting the n-gram values. Here an n-gram is a short sequence of system calls obtained by considering a sliding window of size k and gradually shifting the window by 1. Each shift of the window provides a unique n-gram sequence. Here we have considered size=6 that has proved to be the best size based on analysis carried out with various lengths of n-gram sequences by Warrender et al. [116].

Applying Feature Selection: The collected n-grams are not sufficient enough to represent the most discriminative patterns of short sequences. There exist some n-grams that appear in both types of attack classes. Infact, some n-gram may appear only in intrusive traces with very low frequency which could be because of the general system faults. Hence, we improved BonG by integrating it with TF-IDF, a popular text mining approach, used to selected suitable features (n-grams). The goal of feature selection is to select the most important and optimal subset of features (n-grams) in order to improve the classifier's performance. Feature selection improves the generalization performance and reduces the computational cost of a classifier. It makes the classifier faster in detecting unseen data and simplifies the understanding of data processing. Information Gain (IG) and Document Frequency (DF) have been used in the traditional IDS systems [167]. However, they do not reflect the rarity principle which says that rare patterns of system calls are good for discrimination among large sets of data.

Term Frequency-Inverse Document Frequency (TF-IDF): We applied a TF-IDF measure for feature selection in anomaly detection techniques based on n-gram analysis. TF-IDF is a numerical statistic that is intended to reflect how important a word is in a collection or corpus. This measure is very popular in text mining techniques used in information retrieval. Here, TF-IDF will reflect how important an n-gram sequence is in a collection of n-grams (data set). Repeated n-grams in many traces will not have a good discriminating power, hence selecting rare n-grams out of a given class will be helpful in discriminating the classes. A higher TF-IDF of an n-gram implies a strong relationship of the n-gram with the corresponding class. This is calculated with the help of two measures, Term Frequency (TF) and Inverse Document Frequency (IDF).

Term Frequency: The mismatch of certain sequences with the baseline database as considered in the existing anomaly detection approaches is not sufficient to detect malicious processes. Some rare sequences may appear in a running process because of some general error condition such as a general page fault. Hence, the mismatch alone is not sufficient to detect malicious traces. The count of sequences called Term frequency is also important to consider.

Definition 1: The Term frequency of an n-gram sequence refers to the total number of times the n-gram occurs in a trace (benign or malicious). The Term frequency of an n-gram in trace X within a class i can be represented by

$$TF(ab, X_i) = f(V_{ab}, X_i) \tag{5.1}$$

where $f(V_{ab}, X_i)$ is the frequency of sequence 'ab' in the system call stream X with class i (benign or malicious). Each X_i represents a trace. Hence, this measure will provide the frequency count of 'ab' subsequences in X_i. V_{ab} represents the value of subsequence 'ab' (total number of n-gram sequences with 'ab' term).

Inverse Document Frequency: This measure improves the discriminating power of a term (n-gram). It overcomes the drawback of the document frequency approach and emphasizes the importance of rare system call sequences, as they have more discriminating power, which is important in anomaly detection techniques. Rare system call sequences usually represent abnormal usages of the system and hence, useful for anomaly detection.
Definition 2: IDF is defined as the logarithmic ratio of the number of traces in a collection to the number of traces having the given sequence. The Inverse document frequency of an n-gram sequence can be defined by

$$IDF(ab, C_i) = log(C_i/C_{V_{ab}}) \tag{5.2}$$

where C_i represents the total traces (data set) with the target class i and $C_{V_{ab}}$ represents those traces which contain the 'ab' sequence (represented by a non-zero value of V_{ab}). Here, c can be the total number of classes.
Definition 3: TF-IDF is the product of TF and IDF values. The TF-IDF of an n-gram sequence is defined as

$$TF - IDF(ab, C_i) = f(V_{ab}, X_i) * log(C_i/C_{V_{ab}}) \tag{5.3}$$

We have considered the TF-IDF values of the n-grams as a selection criterion. The top-ranked n-grams are selected using the TF-IDF feature selection method from the FV_DB database and the output is stored in the Test_DB. The details are given in Section 5.3.3.

5.3.2 Learning and detection

This stage uses a detection sub component, called mX_DetectionEngine used for capturing the malicious behaviors of a TVM. Each TVM may have a different guest OS, services and applications installed in them. Hence, an individual TVM will have a unique intrusion profile. It operates in two phases: learning and detection. In detection phase, mX_DetectionEngine generates an intrusion profile as a baseline profile to classify active processes of monitored TVM as normal or malicious. The mX_DetectionEngine uses the Random Forest machine learning technique to learn/detect the behavior of evasive samples.

The performance of individual classifiers is compared with an ensemble of some classifier. We found that Random Forest (RF) technique[168], which is an ensemble learning classifier, provides better results than single classification algorithms. RF performs efficiently even for huge databases. It reduces the over fitting problem caused by a single decision tree [160]. An ensemble learning classifier generates many classifiers and aggregates their results. RF combines the idea of 'bagging' and the random selection of features. In bagging, successive independent decision trees are constructed using a bootstrap sample (training subsets originated from random sampling with replacement of the training set). At the end, a majority vote or average is taken for predictions. The algorithmic steps of RF are briefly explained below:

1. Let N be the number of training cases and P be the number of variables in the classifier. Each of the training instances is represented by $(X, Y) : (x_1, y_1), (x_2, y_2), (x_3, y_3) ....(x_n, y_n)$.
2. A random sample (bootstrap sample) is generated by choosing n times with the replacement of the training set; (X_b, Y_b) replaces (X, Y). Use the rest of the cases to estimate the error of the tree, by predicting their classes.
3. Let p input variables be randomly chosen to determine the decision at a node of the tree; p should be much less than P.
4. Grow an unpruned decision tree (f_b) on this bootstrap sample (X_b, Y_b).
5. For each node, best split is chosen based on the selected p variables. Gini Index is used for finding the best split node.

   $$Gini(T) = 1 - \sum_{j=1}^{j=n} (P_j)^2 \tag{5.4}$$

   where T refers to a dataset with n classes and P_j is the relative frequency of class j in T.
6. Go to step 2. Repeat for each bagging iteration. The total bagging iterations (say B) depends on the total number of estimators considered, derived by parameter-tuning during experimentation.

An unseen sample is passed to each of the trained trees. The output can be defined as the average of the outputs of the generated trees. If bagging is performed repeatedly for 'B' times then a prediction of an unseen sample x' can be made by

$$f = (1/B) \sum_{b=1}^{b=B} fb(x') \tag{5.5}$$

RF has capability to generate rules and provide better detection results. Prediction of a single tree is highly sensitive to noise in the training data. The trained model is stored as a baseline pre-compiled intrusion profile of monitored TVM (described in Section 5.2.4).

In the detection mode, `mX_DetectionEngine` uses the pre-compiled intrusion profile of TVM to classify the running processes (extracted in Test_DB). The execution flow for `mX_DetectionEngine` for detecting the malicious processes is given below:

1. Test_DB file is loaded, representing the current behavior of the monitored TVM.
2. The intrusion profile (trained model) of the monitored TVM, is loaded.
3. The running processes are classified.
4. The alert and log generator is notified on detection of any suspicious activities.
5. A alert with the details of log report is sent to cloud administrator for further activities. The implementation of VIMD using VMGuard for behavior analysis is described in detail below.

5.3.3 Implementation of VIMD with VMGuard as core detection mechanism

VIMD with VMGuard as core detection mechanism has been implemented and validated using a well-known University of New Mexico (UNM) dataset[155] which has been previously used by researchers [73] [84] working in cloud security and is publicly available. The six datasets of UNM, considered for behavior analysis for which enough traces were available, are shown in Table 5.2. Each UNM dataset represents the benign and intrusive system call traces of privileged processes such as sendmail, ps and login. Malicious behavior has been introduced into each of these processes, and the description of the traces of the programs in each dataset is given here [155]. The prototype implementation has been performed in a machine with 16 GB RAM, Core i7 processor, 500

Table 5.2: Details of UNM Dataset

Data set	Number of Intrusive traces considered for training	Number of Normal traces Considered for training	Total n-grams considered (after TF-IDF processing)
CERT synthetic sendmail	23	294	2312
MIT live lpr	460	1001	11201
UNM synthetic lpr	1001	9	14384
UNM live lpr	1001	1231	16959
UNM Live ps	26	24	866
UNM Live stide	105	645	7321

GB HDD, Ubuntu 15.10 as the host OS, Xen 4.6 as hypervisor and two guest VMs: one with Windows 7 and other with Ubuntu 14.04. Python 2.7.10 has been used for implementing the IDS approach.

The evaluation of VMGuard using UNM dataset is discussed. As, the dataset is in form of the traces, we have used this dataset only for evaluating the detection accuracy of VMGuard for program subversion attacks. The validation of other components have been discussed in the implementation of VAED in Section 5.4.4. VAED is validated with the malware dataset, obtained on request from University of California, given by Kirat et al. [158]. The dataset was in form of executable programs which are executed at TVM and extracted by VIMD at VMM-layer using PET, scanned by PV component and analyzed by VAED.

5.3.3.1 Pre-processing of Dataset and Feature Extraction

In order to prepare the dataset for the classifier, first of all the traces need to be arranged in the form of n-gram sequences. Many n-gram sequences of normal and intrusive traces are obtained. The suitable n-grams of normal and intrusive traces are selected using the TF-IDF method. The top 70% of the n-grams with the highest TF-IDF values have been considered for normal traces. Similarly, the top 70% of the n-grams with the highest TF-IDF values for the intrusive traces have been considered. All the n-grams are merged to form a common database for intrusive and normal behavior of a corresponding process with duplicates eliminated. Two files are maintained during implementation for each UNM dataset: FVM_DB and the mapping file. FVM_DB presents

Table 5.3: Collection of n-grams with their Term-Frequency in each Normal Trace

Benign	Ng1	Ng2	Ng3	Ng4	Ng5	Ng6
N1	2	1	0	2	0	2
N2	3	4	0	0	0	3
N3	0	3	1	2	2	1
N4	0	0	0	0	1	1
N5	0	0	0	0	1	0

Table 5.4: Collection of n-grams that belong to Normal Traces with their Document-Frequency

Type	Ng1	Ng2	Ng3	Ng4	Ng5	Ng6
Benign	2	3	1	2	3	4

Table 5.5: Collection of n-grams that belong to Normal Traces with their Inverse Document Frequency

Type	Ng1	Ng2	Ng3	Ng4	Ng5	Ng6
Benign	1.32193	0.73697	2.32193	1.32193	0.73697	0.32

Table 5.6: TF-TDF calculation of various n-grams of Normal Traces

Benign	Ng1	Ng2	Ng3	Ng4	Ng5	Ng6
N1	2*1.32=2.64	1*.73=0.73	0*2.32=0	2*1.32=2.64	0*.73=0	2*.32=0.64
N2	3*1.32=3.96	4*.73=2.92	0*2.32=0	0*1.32=0	0*.73=0	3*.32=0.96
N3	0*1.32=0	3*.73=2.19	1*2.32=2.32	2*1.32=2.64	2*.73=1.46	1*.32=0.32
N4	0*1.32=0	0*.73=0	0*2.32=0	0*1.32=0	1*.73=0.73	0*.32=0
N5	0*1.32=0	0*.73=0	0*2.32=0	0*1.32=0	1*.73=0.73	0*.32=0
TF-IDF	6.60 (rank 1)	5.84 (rank 2)	2.32(rank 5)	5.28 (rank 3)	2.92(rank 4)	1.92(rank 6)

the feature vector represented by $< L, X'_i >$ where L is a label for each feature vector X'_i; $L\epsilon\{normal, intrusive\}$. The mapping file represents the unique n-grams executed at the TVM by the running programs.

The process of creating TF-IDF for different n-grams is as follows:

Let us consider 5 benign programs and 5 intrusive programs. Each program contains a total of 6 different n-grams in different quantities. The term-frequency table of benign behavior is shown in Table 5.3. Term-frequency counts the total number of occurrences of n-grams in a trace. For example, n-gram Ng_1 appears 2 times in the normal trace N_1 and 3 times in the trace N_2. It does not appear in any of the other normal traces. Hence, we can say that the V_{Ng1} value of N_1 is 2 and V_{Ng_1} of N_2 is 3. Any non-zero value of an n-gram corresponds to the presence of an n-gram in the trace, and zero value represents the absence of an n-gram in the trace. Document Frequency represents, how

Table 5.7: Evaluation metrics and their description

Parameter	Description
True Positive (TP)	IDS detects intrusive program execution as malicious
False Positive (FP)	IDS detects normal program execution as malicious
True Negative (TN)	IDS detects normal program execution as normal
False Negative (FN)	IDS detects intrusive program execution as normal
TPR (Detection Rate)	TP/(TP+FN); The proportion of correctly classified intrusions to the actual size of the attack class
False Positive Rate (FPR)	FP/(TN+FP); The proportion of incorrectly classified intrusions to the actual size of the attack class
False Negative Rate FNR	FN/(TP+FN);The proportion of incorrectly classified normal programs to the actual size of the normal class

many traces contain a particular n-gram. For example, DF(Ng_1)=2 means that Ng_1 appears in two normal traces as shown in Table 5.4. Now the Inverse Document Frequency is used to improve the weight of the rare n-grams, as rare sequences have more discriminating power as shown in Table 5.5. It is calculated by the the taking the logarithmic ratio of total traces to traces which contains the n-grams $log_2(C/C_{Ng})$. Now the values in Table 5.6 are obtained by multiplying the TF score by the IDF score for each n-gram. The values corresponding to different traces are summed to find the final TF-IDF of the n-gram. The top five n-grams selected after the feature selection process would be $Ng_1, Ng_2, Ng_4, Ng_5, Ng_3, Ng_6$.
Similarly,the TF and IDF scores for n-grams for each of the intrusive traces have been calculated. In our implementation with the UNM data set, we have collected different sets of n-grams for each dataset after TF-IDF processing.

5.3.3.2 Results and Discussion

The standard performance metrics are described in Table 5.7. For evaluation, we have applied a k-fold cross-validation method over the collected data sets and taken k as 10. The dataset for each category of a privileged process is tested. Initially, the UNM dataset is tested with different machine learning algorithms such as Random Forest (RF), Support Vector Machine (SVM) [169], Naive Bayes (NB)[170] and Ensemble Classifier (AdaBoosted SVM) [171]. NB makes the assumption that features are independent of each other. This assumption may not

Table 5.8: Detection rate (%) of different classifiers for UNM dataset

Dataset	**Ensemble Classifier (Random Forest)**	**SVM**	**Naive Bayes**	**Ensemble classifier (boosting with SVM)**
CERT_synthetic_sendmail	98.7382	97.4763	62.4606	98.7382
MIT_live_lpr	99.9316	98.13	97.3306	98.69
UNM_live_lpr	99.9552	98.6	98.2975	99.121
UNM_live_stide	99.6	97.3333	99.6	98.131
UNM_ps	94	84	90	90
UNM_synthetic_lpr	100	99.12	96.8317	99.12

Table 5.9: Detailed Performance Results of Random Forest Technique for UNM dataset

Process Name	**Correctly Classified Instances**	**Incorrectly Classified Instances**	**TPR (%)**	**FPR (%)**	**FNR (%)**
CERT_synthetic_sendmail	313	4	98.7382	4.1	1.2818
MIT_live_lpr	1460	1	99.9316	1	0.1
UNM_live_lpr	2231	1	99.9552	1	0
UNM_live_stide	747	3	99.6	2.5	0.4
UNM_ps	47	3	94	6.2	6
UNM_synthetic_lpr	1010	0	100	0	0

be true for intrusion detection, where one system call pattern may be co-related with other system call patterns. However, the training time of NB is linear and provides results much faster than other classifiers. Training time of the SVM-based classifier was high for larger dataset when compared to other classifiers. This is due to the algorithmic complexity associated with SVM in applying a suitable kernel function over the dataset and fine tuning the parameter settings used.
Here we have applied a radial-basis kernel-based SVM. A kernel-based SVM requires O(n*m) computations for training the dataset, where n is the total number of training instances and m is the total number of features. Hence, the training phase is computationally expensive. On the other hand, RF, which is an ensemble of Decision Trees, builds the trees in O(M(nmlogn)), where M is the number of trees initialized while running the algorithm, n is the number of instances and m is the number of features. RF is comparatively efficient even for huge datasets. Each of the parameter values is fine tuned according to the performance of the classifier for particular types of intrusion dataset.

We found that boosting the SVM increases its performance but also results in more complex analysis leading to slower training for larger datasets. We found that RF provides better results than the other three classifiers for the UNM datasets as shown in Table 5.8. The results are also shown in the Figure 5.9.

RF technique is efficient for large datasets and is one of the most

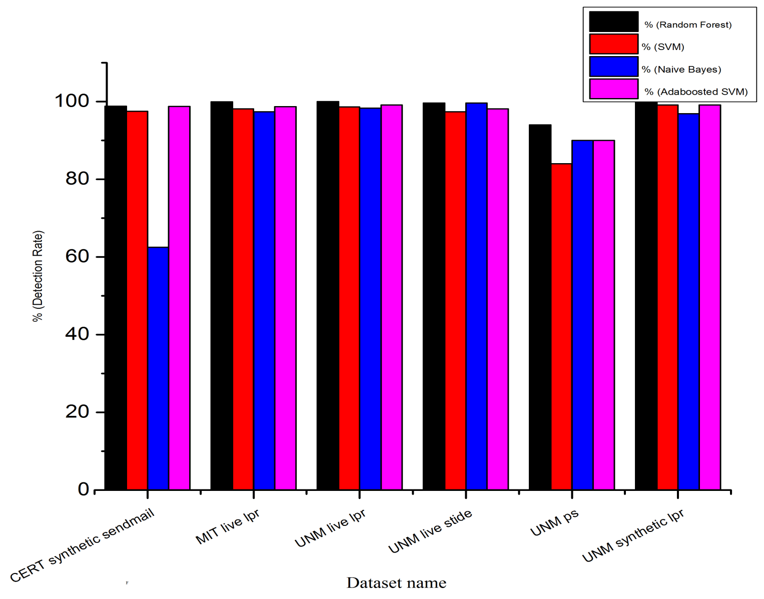


Figure 5.9: Detection Rate of Different Classifiers for UNM Dataset

accurate learning algorithms available. It is capable of handling several thousands of input variables; hence RF performs better for datasets with richer feature set. For UNM_ps, RF achieves 94% TPR whereas SVM achieves 84% TPR. Naive Bayes and adaboosted SVM achieve 90% TPR. RF achieves a 100% detection rate for UNM_synthetic_lpr and around 99.9% TPR for most of the UNM datasets. It is less sensitive to outliers and parameter choices. Adaboosted SVM provides similar results but is sensitive to noisy data and outliers, and is found to produce more false alarms (FA) for UNM traces (10%<FA<25%). During or experimentation of VMGuard, many thousands of n-grams are generated for datasets. Some of the n-grams may be noisy and may not provide complete information. Therefore, RF is more suitable for such cases, and the results seem to be promising. It provides a good detection rate for most attacks (94%-100%) with acceptable false positives (FPR) and false negatives (FNR). RF provides the highest detection rate of 99.9552% with 1% FPR and negligible FNR for the largest dataset considered (i.e. UNM_live_lpr). The detection rate is low, around 94%, for the UNM_ps dataset, with 6.2% FPR and 6% FNR. The performance for other datasets can be seen in Table 5.9 and is illustrated in Figure 5.10. The results show that RF provides good results with fewer false alarms.

It was observed that the dataset size alone is not sufficient for achieving a good detection rate with low false alarms; the feature vectors

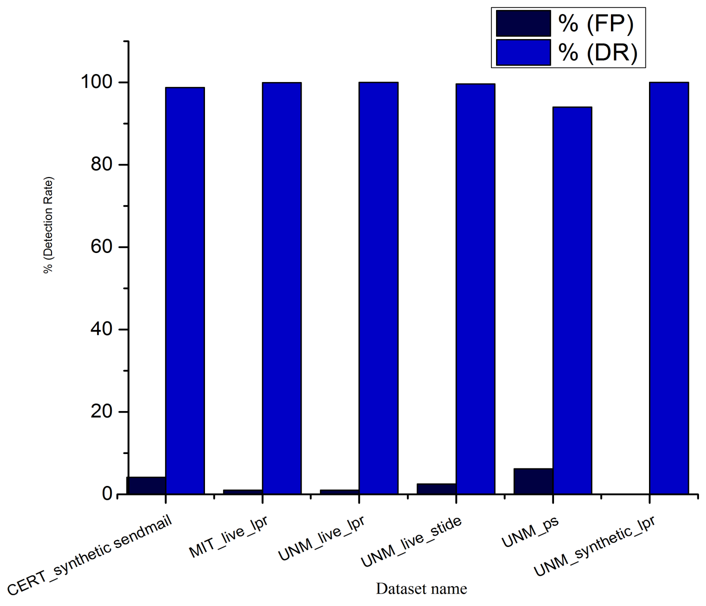


FIGURE 5.10: Detection Rate and False Positive Rate for different UNM Datasets using Random Forest

obtained from the dataset are equally important. Each feature vector represents the behavioral characteristics of a malware class, which should be helpful to distinguish it from other classes. We also observed that if one machine learning algorithm provides better results for detecting one particular intrusion, it may not provide the similar results for other intrusions. For example, RF provides 99.95% TPR for UNM_live_lpr. However, it provides the much lower detection rate of 94% for the UNM_ps dataset. This is because of the behavioural differences between various attacks, represented by different sets of feature vectors.

The classifiers'performance depends on the discriminative power of features. Further, we can say that a classifier with low time complexity may or may not provide a better classification rate, like Naive Bayes, and a classifier with a higher detection rate may have slow training for large datasets, such as Adaboosted SVM. We observe that the ensembles of classifiers provide better result than individual classifiers, because the error rate generated by one classifier may get minimized when creating an ensemble with another classifier. For example, we observed that the overall results of Ensemble Trees (RF) and Adaboosted SVM are better then when only SVM or Naive Bayes is considered, as shown in Figure 5.9.

Table 5.10: Performance Overhead: Processing Time per Sample (seconds)

Process Monitoring	
Parameters	Time (best-worst) (seconds)
Introspection time	60-700
Process Validation	0.06 -1.453
Trace Pre-Processing	0.0231 -45.132
Detection Time	1.011 - 1.121
Total	61.0341 - 747.706

The overhead associated with the detection approach, per sample execution, is discussed next. Since, UNM dataset was in form of traces, the same Windows malware dataset, used for evaluating PV, is considered for overhead calculation because it contains the executable programs. The system overhead directly depends on the program trace analysis time. It is observed that the analysis time is high for longer traces and low for shorter traces. The traces are extracted on a time frame of fixed size (seconds). A time frame of 60 seconds is considered as a best case, assuming all traces terminated within that time. It is observed during experimentation, most of the traces usually terminated with 700 seconds. It is taken as the worst case, which would seldom occur. The maximum time depends on the largest trace. Program-trace analysis time depends on following: system call tracing time (60 seconds-600 seconds), process validation time (0.06 seconds $\sim$ 1.453 seconds), trace pre-processing time (0.023 seconds - 45.132 seconds) and detection time (1.011 seconds $\sim$ 1.121 seconds). Therefore, the time to process a sample was captured as 61.0341 seconds (best case scenario) $\sim$ 747.706 seconds (worst case scenario) respectively which is the total time execution of all phases. The maximum overhead incurred in processing a sample was observed as 747.706 seconds. If short processes are running in the memory which terminates frequently, the overhead will be less. The overall time depends on the number of applications running in the TVM and the trace length. The observed values, based on the experimentation, are shown in Table 5.10. The additional time is taken for creating intrusion profiles of malwares in an offline mode.

The execution of both process validation and behavior analysis using VMGuard have been successfully evaluated. VMGuard is found to perform well to detect the program subversion attacks and provides the accuracy of 94%-100% with low false positives (0%-6.2%) in detecting program subversion attacks. The detection mechanism of VMGuard is effective and easy to be implemented and does not any require complex analysis making it a good choice for security administrators.

5.4 VAED: Detection and Implementation

VAED analyzes the execution traces, obtained from the memory introspection phase of VIMD using three sub-detection components: (i) Program Semantics Extractor (PSE) (ii) Feature Transition Matrix Generator (FTMG) (iii) eX_DetectionEngine. PSE pre-processes the collected traces and generates SCDG for each trace. SCDG is useful to extract useful behavioral semantics of various system call transitions of each program. FTMG generates the matrix of probability values of transitions for each SCDG and eX_DetectionEngine applies machine learning to learn and detect the evasive malware behavior base on the extracted behavioral values. The execution flow of various security functions using VAED is given in Figure 5.11. A detailed description of its detection stages and the implementation of VIMD with VAED as core detection component is presented in subsequent sections.

5.4.1 Program-semantic extraction using SCDG

The execution traces collected from memory analysis phase are preprocessed to construct the behavioral semantics. This stage uses the detection sub-component Program Semantics Extractor (PSE) for this purpose. PSE extracts the program semantics by generating SCDG for each trace. In SCDG generation, each trace is represented as a directed graph , based on Markov Chain principle [172]. Each transition between two system calls is assigned a probability weight. A graph G is represented by <V, E>. V is a set of vertices where each vertex represents a unique system call occurring in the trace. E is a set of edges where each edge represents a unique transition between two system calls. The transition could be between same system calls such as NtAllocateVirtualMemory→ NtAllocateVirtualMemory occuring multiple times. An edge weight $P_{e_{ij}}$ is used to represent the probability between two vertices v_i and v_j. Edge weight ($P_{e_{ij}}$) denotes the transition probability between $v_i \rightarrow v_j$ in a Markov Chain.

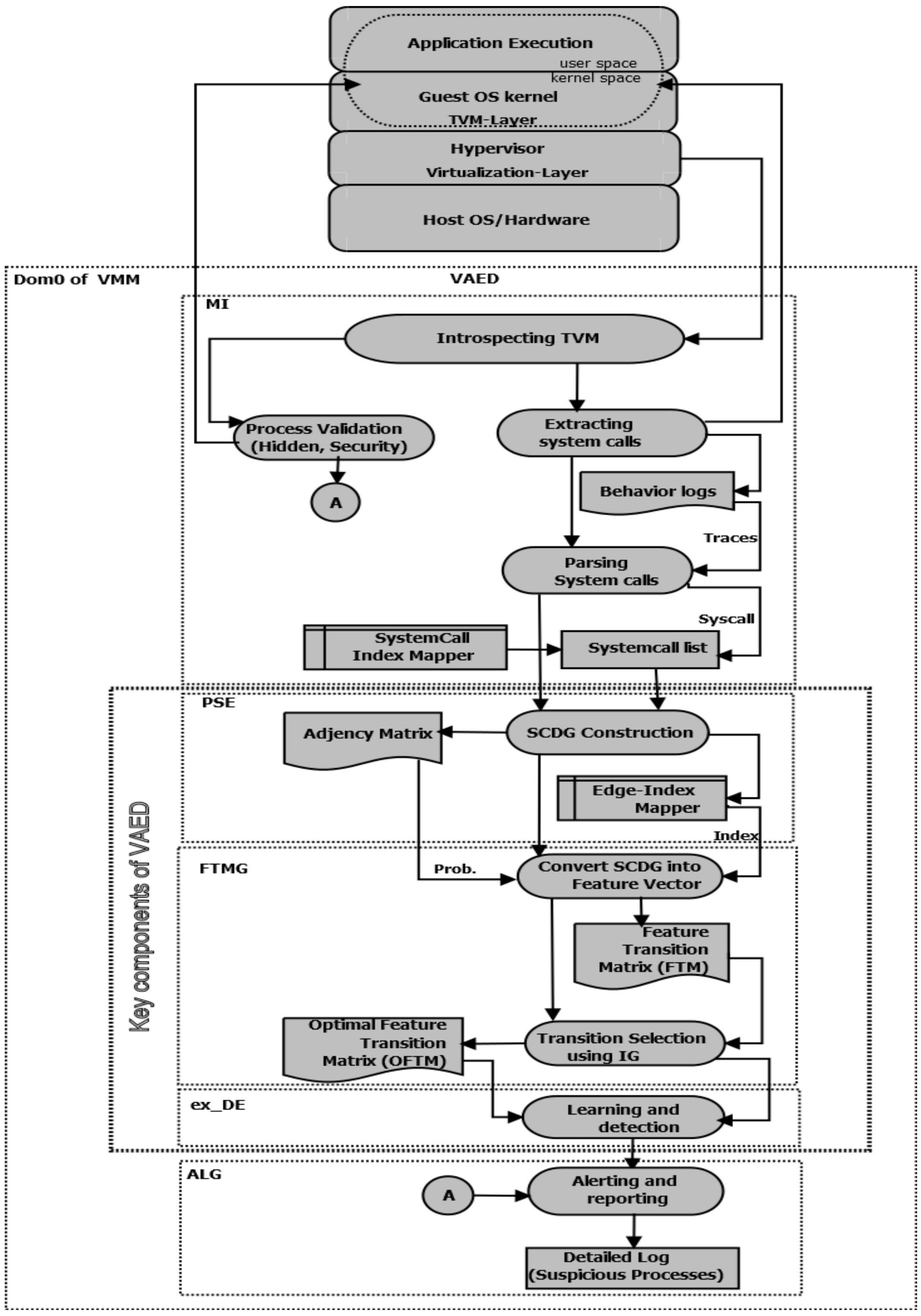


Figure 5.11: Execution Flow of various security functions of VIMD with behavior analysis using VAED

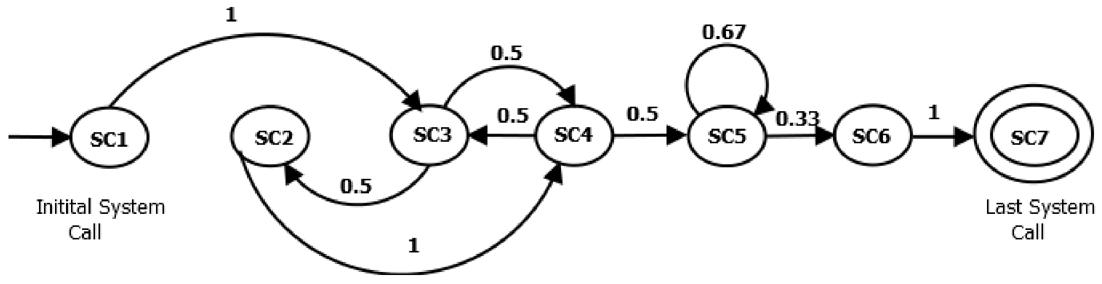


FIGURE 5.12: System Call Dependency Graph

According to Markov Chain property, for each vertex v_i:

$$\sum_{v_j=1}^{n} P_{e_{ij}} = \begin{bmatrix} 0 & \text{No transition from v}_i \\ 1 & \text{Otherwise} \end{bmatrix} \tag{5.6}$$

This means that summation of the transition probabilities of all outgoing edge of a vertex should be 1. The summation is 0 if no outgoing edges exist from a vertex. Secondly, the transition probability ($P_{e_{ij}}$) should be calculated by formula as follows:

$$P_{e_{ij}} = f(v_i \rightarrow v_j) / \sum_{v_{j=1}}^{v_j=n} f(v_i \rightarrow v_j) \tag{5.7}$$

Here f represents the frequency of occurrence of a transition. An example how SCDG is created and stored in adjacency matrix for a given sample is described. Let Γ denotes a system call trace. Γ={NtQueryInformationProcess →NtQuerySystemInformation→ NtAllocateVirtualMemory → NtOpenDirectoryObject → NtQuerySystemInformation → NtOpenDirectoryObject → NtOpenFile → NtOpenFile → NtOpenFile → NtOpenKey → NtClose}. VAED replaces each system call by unique index extracted from system call-index mapper. Hence, trace Γ can be represented by {SC1, SC3, SC2, SC4, SC3, SC4, SC5, SC5, SC5, SC6, SC7} sequence. The SCDG generated by applying Markov Chain Principle (using equation 5.6 and 5.7) is shown in Figure 5.12. SCDG is stored in the form of adjacency matrix (AM) as shown in table 5.11. AM is an n*n matrix where n=$|V|$, representing the trace behavior. It can be seen from the table that summation of the cell values of a row are 1 or 0 for each vertex. Value 0 represents there is no outgoing edge from the vertex (eg. SC7). If a particular transition is occurring in a particular malware sample, it may also occur in another sample. Hence, each

TABLE 5.11: Adjacency Marix corresponding to SCDG

Index	SC1	SC2	SC3	SC4	SC5	SC6	SC7
SC1	0	0	1	0	0	0	0
SC2	0	0	0	1	0	0	0
SC3	0	0.5	0	0.5	0	0	0
SC4	0	0	0.5	0	0.5	0	0
SC5	0	0	0	0	0.67	0.33	0
SC6	0	0	0	0	0	0	1
SC7	0	0	0	0	0	0	0

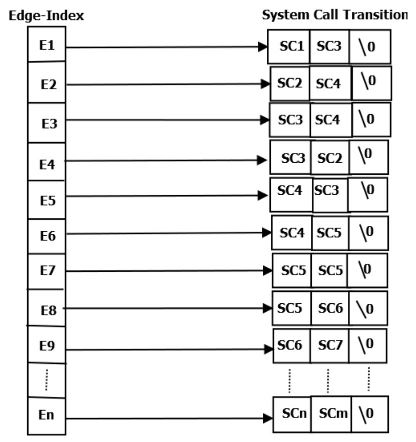


FIGURE 5.13: Edge-Index Mapper of VAED

unique transition between two subsequent system calls, is stored in an Edge-Index Mapper (EIM) as shown in Figure 5.13. EIM is a common mapping file maintained by VAED which provides the mapping of all possible transitions of system to an edge-index and is updated when an SCDG is generated.

5.4.2 Feature Transition Matrix Generation

The creation of SCDG for malware and benign files are not alone sufficient to detect malwares. Graph isomorphism is a NP complete problem. Hence, VAED does not employ graph similarity measures as used by graph similarity solutions such as gSPAN [173]. VAED generates the Feature Transition Matrix (FTM) of each of the SCDG using a proposed scheme as shown in Algorithm 3. This detection stage uses the detection sub-component Feature Transition Matrix Generator (FTMG) for same purpose. The FTMG takes Adjacency Matrix (AM), Edge-Index Mapper (EIM) and SCDG as input. As stated above, AM contains the transition probability information ($P_{e_{ij}}$) for each transition $v_i \rightarrow v_j$

calculated by applying equation (1) and (2) of Markov Chain principle. EIM represents the unique edge label for each transition. Each transition of SCDG is mapped into a unique <transition-value pair> which are stored in a Feature Vector (V) and represents the semantic information of the malware patterns occurring in the trace. All <transition-value pair > are generated and stored in V. For the example trace Γ, the obtained values in V are {(E1, 1), (E2, 1), (E3, 0.5), (E4, 0.5), (E5, 0.5), (E6, 0.5), (E7, 0.67), (E8, 0.33), (E9, 1) }. The information in V is stored in the FTM. The first column of FTM represents the type of SCDG (attack class (T, E, P)/benign class of trace) and first row represents the labels of unique transitions. All other cell values represent the transition probabilities. It is zero, if no transition exist for a given trace.

FTM contains a large number of features (transitions). The presence of non-zero value corresponding to each feature indicates the presence of system call transition in the trace. A zero value indicates the absence of corresponding system call transition. The values represent the probabilities obtained by applying markov chain property in each trace. If a classifier is trained for all feature values, it may deteriorate its performance. Hence, we applied feature selection in the FTM to select most important and optimal set of transitions. Hence, we applied feature selection to select most important and optimal set of transitions from FTM. We have applied Information Gain Ratio (IGR) [174] to select features in continuous values dataset. Information Gain (IG) is biased towards multi-valued attributes. Information Gain Ratio (IGR)

Algorithm 3: Algorithmic steps of generating Feature Transition Matrix (FTM)

```
Input: SCDG, Adjacency Matrix (AM), Edge-Index Mapper (EIM) and Empty
Feature Vector (V), Empty FTM
Result: FTM
for each SCDG do
    for each vertex $v_i$ in SCDG do
        for each outcome edge $v_i \rightarrow v_j$ of vertex $v_i$ do
            $P_{e_{ij}}$=Read_Probability (AM);
            $E_{ij}$ = Extract_Label(EIM);
            Add transition-value pair $<E_{ij}, P_{e_{ij}}> \rightarrow$ V;
        end
    end
    Append Values Feature Vector (V) to FTM;
end
```

solves the drawback of IG. IGR extends the IG to make it applicable to be used with attributes with large number of values. It provides the normalized score of feature's contribution for classification.

Let S be the sample of benign and malware traces in FMT. $|S|$ represents the total number of sample traces in S with class category i: {0 (for Exception-based), 1 (for Time-based), 2 (for Process-feature based), 3 (for (Benign class) }. The expected information needed to classify an instance (tuple) for partition S is known as entropy. This information is useful to identify a class label of an instance in S. In S, let k_i be the total number of traces having class category i. Then entropy of S is calculate by following formula:

$$H(S) = -\sum_{i=0}^{i=3} P_i log_2 P_i \tag{5.8}$$

where

$$P_i = k_i/|S| \tag{5.9}$$

P_i denotes the probability that a random instance in partition S belongs to class i. Also, 0 <= H(S) <= 1,the information entropy H(S) represents the purity of the collection of information. The smaller the value of entropy, the lesser the degree of chaos.

If the instances in S has to be partitioned (classified) on some feature attribute A {a1, a2, a3......av}, then S will split into v partition sets {S1, S2, S3.....Sv}. Then the information entropy of S based for an attribute A will be calculated by formula:

$$H_A(S) = \sum_{j=1}^{j=v} \frac{|S_j|}{|S|} * H(S_j) \tag{5.10}$$

where $|S_j|$ is total number of samples in each subset and $H(S_j)$ is the entropy of partition $|S_j|$ calculated by formula 5.8 and 5.9. Information Gain (IG) is used to measure the information obtained from the transition for the class prediction and calculated by following formula by partitioning on A:

$$InformationGain(A) = H(S) - H_A(S) \tag{5.11}$$

where H(S) is the information entropy of the whole data set. IGR utilizes the split information value which corresponds to the information obtained by partitioning the dataset S into v sub-partitions on some attribute A. The split information on A is calculated by following formula:

$$SplitInfo_A(S) = \sum_{j=1}^{j=v} \frac{|S_j|}{|S|} * log_2 \frac{|S_j|}{|S|} \tag{5.12}$$

where high value of SplitInfo infers that partitions have equal size (uniform) and low SplitInfo refers that few partitions contains most of the tuples. Finally the Information Gain Ratio (IGR) is defined as:

$$InformationGainRatio(A) = InformationGain(A)/SplitInfo(A) \tag{5.13}$$

The more the value of IGR for an attribute A, the more weighted it has got for considering it for classification. All the features of FTM are ranked by using the discussed feature selection approach and top ranked features are selected to form optimal feature transition matrix (OFTM). Features with negligible information gain ratio are discarded. The learning and detection module is described in next section.

5.4.3 Learning and detection

One of the important security function of VAED is to learn and detect the behavior of evasive malware. This stage uses the detection subcomponent `eX_DetectionEngine` for this purpose. It operates in two mode: learning and detection. In learning mode, `eX_DetectionEngine` creates the intrusion profiles of the TVM by employing the classifier fusion of diverse classifiers over OFTM, in offline mode. A OFTM is a probability transition matrix which represents the transition probabilities of selected transitions for each class category of known sample (Exception-based Evasion, Time-based Evasion, Processor-Feature based evasions and Benign samples). VAED applies the weighted voting based scheme [175] to fuse the results of the diverse classifiers. This scheme is not computationally intensive like other fusion methods [176] and applicable to real time applications [177]. In a case study done by Moreno-Seco et al. [178], weighted voting scheme over diverse classifiers found to perform well which improved the results of individual classifies.

Hence, different types of classifiers (diverse classifiers) have been considered: probability based (Naive Bayes (NB)[160]), hyperplane based (Support Vector Machine (SVM) [179]), rule-based (Decision Tree (DT) C 4.5 [180] and Random Forest (RF) [168]). The detection engine can be mathematically derived as follows by using n set of classifiers:

A training set $X : \{(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4).......(x_k, y_k)\}$ is considered where each x_i instance is a set of features and each y_i is the class label for instance x_i. Here y_i is class label in set $Y : \{c_1, c_2, c_3, ..., c_m\}$. A compound classifier can be obtained for training set X which is derived by classifier fusion and achieves highest accuracy than single classifier results.

A total of n classifiers: $\{\xi^1, \xi^2.........\xi^n\}$ are assumed. Each of them decides if a sample in X belongs to one of the class in Y based on their output prediction probability. Using the naive brute force approach [181], each of them is assigned a weight w_i from set $W = \{w_1, w_2, w_3......w_n\}$. In brute force approach, each classifier is prior trained for all possible combination of weights in the range of $(1, m)$ and best combination weights are chosen which provides the highest accuracy for combined classifiers' output. This parameter tuning of weights is done only once in algorithm in an offline mode. Once weights are determined, they will remain fixed for the detection algorithm.

For making a common decision by the ensemble of diverse classifiers; a common decision model (M) for X, can be derived by the following classifier fusion equation:

$$\xi'(X) = argmax_{j\epsilon c_k} \sum_{l=1}^{l=n} \delta(j, \xi^l) * w_l \tag{5.14}$$

where $\delta(j, \xi^l)$ is the predicted probability of the classifier for class j. The output is decided based on the maximum prediction probability derived by the classifier fusion by using equation 5.14. The class which has the maximum prediction probability is returned as output for the test instance x. Here the prediction probability of fused classifier for a class (c_k) is the summation of the multiplication of predicted probability and classifier's weight (w_l) for all n classifiers. The ensemble classifier is trained for the malware and benign classes and the trained model is stored in the baseline database, representing a *pre-compiled intrusion profile*. The details about intrusion profile generation is explained in Section 5.2.4.

In the detection mode, `eX_DetectionEngine` uses the pre-compiled intrusion profile of TVM (stored in form of `Detection_Model`) to classify the running processes. The execution flow for `eX_DetectionEngine` for detecting the malicious processes is given below:

1. PET is invoked to take live trace of all processes running in TVM.
2. OFTM_test is generated by following the above discussed procedure of graph construction, FTM generation and feature selection.
3. OFTM_test file is loaded representing the current behavior of monitored TVM.
4. A trained `Detection_Model` is loaded which represents the Intrusion Profile (IP) of monitored TVM.
5. `Detection_Model` is executed with test set OFTM_test and classify the running processes.
6. An alert is generated on detection of suspicious activities to cloud admin.

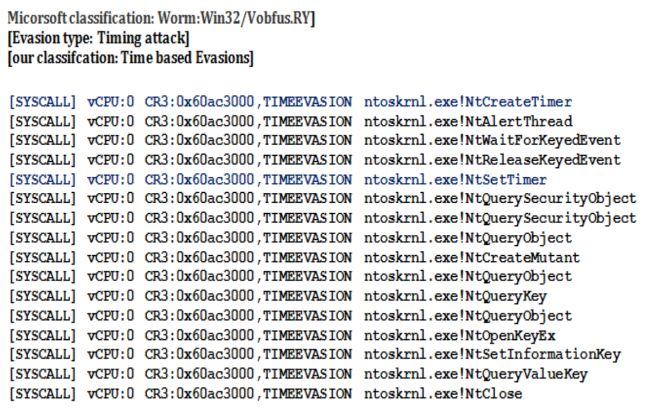
Cloud administrator performs further analysis on suspicious processes based on the application and user activity logs of TVM and behavior logs generated by VAED. Cloud administrator can remove the applications generating the malicious traces. If a TVM generates a large number of malicious processes, it is isolated and restored to a previous checkpoint when it was benign. The implementation of VIMD using VAED for behavior analysis is described in detail below:

5.4.4 Implementation of VIMD with VAED as core detection mechanism

VIMD with VAED as core detection mechanism has been validated with the evasive malware sample of windows evasive binaries. The evasive samples are collected from University of California [158] on request. We have considered 80 samples of time-based evasions, 86 samples of processor-feature based evasions, 181 samples of exception-based evasions and 132 benign samples named as 'benign'. Two physical machines with 16 GB RAM, 1 TB HDD, core i7 processor, Ubuntu 15.10 and Xen 4.6 hypervisor have been considered. This can be considered as compute server of cloud. We have also configured two guest VMs with Windows 7 guest OS and 8GB RAM and 100 GB HDD which can be considered as TVMs. Python 2.7.10 has been used for implementing the proposed VAED approach.

5.4.4.1 Preparing the Dataset and Feature Extraction

The dataset contains the imbalanced distribution of the classes. Hence, an adaptive synthetic sampling approach (ADASYN) [182] is applied over the dataset to perform the oversampling of the minority samples that are harder to classify. In which, some synthetic samples are created using the euclidean distance from K nearest neighbors that belong to the minority class. There exists other sampling methods such as SMOTEBoost [183] and DataBoost-IM [184] for dealing with class imbalance problem. However, ADASYN is more efficient than other approaches. Unlike other approaches, it does not rely on the hypothesis evaluation for generating the synthetic samples. ADASYN adaptively updates the distribution based on the characteristics of data distribution and found to perform well when compared to other approaches [182]. The sampled dataset is ready for experimentation. Initially the experimental set up is used to generate the evasive malware and benign sample traces by using the PET component. Benign applications such as drivers, network utilities, games, multimedia, browser, social chat messenger, teamviewer etc. are installed in the TVMs. The original TVM state is saved and a backend/snapshot of original disk volume is created. A clone TVM is created and the malware evasive binaries are injected in a clone TVM from hypervisor using libguestfs tool [166] at



```
Micorsoft classification: Worm:Win32/Vobfus.RY]
[Evasion type: Timing attack]
[our classifcation: Time based Evasions]

[SYSCALL] vCPU:0 CR3:0x60ac3000,TIMEEVASION ntoskrnl.exe!NtCreateTimer
[SYSCALL] vCPU:0 CR3:0x60ac3000,TIMEEVASION ntoskrnl.exe!NtAlertThread
[SYSCALL] vCPU:0 CR3:0x60ac3000,TIMEEVASION ntoskrnl.exe!NtWaitForKeyedEvent
[SYSCALL] vCPU:0 CR3:0x60ac3000,TIMEEVASION ntoskrnl.exe!NtReleaseKeyedEvent
[SYSCALL] vCPU:0 CR3:0x60ac3000,TIMEEVASION ntoskrnl.exe!NtSetTimer
[SYSCALL] vCPU:0 CR3:0x60ac3000,TIMEEVASION ntoskrnl.exe!NtQuerySecurityObject
[SYSCALL] vCPU:0 CR3:0x60ac3000,TIMEEVASION ntoskrnl.exe!NtQuerySecurityObject
[SYSCALL] vCPU:0 CR3:0x60ac3000,TIMEEVASION ntoskrnl.exe!NtQueryObject
[SYSCALL] vCPU:0 CR3:0x60ac3000,TIMEEVASION ntoskrnl.exe!NtCreateMutant
[SYSCALL] vCPU:0 CR3:0x60ac3000,TIMEEVASION ntoskrnl.exe!NtQueryObject
[SYSCALL] vCPU:0 CR3:0x60ac3000,TIMEEVASION ntoskrnl.exe!NtQueryKey
[SYSCALL] vCPU:0 CR3:0x60ac3000,TIMEEVASION ntoskrnl.exe!NtQueryObject
[SYSCALL] vCPU:0 CR3:0x60ac3000,TIMEEVASION ntoskrnl.exe!NtOpenKeyEx
[SYSCALL] vCPU:0 CR3:0x60ac3000,TIMEEVASION ntoskrnl.exe!NtSetInformationKey
[SYSCALL] vCPU:0 CR3:0x60ac3000,TIMEEVASION ntoskrnl.exe!NtQueryValueKey
[SYSCALL] vCPU:0 CR3:0x60ac3000,TIMEEVASION ntoskrnl.exe!NtClose
```

FIGURE 5.14: Trace Extraction of a Malware Process from VMM

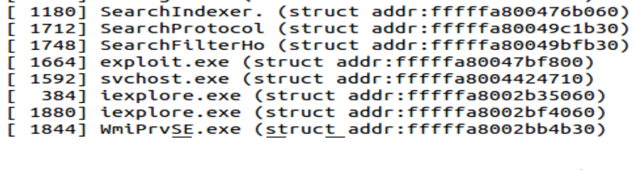
the time of intrusion profile creation. PET is used to collect the execution tracing which are stored in behavior logs. The sample output of program execution tracing is shown in Figure 5.14. Once behavior log is generated, system call parser is used to extract the sequence of system calls executed by each program. The new dataset (traces) is generated, named as `Dnew`.

Each of the trace is converted into SCDG by PSE sub-component of VAED, which represents the transition probabilities of each transition, stored in form of adjacency matrix (AM). Since it is difficult to find the discrimination power of a transition, all SCDG are processed and a <transition-value pair> is generated for each SCDG and stored in FTM. VAED applies IGR for selecting most suitable transitions (features). A total of 3955 features are obtained after generating FTM. However, we have selected 2767 transitions, after applying IGR in the FTM having good IGR value. Transitions with IGR value closer to 0 have not been considered. The selected features (transitions) are stored in OFTM. During the experimentation, its being observed that most of the processes terminates within 10 $\sim$ 12 minutes. Hence, we set the maximum trace extraction time to 12 minutes ($\sim$ 720sec). However, if some evasions are executing repeated transitions such as NtRaiseException$\rightarrow$ NtQueryInformationProcess (observed in time-based evasion) in loop, they are terminated forcefully when timer expires. If an attacker injects stalling code in a program which may cause the program to run more than 12 minutes. It means that it may perform the delaying operations for more than 12 minutes. Since VAED is based on the semantic information, VAED will detect that some system call transition is occurring abnormally than its usual occurrence. For example, a malware can invoke GetTickCount () system call (Windows TVM) multiple times in loop which is an abnormal behavior. It will try to access the same information (the time elapsed since the booting of machine) multiple times. The transition is invoked just once or twice to get some system related information. If a program is producing the same transitions in loop irrespective of how many iterations (directly relate to time of delay), its semantics will be captured by VAED.

5.4.4.2 Results and Discussion

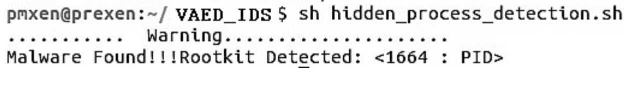
To evaluate the primary security check of VIMD, we executed some malicious and normal programs in TVM. Some of the suspicious processes are detected by the traditional anti-virus tool (`Avast` is considered) installed at TVM and hence, not found in memory of TVM. However, anti-virus tool failed to detect a process with PID 1664 which was found to be running with name `exploit.exe` and detected by PV component deployed at Dom0 of VMM as shown in Figure 5.15 and Figure 5.16. The presence of security processes namely `AvastSvc.exe` and `avastui.exe` is checked which were found to be running in TVM. However modern malwares such as `torpig` and `config` can disable them [101].

To perform a deep analysis of semantic behavior of programs running in TVM, other modules of VIMD are executed. The malicious programs are executed by PET, pre-processed by PSE and stored in OFTM by FTMG, as discussed before 5.4.4 (A). Now, the performance of different types of classifiers: probability based (Naive Bayes (NB) [160]), hyperplane based (Support Vector Machine (SVM) [179]), rule-based (Decision Tree (DT) C 4.5 [180] and Random Forest (RF) [168] and fusion of these classifiers using majority weighted voting scheme, is considered and compared. The fusion of classifiers using voting scheme is computationally less intensive and applicable to real time applications as discussed in Section 5.4.3. NB is found to be the fastest algorithm but NB assumes that all features are independent with each other which may not be true in malware detection where system call sequences depend on each other. SVM is very sensitive to parameter tuning which significantly changes with change in the dataset values. It is complex when compared to rule based tree classifiers. We have performed the



```
[ 1952] taskmgr.exe (struct addr:fffffa800499e610)
[ 1180] SearchIndexer. (struct addr:fffffa800476b060)
[ 1712] SearchProtocol (struct addr:fffffa80049c1b30)
[ 1748] SearchFilterHo (struct addr:fffffa80049bfb30)
[ 1664] exploit.exe (struct addr:fffffa80047bf800)
[ 1592] svchost.exe (struct addr:fffffa8004424710)
[  384] iexplore.exe (struct addr:fffffa8002b35060)
[ 1880] iexplore.exe (struct addr:fffffa8002bf4060)
[ 1844] WmiPrvSE.exe (struct addr:fffffa8002bb4b30)
```

FIGURE 5.15: Hidden Process detection at VMM (PID 1664)



```
pmxen@prexen:~/ VAED_IDS $ sh hidden_process_detection.sh
...........  Warning....................
Malware Found!!!Rootkit Detected: <1664 : PID>
```

FIGURE 5.16: Output of Process Validation

Table 5.12: Evaluation metrics and their description

Parameter	Description
True Positive (TP)	IDS detects the intrusive program execution as malicious
False Positive (FP)	IDS detects the normal execution of the system as malicious
True Negative (TN)	IDS detects the normal program execution as normal
False Negative (FN)	IDS detects the intrusive program execution as normal
TPR (Detection Rate, Recall, Sensitivity)	TP/(TP+FN); The proportion of correctly classified intrusions to the actual size of the attack class
FPR	FP/(TN+FP); The proportion of incorrectly classified intrusions to the actual size of the attack class
FNR	FN/(TP+FN);The proportion of incorrectly classified normal programs to the actual size of the normal class
TNR (specificity)	TNR=TN/FP+TN ; The proportion of correctly classified normal to the actual size of normal class
Accuracy	Acc=TP+TN/(TP+TN+FN+FP); Percentage of correct classifications over all test results

hyper-parameter to obtain the best parameter for SVM and set its parameters after tuning (kernel=rbf, C=100, gamma=1.0000). Tree based classifiers provide good performance for intrusion detection because of their implicit feature selection power and rule based classification. However, a single DT classifier does not provide good results due to over fitting problem of the training dataset. Hence, we have used ensemble of DT C4.5 classifier particularly Random Forest (RF) which takes random samples of dataset and trains different DT for each set. However, RF is sensitive to number of estimators (DT) values and requires tuning of number of estimator parameter. We have considered a range from 2-170 estimators and applied bagging method during parameter tuning as shown in Figure 5.17. To predict the number of estimators, Out of Bag (OOB) error is calculated in each iteration. Out of Bag (OOB) error is the average error for each instance (Xi, Yi) predicted by n classifiers trained over a bootstrap sample that does not contain (Xi, Yi) [185]. In order to tune the RF parameter, a curve between Out-of-Bag (OOB) error and number of estimators (DTs) is plotted. We have to find the number of estimators (decision trees) for which OOB error is minimum. OOB highly affects the classifiers performance and directly relates to generalization error. We found that if number of estimators=80, OOB error is minimum and hence, we choose the same value during classification. OOB fluctuates from range 2-80 (number of estimators) and after 80 it remains nearly same. RF builds the trees in O(M(nmlogn)) where M is the number of trees initialized while running the algorithm and n is the number of instances and m is the number of features. Random Forest runs on an average time even for huge datasets. Now, training RF with IGR as feature selection criteria having total 80 estimators, the least OOB error is produced. The standard evaluation metrics are described in Table 5.12 and have been considered for evaluating the

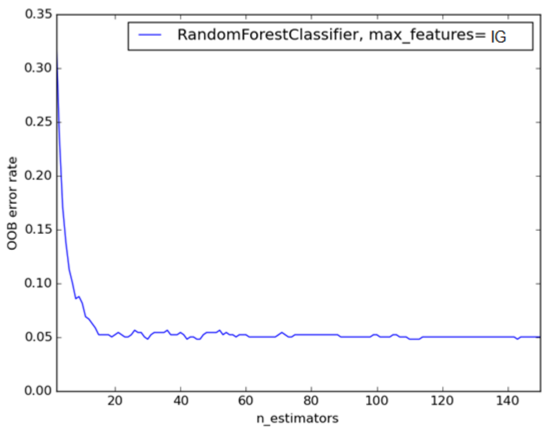


FIGURE 5.17: OOB error vs No. of Estimators

approach.

Fusion of classifiers (ensemble learning) is proved to achieve a better accuracy. The weighted voting scheme have been applied with the discussed classifiers. RF is also ensemble based learning approach. However, it considers multiple instances of same classifier (DT). The motivation behind using fusion of diverse classifiers, is to combine the conceptually different classifiers based on weighted voting rule scheme. Here, if a classifier is very confident (provides good prediction probability for classification), it is assigned more weight over other classifiers. We have performed parameter tuning of weights in which all four classifiers are executed and their results are combined for different combination of weight values in rnage (1, 4). The best combination weight is chosen at the time of training (W1(SVM): 3, W2 (NB): 1, W3(DT):2, W4 (RF):3) which provides the best results. We have applied k-fold cross validation with classifiers to train/test them for evasive and benign samples. We have taken k=10. It means that the dataset is divided into 10 portions. Initially first nine portions are used for training and last portion is used for testing the classifier. In the next iteration, next nine portions are considered for training and one portion (which is not considered for testing in previous iterations) is used for testing. The process is repeated 10 times (folds).

We found that each of the classifier is providing different results for each of the evasive classes which is because of the difference in attack characteristics. For exception-based evasions, NB is providing a poor accuracy of 94.166% with 1.9093% FPR and 14.917% FNR. The false

Table 5.13: Classification Results for Exception based Evasions

Classifier	Acc	TPR	TNR	FPR	FNR
SVM	97.021	94.4011	98.568	1.43198	5.6238
DT	96	93.370	97.136	2.8639	6.6298
NB	94.166	85.0828	98.0906	1.9093	14.917
RF	97.333	94.475	98.568	1.0432	5.5248
Fusion	**97.666**	**95.027**	**98.8066**	**1.1933**	**4.9723**

Table 5.14: Classification Results for Process Feature based Evasions

Classifier	Accuracy	TPR	TNR	FPR	FNR
SVM	97.488	89.987	99.1576	0.64239	9.02255
DT	96.166	94.7368	96.5738	3.4261	5.2631
NB	94.1666	93.2330	94.432	5.5674	6.67669
RF	97.5	90.977	99.357	0.59642	9.2122
Fusion	**98.833**	**95.488**	**99.7858**	**0.2141**	**4.511**

alarms are high and detection rate is also low (85.0828%) which is not acceptable. The details are shown in Table 5.13. DT is providing better results when compared to NB. It provides 96.0% accuracy with comparatively low false alarms (2.8639% FPR and 6.6298% FNR). The detection rate is improved to 93.370% and TNR (specificity) comes out to be 98.0906% which is fairly acceptable. DT provides a rule based classification and its performance can be improved with reduced classification error using ensemble of DT (RF). RF classifier combines the idea of bagging and random selection of features. It consists of many decision trees. In bagging, successive trees are independent to each other and constructed using a bootstrap sample (training subsets originated from random sampling with replacement of training set). At the end, majority vote or average is taken for predictions. An unseen sample is passed to each of the trained tree. Output can be defined as the average of the output of the generated trees. For exception based evasions, RF provides 97.333% accuracy with 1.43198% FPR and 5.5248% FNR. The detection rate (TPR) is also better than NB and DT which is 94.475%. SVM found to perform comparable to RF. The fusion of classifiers is providing the highest accuracy of 97.666% with low false alarms (1.1933% FPR and 4.9723%) with highest specificity 98.8066%.

For Processor-Feature based evasions, accuracy of DT is 2% higher than accuracy of NB (94.166%) as shown in Table 5.14. DT is observed to provide good true alarms for benign and malware class. The detection rate is 94.7368% with 3.4261% FPR. NB provides the poor detection

Table 5.15: Classification Results for Time based Evasions

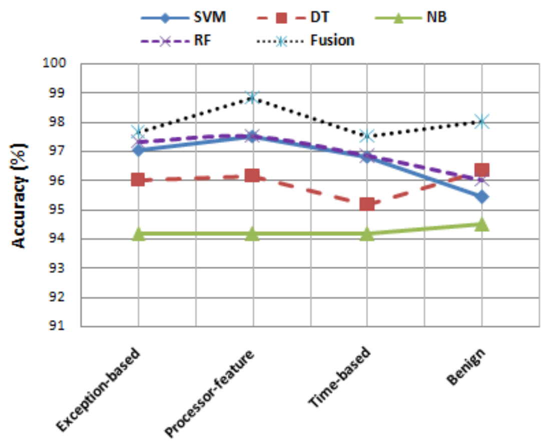
Classifier	Acc	TPR	TNR	FPR	FNR
SVM	96.813	93.216	97.2820	2.0179	6.4935
DT	95.166	89.6104	97.085	2.9147	10.3896
NB	94.1666	93.5064	94.39461	5.60538	6.49350
RF	96.833	93.506	97.9820	1.9910	6.393
Fusion	**97.5**	**96.1038**	**97.9820**	**2.0179**	**3.8961**

Table 5.16: Classification Results for Benign Processes

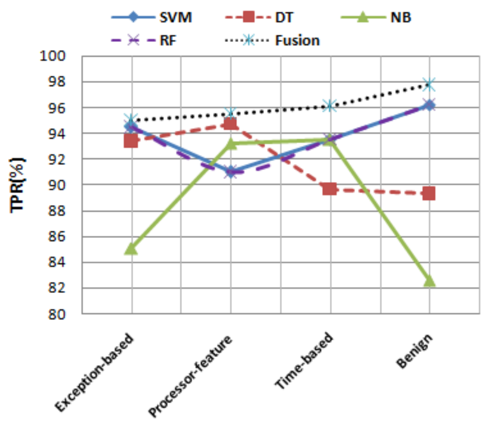
Classifier	Acc	TPR	TNR	FPR	FNR
SVM	95.99	96.011	95.14017	4.0598	3.7878
DT	96.333	89.2906	98.2905	1.7094	10.6060
NB	94.5	82.5757	97.8632	2.1367	17.4242
RF	96	96.2121	95.940	3.9598	3.087
Fusion	**98**	**97.7272**	**98.07692**	**1.9230**	**2.2727**

rate of 93.2330% with high FPR of 5.5674%. In this case also, more complex classifiers (considered RF and SVM) are providing nearly similar accuracy of 97.500% and 97.488% respectively. The false alarms for RF are slightly lower than SVM and its FPR is below 1% and FNR is around $\sim$ 9%. They are outperforming the DT and NB and providing good detection results. The fusion of all these classifiers again found to outperform the existing results. It achieves highest accuracy of 98.833% with very low false alarms among other classifiers (0.2 $\sim$ 4.5). The specificity is observed to be 99.78% with highest detection rate of 95.488% which is fairly acceptable for detecting evasive samples which tend to hide their behavior. The metrics can be compared from Figure 5.18 where each sub figure shows the performance of different classifiers for detecting different classes.

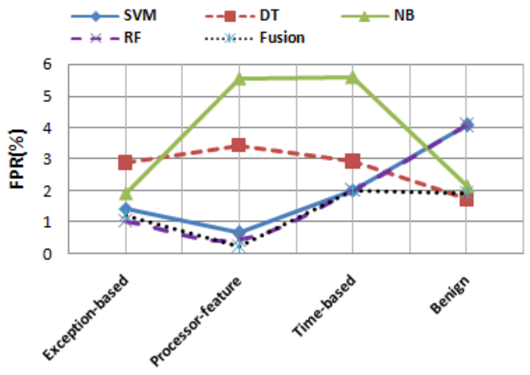
For time based evasions, NB is providing the 94.1666% accuracy with false alarms in the range of 5.5$\sim$6.5. However, DT failed to provide good detection rate which is below 90%. However, the probability based classifier achieves $\sim$ 93% detection in this case. Time based evasions have similarity with normal processes and hence, the FNR is higher in most cases. All the classifiers, other than fusion of classifiers, are achieving the detection rate $<$ 94%. The fusion achieves the detection rate of 96% with 2.0179% FPR and 3.8961% FNR which is highest among all cases as shown in Table 5.15. The specificity is 97.9820%. The results for classification of benign process are shown in Table 5.16 which depicts that fusion of classifiers provides the highest accuracy of 98% for correctly classifying the normal processes. The accuracy



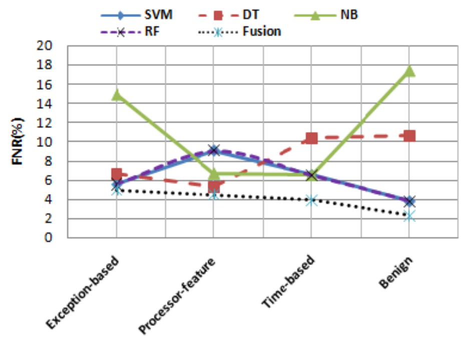
(A) Accuracy for different classes



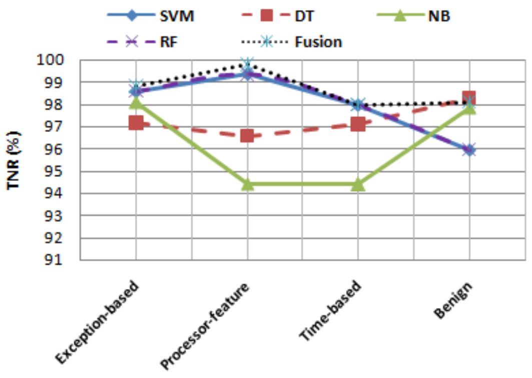
(B) TPR for different classes



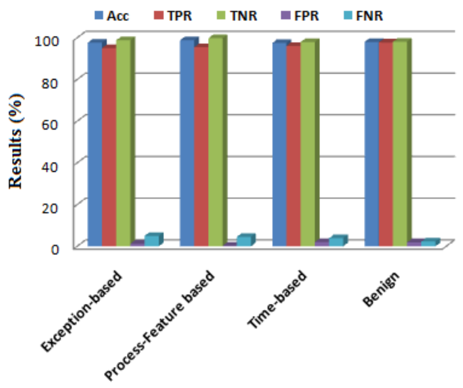
(C) FPR for different classes



(D) FNR for different classes



(E) TNR for different classes



(F) Performance results for fusion of classifiers

FIGURE 5.18: Performance Metrices of VAED for Evasion-based malware Detection for `Dnew` using different classifiers

of fusion is higher in compared to other classes as shown in Figure 5.18a and TNR is lower as shown in Figure 5.18e which is important in security applications where users are not interested to face trouble in executing the benign applications.

Table 5.17: Performance Overhead: Processing Time per Sample (seconds)

Process Monitoring	
Parameters	Time (best-worst) (s)
Introspection time	60-700
Process Validation	0.06 -1.453
Trace Pre-Processing	0.0423 -90.324
Detection Time	1.21 - 1.312
Total	61.3123 - 793.08

The results for other classifiers are also fine and accuracy is in the range of 94% $\sim$ 96%. NB and DT are providing much FNR $>$ 10% in this case and are not a good choice for selection again. We plotted a bar chart to compare the performance results of classifier fusion for different classes as shown in Figure 5.18f. We can infer from Figure 5.18d and 5.18c that fusion of classifiers provides the low false alarms in compared to other classifiers. The detection rate also goes higher for detecting different types of evasions as shown in Figure 5.18b. The performance of VAED is acceptable for detecting evasion based malwares.

The overhead associated with the approach per sample have been discussed as shown in Table 5.17. The system overhead directly depends on the time taken for program trace analysis. This time is less for the shorter traces and more for longer traces. All the traces are extracted at a time within fixed time frame. We have considered 60 s of time as a best case, assuming that traces terminate within 60 seconds. If a trace terminates with 700 seconds, it would be worst case considered which is a very rare case during observed during experimentation. The maximum time incurred depends on the largest trace. Program-trace analysis time depends on following: Introspection time (60s-600s), process validation time (observed negligible 0.06s - 1.453s), feature vector construction time (0.0423 s - 90.324 s) and detection time (observed negligible 1.212s - 1.312s). Therefore, in both worst and best scenarios, the time to process a sample was captured as 61.3123s - 793.08s respectively which is the total time execution of all phases. The maximum overhead incurred in processing a sample was observed as 793.08s as shown in Table 5.17. If short processes are running in the memory which terminate frequently, the overhead will be less. The overall time depends on the number of applications running in the TVM and the trace length. The additional time is taken for offline analysis to create intrusion profiles of malware which is a one time effort. The malware trace extraction takes a long time (say a few weeks). OFTM creation time (observed 164.05672s) with feature selection (observed $\sim$ 2.21s) for all intrusive samples and training time (observed 1.64s).

The executions of all detection components of VIMD with VAED as core detection mechanism have been successfully evaluated. VAED is found to perform well to detect the evasive malware attacks and provides the detection accuracy of 97.50%-98.8333% with 0.2141%-2.0179% false positives in detecting evasive malware. The detection mechanism of VAED is found to be effective for detecting evasive malware at VMM-layer.

5.5 Comparison with Existing VMM based IDS

We have compared VIMD with other intrusion detection frameworks in a virtualization environment: Maitland [91], Xenini-IDS [69] and ShadowContext [93]. The various parameters considered for comparison are shown in Table 5.18. VIMD, Xenini-IDS and ShadowContext are based on dynamic analysis and VMI approaches. The main objective behind VIMD, Maitland and Xenini-IDS is attack detection. ShadowContext prevents the modification in privileged system calls by providing a system call redirection based VMI approach. This approach forces the monitored system calls to execute in a secure location. The attack prevention of ShadowContext may not be a very good choice as it imposes a significant overhead into the system. This is because that the monitored system calls of programs of all the TVMs are executed in Dom0 memory of the hypervisor, protecting them, from in-guest malware attacks. Xenini-IDS uses an interrupt handling technique in which interrupts (0x80) are first diverted to the hypervisor (Xen is patched with Xenini which is a security module running at VMM), which further divert control to XenIDS (running at Dom0) for performing malware detection. Maitland uses the guest OS hooks based VMI technique where some security modules execute as kernel hooks in guest OS. These security modules further send the VM information to analysis module, running at Dom0. VIMD employs a breakpoint injection based VMI technique for trapping syscall traces and detecting in-guest malware from Dom0 of the hypervisor. Unlike other VMI approaches, breakpoint injection technique is much flexible to be implemented at different guest-OS and

Table 5.18: Comparison of Proposed technique with existing work

Parameter	**VIMD**	**Maitland [91]**	**Xenini-IDS [69]**	**ShadowContext [93]**
Technique	VMI with Dynamic Analysis	VMI	VMI with Dynamic Analysis	VMI with Dynamic Analysis
Motive	Attack Detection	Attack Detection	Attack Detection	Attack Prevention
VMI Technique	Break Point Injection	Guest OS hooks	Interrupt Handling	System Call Redirection
Placement of IDS	VMM (Dom0)	VM and Dom0	VMM and Dom0	VM and Dom0
Feature Extraction	BonG (VMGuard) SCDG (VAED)	Memory Writes (String)	System Call Sequence (String)	System Call (String)
Ordering of System Call	Considered	NA	Considered	Considered
Attack Resistantance	High	Medium	Medium	Medium
Feature Selection	TF-IDF (VMGuard) IGR(VAED)	NA	NA	NA
System Model	Freq-based(VMGuard) Prob-based(VAED)	NA	String-based	NA
Storage Requirement	Medium	More for signature system	High	Low
Machine Learning	Applied	NA	NA	NA
Adaptability	More	Lesser than A	Lesser than A	Lesser than A
Process Validation	Considered	Not Considered	Not Considered	Not Considered
Hidden Process Detection	Considered	Not Considered	Not Considered	Not Considered
Levels of Security Check	Two	One	One	One
IDS Subversion	Difficult	Moderate	Difficult	Moderate
Robustness of System	High	Medium	Medium	Medium
Detection Rate	94%-100% (VMGuard ,UNM), 97.50%-98.8333% (VAED, Dnew)	NA	100% (UNM sendmail)	NA
Hypervisor dependability	dependant	dependent	dependant	dependant

do not require additional security modules at guest OS kernel, making it suitable for security applications for cloud environment.

VIMD have been implemented with two feature extraction approaches: BonG (when VMGuard is used as core detection mechanism) and SCDG (when VAED is used as core detection mechanism) at the VMM-layer. VIMD with core detection mechanism as VAED, supports the probability model and constructs System Call Dependency Graph (SCDG) for each trace which is built by using Markov Chain principle. The ordering of system calls and transition probability distribution of system calls both are maintained by VAED. Maitland does have its own detection mechanism, however a simple pattern matching approach is used by author for demonstration. Xenini-IDS considers the ordering of system calls. However, since it employs string-based features, it is prone to string based attacks in which legitimate codes are inserted in the malware process to increase the number of matches compared to mismatches. ShadowContext fetches the details of selected system calls of monitored programs and executes them at Dom0. It maintains their ordering in the execution sequence of a program. However, a piece of code is injected in a dummy monitored process (used for communication with Dom0) which runs at the monitored machine. The security of this code is very important aspect here. In VAED detection mechanism of VIMD, the feature vectors contain continuous values, it is difficult to bypass the security system by performing string or frequency manipulation.

Unlike all other approaches, VIMD uses feature selection method for each of the detection mechanism: TF-IDF (in VMGuard) and IGR (in VAED) which improves the discriminating power of system call sequences/transitions and improves the performance of detection engine.

In VIMD, none of the security code runs at the monitored machine, making it more attack resistant and secure from in-guest malware attacks. The attack resistant of rest of the approaches (Maitland and ShadowContext) is moderate, as security modules are distributed among monitoring TVM and Dom0 VM, requiring the security of their own modules, running at TVM. Unlike VIMD, Xenini-IDS uses the system call sequences (string patterns) for monitoring. It does not incorporate the semantics of the sequences in different traces as incorporated by VIMD, making it less resistant to string (system call) manipulation attacks. Hence, Xenini-IDS achieves moderate-level of attack resistance.

The storage requirement of VIMD is different for each detection mechanism. In case of VMGuard, it stores the frequency count of appropriate unique n-grams selected by applying the TF-IDF algorithm. This reduces the storage requirement to a greater extent when compared to existing schemes ISCS [84], BoS [67]) which stores all possible system call sequences/system calls of monitored programs. If a total of m, n-grams are selected, then O(km) storage will be required, where k is a constant representing the number of bytes required to store the count of an n-gram. In case of VAED, it stores the probability distribution of appropriate unique system call transitions by applying IGR. The storage requirement is reduced to a greater extent than existing approaches. If there are total n transitions are selected, then O(cm) storage will be required where c is a constant representing the byte required to store a probability value of transition. Maitland requires more storage than VIMD if various pattern matching rules are used to find any abnormality in the extracted memory patterns. Xenini-IDS also requires more storage than VIMD in storing all possible system call patterns of all normal traces which directly depends on the total number of execution traces. ShadowContext does not maintain any baseline profile. However, some context registers and in-guest buffers are used at the time of monitoring.

VIMD improves the generalization power of the detection engine using statistical learning techniques (machine learning approaches), which improves the ability of the detection engine to detect similar unseen attack processes. The other discussed approaches lack the statistical learning based approach, and hence, limit their adaptability to learn the behavior of newly discovered attacks.

VIMD provisions two-levels of security check, unlike other approaches which have just one. The primary check provides a basic security check to detect any hidden processes and also checks the presence of running processes of traditional security-tool at the TVM (process validation), which makes it more robust against stealth attacks. The secondary security check performs a detailed behavioral analysis of processes, as some malicious processes may attach themselves to normal programs and keep running in the system.

VIMD, Xenini-IDS and ShadowContext have better robustness than Maitland approaches because of completely out-of-the-VM implementation of the monitoring agent. However, unlike VIMD, rest approaches do not check the presence of VM rootkits and do not validate whether a tenants security tool (such as an Anti Virus) is running correctly. Moreover, ShadowContext can behave abnormally if the security manager fails to design the system call redirection properly. A redirected system call can produce unexpected results and can even crash the guest OS.

VIMD have been validated with the UNM dataset and Evasive malware dataset. The UNM dataset is a standard dataset that has been widely used by researchers in validating their techniques. It is difficult to replicate the results from self-generated dataset which is not publicly available. VIMD achieves accuracy from 94%-100% in detecting anomalies using VMGuard detection approach. It achieves an accuracy of 97.50%-98.8333% for detecting evasive malwares using VAED. Maitland is not tested for any attack dataset. Xenini-IDS achieves 100% accuracy for UNM sendmail dataset. However, it uses traditional STIDE detection mechanism [115], which is a very older approach and based on the principle of collecting all normal system call sequences of monitored programs. Any deviation will signal to abnormal behavior. Hence, it is not suitable in cloud environment where user's behavior keeps on evolving as it will produce more false alerts for evolving new behavior. ShadowContext is a prevention strategy and hence, it has not been tested with a attack dataset. VMI based approaches are hypervisor dependent. Xenini-IDS is implemented on Xen 4.0.1 and Shadow-Context on KVM (version unspecified). A key advantage of VIMD is that, being an introspection-based application, it runs as a security application at Dom0 of the hypervisor and provides efficient intrusion detection schemes at the VMM-layer in cloud.

5.6 Conclusion

A VM introspection based malware detection system has been proposed for detecting intrusions at the virtualization layer in cloud environment. Our work addresses four main aspects: (i) Placement of the security tool at a trusted location; (ii) Provision of primary security check to perform process validation and program execution tracing at VMM (iii) Provision of secondary security check by providing two different system call behavior analysis approaches at hypervisor (iv) Use of a machine learning techniques to learn about attack patterns. The first technique for behavior analysis is VMGuard which extracts the semantics for attacks using the TF-IDF integrated BonG approach and have been found to perform well for program modification attacks. The second technique for behavior analysis is VAED which is mainly designed to detect specific set of malwares called evasive malware. It employs SCDG analysis for malware detection. It not only preserves the ordering of system calls in form of system call transitions but also considers frequency and probability distribution between each pair of system call which makes it suitable to detect complex behavior of evasive malware. In both these approaches, learning module is applied to create a generic behavior of malware samples. So, even if there is a small variation in malware code, its semantic information can be captured and detected by security tool. In the proposed security architecture, VIMD is placed at Dom0 of a VMM, and TVMs are introspected using VMI libraries. The run-time behavior of programs is extracted. Machine learning approaches are applied to generalize malware behavior and improve the adaptability of the system. VIMD is suitable for a virtualization based cloud environment. A cloud administrator can control and monitor the security tool from Dom0 of VMM, which helps to prevent IDS subversion from nefarious tenants. VIMD achieves accuracy from 94%-100% in detecting anomalies using VMGuard detection approach. It achieves an accuracy of 97.50%-98.8333% for detecting evasive malwares using VAED.

Chapter 6

Malicious Network Packet Detection (MNPD)

This chapter describes the design and implementation of 'Malicious Network Packet Detection' (MNPD), one of the sub IDS instance of CloudHedge. MNPD monitors the network traffic at VMM-layer and Network-layer in cloud. It supports the network introspection capability along with the network behavior analysis to deal with intrusions in cloud. MNPD sub IDS instances are distributed at each privileged domain of Virtual Machine Monitor (VMM) and Cloud Network Server, running outside the Tenant Virtual Machines (TVMs). Each instance is configured and controlled by cloud administrator. The security design of MNPD is described with the explanation about various detection components in detail.

6.1 Introduction

A number of attack incidents are reported by IT companies or organizations depending heavily on IT every year. Cisco Annual Security report [186] mentioned that spams related to the Boston Marathon bombing made up of 40% of all spam messages delivered worldwide. European Network and Information Security Agency (ENISA) [35] reported that Dropbox was attacked by Distributed Denial of Service (DDoS) and suffered a substantial loss of service for more than 15 hours. Amazon cloud was infested with DDoS botnets [187]. Researchers have proposed various defensive solutions. The defensive mechanisms are mainly categorized into two types: traditional approaches (like signature matching and machine learning etc.) and virtualization based approaches (like VM introspection). The survey of detection mechanisms has been discussed in Chapter 2. Security researchers have applied these mechanisms to detect the network intrusions in cloud. However, there are some limitations associated with the existing security solutions as discussed in detail in Chapter 2. They are briefly summarized below.

Some researchers have proposed the use of traditional intrusion detection approaches at the TVM-layer for detecting network intrusions in cloud [74][82][90][139][188]. These approaches share the TVM resources which are being shared by monitoring programs and can be directly accessed by attackers. The existing network intrusion detection system (NIDS) proposals [71][83][87] where the monitoring is done at the cloud servers or node controllers, are not efficient. These approaches fail to detect VM attacks which are targeted from one TVM to another TVM sharing the same physical server as the tenant's virtual traffic never passes through any physical interface. Some of the security proposals apply signature matching technique [141][149] at the VMM-layer which may impose significant overhead into the system as discussed in Chapter 2. Infact, most of the works [126] [90] have been validated using datasets like KDD99 or NSL-KDD99 [150]. These are comparatively old and have been criticized by other researchers for not representing the state of the art network features and their statistics [189].

Some of the other recent approaches (eg. Maitland[91], FMA[190]) support introspection. However, they are mainly designed for VM memory analysis and not intended for traffic analysis applications. The recent work [126] (validated with KDD99) is based on network traffic monitoring at VMM-layer and does not incorporate the introspection features at the hypervisor. In fact, in all the existing IDS techniques, as discussed before, the detection becomes more complex for the forged IP/MAC addresses. These attacks are difficult to detect at the VMM-layer. As the actual VM IP information of a packet is lost when a packet crosses the virtual bridge and reaches the physical interface. Moreover, the detection of intrusions at the Network-layer as as important as the detection at other layers.

We propose a network behavior monitoring system, called as 'Malicious Network Packet Detection (MNPD)' to provide security from network intrusions in cloud. MNPD provides two-levels of security checks. One instance of MNPD is deployed at the Network-layer of Cloud Network

Server (CNS), providing primary security check from network attacks at centralized server. However, not all tenant traffic passes through the CNS, specially communication between co-located tenants in the same server. Hence, another instance of MNPD is deployed at privileged domain (Dom0) of VMM-layer of Cloud Compute Server (CCoS); providing secondary security check. Xen VMM [10] is considered for the implementation of MNPD which manages all the unprivileged domain (DomU) of hypervisor. MNPD is configured to listen on virtual network interfaces (VNIs) on both servers. MNPD provides VM introspection to gain the VM related information using open source tools such as Libvirt [77], XenStore [191], dnsmasq server [192] from Dom0 of hypervisor. The information is later used to perform traffic validation at VMM-layer of CCoS to detect spoofing attacks. However, an attack can be targeted using correct Source Internet Protocol (IP) and/or Media Access Control (MAC) address. Hence, MNPD performs behavior analysis of traffic at both Network-layer and VMM-layer by using machine learning approaches particularly Random Forest classifier (RF) with ensemble of feature selection method. MNPD employs two popular feature selection approaches i.e. Chi Square [193] and Recursive Feature Elimination (RFE) [194] methods in parallel and combines their results to transform the original dataset into new dataset. This reduces the dimension of dataset and removes the less relevant features, which improves the classifiers' performance. The motive of feature union is to reduce the biasness of one feature selection method towards the heterogeneous dataset which contains different datatypes. On detection of suspicious network packets, alert signals are generated and sent to the cloud administrator for further action. MNPD is well suited to provide the third line-of-defense from network intrusion in cloud.

6.2 Network Introspection at Xen

In this section, we discuss the information related to various domains at Xen, Xen-API tools, database, administrative interface, networking drivers and Dynamic Host Configuration Protocol (DHCP) server which are used by MNPD for VM network introspection.

Dom0 is the trusted privileged domain in Xen that starts first on boot. It manages all other untrusted domains of tenants called as DomU domains. Dom0 is isolated and secure from access of tenant users. There are two interfaces provided by Xen. One interface is used by tools (Xen-API/Xen management API) and the other is used by guests (Xen systemcall/hypercall API) [195]. Xen API provides the bridge between the low-level daemons and userspace applications. The request from userspace application is parsed and communicated to the Dom0 kernel. It then serves the request using the management functions. In the implementation of MNDP with Xen as VMM, libvirt library [77] is used to provide Xen API and facilitates the management functionalities at hypervisor. This library is widely used in the development of cloud based solutions by companies like Oracle and SUSE in their OpenStack-based Cloud [196]. Libvirt is an open source management library in virtualization environment and provides support for many APIs: storage management, networking and domain management to perform management related tasks. For example, libvirt supports APIs to do domain management such as provision, modify, create, control, migrate, stop etc. It includes an API library, a daemon (`libvirtd`), and a command line utility (`virsh`) to perform management related task. Dom0 maintains a database named as XenStore which maintains the configuration information of different domains which is shared between them. The domains read and write in the database to communicate with other domains. A daemon named `xenstored` runs in the Dom0 of Xen to provide backend storage to XenStore. In Dom0, a special daemon `xend` runs and provides the administrative interface to hypervisor which is used by cloud administrator to define policies.

Dom0 is provided with device driver domain. It runs the native network interface card (NIC) driver. As DomU machines cannot directly access the physical NIC, Xen provides two part virtual NIC driver to them. The first part is named as `FrontendDriver` which is installed on DomU and second part is named as `BackendDriver` which is installed on Dom0. `FrontDrivers` and `BackendDriver` communicates with each other using XenStore, XenBus and event channels [191].

Xen provides ethernet bridge (`virbr0`) which connects the physical NIC to virtual NICs as shown in Figure 6.1. The bridge multiplexes and demultiplexes the traffic between physical NIC and each virtual NIC. When a TVM wants to transfer data to internet, it sends the request to `FronendDriver` which forwards the request to `BackendDriver`. All requests are queued in the driver domain of Dom0 and forwarded to actual physical device. The `FrontendDrivers` are named as `ethN` and

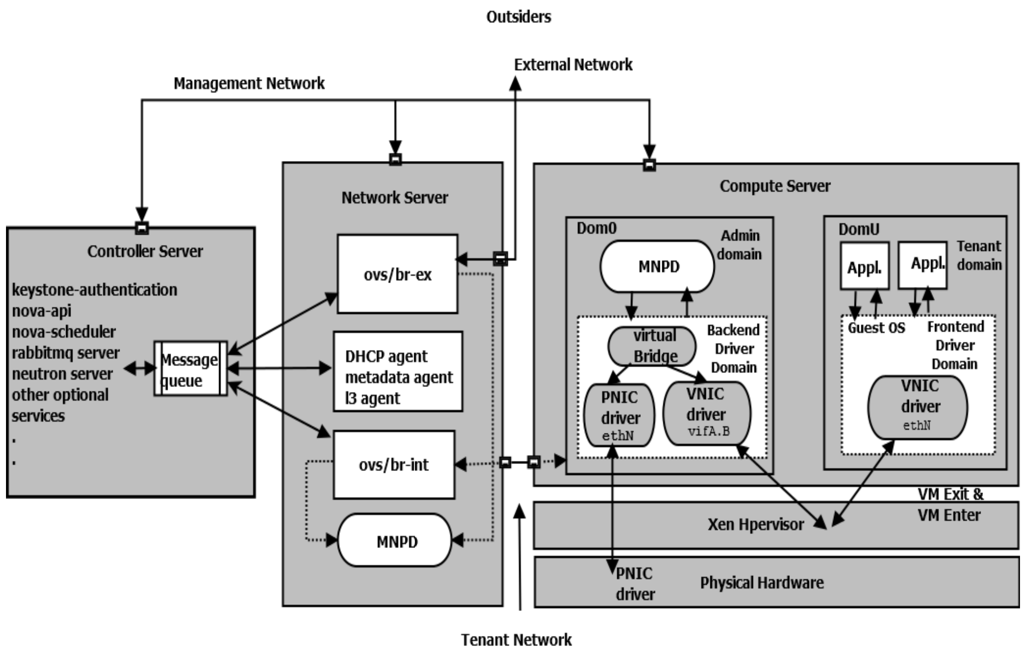


FIGURE 6.1: Security architecture for deployment of MNPD instances in cloud environment

`BackendDrivers` are named as `vif.DomID.DevID` where DomID is domain ID of VM and DevID is interface ID. For example, vif5.0 can be interpreted as virtual network interface of VM having domain ID 5 and interface ID 0.

The OpenStack, a popular cloud management software uses `dnsmasq` service at Dom0 of Compute Servers to assign the private IP addresses to each TVM [192]. Hence, for implementing MNPD, `dnsmasq` service is used for setting up Dynamic Host Configuration Protocol (DHCP) Server at Dom0 of hyperivor. The libvirt supports this light weighted service which provides network infrastructure for small network (such as network for co-located tenant machines): Domain Name Server (DNS), DHCP, route advertisement and network boot. The DHCP server of `Dnsmasq` supports both static and dynamic leases, multiple networks and IP address ranges. Cloud administrator can configure the service to assign the range of IP addresses, gateway, DNS-server, and subnetmask etc using the `dnsmasq.conf` configuration file (`/etc/dnsmasq.conf`).

6.3 MNPD: Security Design

The proposed Malicious Network Packet Detection (MNPD) as a sub-IDS instance of CloudHedge, is designed to detect attacks against

TVMs in cloud environment. MNPD provides the third-line of defense from attacks, having two-levels of security check at CNS and CCoS respectively as discussed before in Section 6.1. The MNPD approach is concerned mainly with the analysis of network traffic traces. It provides good performance for network attack detection. However, it does not promise to achieve the same accuracy for low-frequency attacks such as malware, rootkits, etc. The network connection statistics of these attacks are similar to normal network connections. The approach to detect such type of attacks is proposed in our earlier works (discussed in Chapter 4 and Chapter 5) which are based on system call analysis.

The following assumptions are made in the security architecture: A Cloud Service Provider (CSP) is a faithful and reputed organization and VMM platform is trusted. MNPD has privilege to access the application specific information of monitored VMs. This privacy concern is clarified between CSP and tenant users at the time of registration in form of Service Level Agreement (SLA). For example, Google says that it reserves the right to review the tenant's applications and data. Users sign this agreement at the time of sign up/registration [163] .

The main objectives of the MNPD are as follows:

1. To perform the behavioral analysis of network traffic at Network-layer of CNS to provide primary security defense from attacks.
2. To detect the spoofing attacks (both IP and MAC) originated from TVMs and also to perform the behavioral analysis of network traffic at VMM-layer of CCoS to provide secondary security defense from attacks.

The detection of network attacks such as, TCP-SYN flooding, ICMP flooding, UDP flooding and scanning etc., is essential at primary stage at CNS. These attacks slow down the performance of target VM by making its resources unnecessarily busy. Spoofing attacks from TVM to another TVM, make the detection of network attacks more difficult to achieve. Hence, whenever a packet is generated by a TVM, its IP and MAC are validated at hypervisor and only non-spoofed packets are passed to behavior analyzer at CCoS for further analysis for detection of network attacks.

At a high-level, there are four main detection components of MNPD as follows: Behavior Capturing Engine (BCE), Traffic Validator (Trval), Network Behavior Analyzer (NBA) and Alert & Log Generator (ALG).

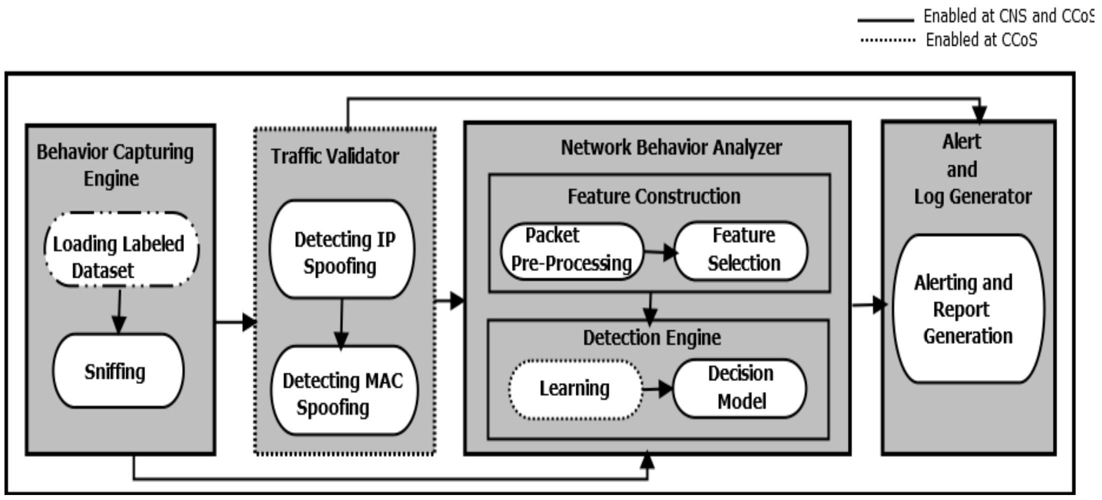


FIGURE 6.2: Security architecture of MNPD

The various detection components of MNPD are shown in Figure 6.2. BCE sniffs the packets and generates logs. It also provides the attack dataset at the time of learning. TrVal validates the TVM traffic for correct IP and MAC address. Only legitimate traffic is allowed to reach the destination after behavior analysis at Dom0. NBA learns the attack patterns in learning phase and detects any suspicious network pattern in detection phase. ALG generates the alerts and logs if suspicious activity is detected by NBA detection component. The logs are shared with cloud administrator. Only legitimate packets are allowed to pass from physical interface of CCoS and forwarded to other servers (CNS/other CCoS). TrVal ensures that virtual traffic reaching to CNS or other CCoS, has correct virual IP/virtual MAC. Therefore, TrVal is activated at CCoS only whereas all other detection components are activated at both the servers. The execution flow of various security functions of MNPD is shown in Figure 6.3. A detailed description of each of the detection components of MNPD is discussed in subsequent subsections.

6.3.1 Behavior Capturing Engine

A Behavior Capturing Engine (BCE) sniffs the packets ingress or outgress on the monitored TVM. MNPD uses tcpdump [197] for sniffing and logging the packets. At CNS, MNPD is deployed at the host OS server and is configured to capture the packets passing through Open

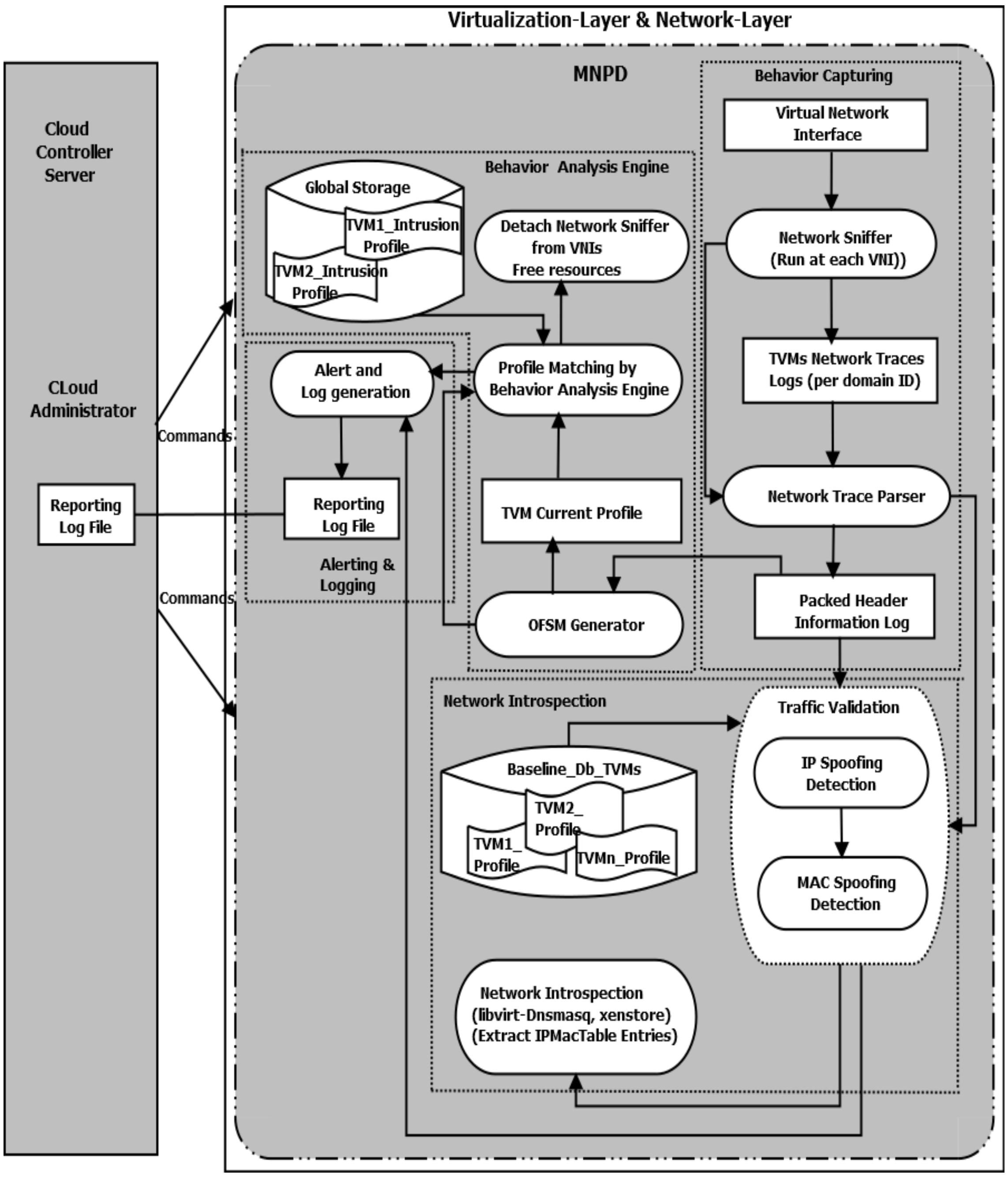


FIGURE 6.3: Execution flow of MNPD Sub IDS security functions

Virtual Switch (OVS). OVS plug-in supports two types of bridge interfaces namely br-int and br-ex. A br-int is an integration bridge which connects the VMs with each other. The internal network traffic flows through this bridge. A br-ex is an external bridge which connects the VMs to external world or vice versa. A br-ex bridge is connected to physical interface of the CNS as shown in Figure 6.1 (refer Section 6.2). At CCoS, the behavior is captured at the VNICs of the monitored machine. Packet Sniffing is done in parallel for each of the monitored VM by listening on the corresponding Virtual Network Interface (VNI) of machine and VM profiles are created. Each VM profile is identified by a particular domain name and domain ID, set by administrator and hypervisor respectively. Each profile represents the behavior statistics of network connections which are stored in .pcap file. Another important task of BCE is to provide the attack traffic records to TrVal and NBA for validating and learning the behavior of attacks.

6.3.2 Traffic Validator

Intrusion Detection System deployed at the TVM being monitored can be compromized and full control can be obtained by an attacker. Hence, TVM-layer security mechanisms which ensure that a tenant is generating only legitimate packets, cannot be fully trusted. In fact, the detection process will become cumbersome, if the compromised TVM floods the other TVMs with spoofed source addresses. Traditional IDSes do not provide the introspection facility to detect such network attacks from hypervisor. At the VMM-layer, Traffic Validator (TrVal) provides the traffic validation functionality to ensure that TVMs are generating traffic from a correct Source IP and MAC address. It takes the input from BCE and redirects the non-spoofed packets to NBA for detailed analysis as attacks can also be generated from a legitimate source address. The proposed Algorithm 4 shows how TrVal validates the traffic from outside the VM at Cloud Compute Server (CCoS).

VM traffic passes through VNI and then through virtual bridge before reaching an actual physical interface. Once the traffic reaches actual physical interface, its information about the actual VM IP is lost. Hence, MNPD captures the packets at VNI itself and detects the spoofing attack by TrVal component. Each VNI (`vif.DomID.DevID`) is associated with a DomID as discussed in Section 6.2. Hence, it is

Algorithm 4: Proposed algorithm for Detecting IP and MAC Spoofing at Hypervisor

```
 Global Baseline_Db ← {}, Test_Db ← {}, Flag ← {}
 MACIPTable ← /var/lib/libvirt/dnsmasq/virbr0.statusOutPut : AlertandLogs
Function MNPD_SpoofingDetection()
    BaselineDB_Generation();
    for each VNI.DomID.DevID in Xen do
        PacketCapturing_VNI.DomID(VNI.DomID.DevID);
        //We are calling it in parallel for each VNI
    end
return
Function BaselineDB_Generation()
    Start dnsmaqservice of libvirt and Configure it for VMs network setting;
    for all tenant VMs under monitoring and in running mode do
        Domainname.value,DomainID.value=getDomainnameID(); Real.IP,
        Real.MAC=ExtractReadIPMAC(MACIPTable);
        Baseline_Db=CreateBaselineDb(Domainname.value, DomainID.value, Real.IP, Real.MAC);
    end
return
Function PacketCapturingVNI(VNI.DomID.DevID)
    for each packets originated from VNI having domain ID as DomID do
        src.IP, eth.MAC=getPacketHeaderInfo(Packet);
        Append.Test_Db(Src.IP, eth.MAC, DomID);
    end
    IP_SpoofingDE(Test_Db);
    MAC_SpoofingDE(Test_Db);
return
Function IP_SpoofingDE(Test_Db)
    Flag=0;
    for each DomID in Test_DB do
        Extract Real.IP of VM with domain id equals to DomID from Baseline_Db;
        for each Packet of VM (DomID) do
            Extract src.IP from Test_Db;
            if src.IP!=Real.IP then
                Flag=1;
            end
            if Flag==1 then
                Print Domain (Domainname.value) is generating IP Spoofed packets with Spoofed address
                src.IP;
            end
        end
    end
return
Function MAC_SpoofingDE(Test_Db)
    Flag=0;
    for each DomID in Test_DB do
        Extract Real.MAC of VM with domain id DomID from Baseline_Db;
        for each Packet of VM (DomID) do
            Extract eth.MAC from Test_Db;
            if eth.MAC!=Real.MAC then
                Flag=1;
            end
            if Flag==1 then
                Print Domain (Domainname.value) is generating MAC Spoofed packets with Spoofed
                address eth.MAC;
            end
        end
    end
return
```

possible to identify which VM has generated the traffic which is originated from the VNI being monitored. Once the domain ID is known, MNPD can then identify which domain (such as Ubuntu-Guest-VM1 or Ubuntu-Guest-VM2) is generating the traffic by mapping the domain ID with domain name using Xen API commands.

Each domain is mapped to its real IP address and real MAC address at hypervisor. This is achieved by installing DHCP service (dnsmasq is considered in the implementation of MNPD) at the Dom0 and configuring it for each VM network settings. The configuration file is located at following path: `/etc/dnsmasq.conf`. The DHCP service is configured to allocate an IP address from a given range of IP addresses at the time of VM launch.

The MAC address is allocated to each VM at every launch operation which is configured at the VM configuration file (`/etc/xen/Ubuntu-Guest-VM1.cfg`). The configuration information related to MAC address for each VM is also saved in XenStore. However, XenStore does not provide the IP related information and is shared only with Domains. DHCP service stores the actual configured networking settings (IP address, MAC address etc.) at certain location in Xen (`/var/lib/libvirt/dnsmasq/virbr0.status`) which is stored on each booting process of VM. MNPD introspects the MAC and IP entries at that location, configured by dnsmasq DHCP service for obtaining actual network configuration information.

A baseline database (`Baseline_Db`) is created to store all the VM related information (`Domainname.value`, `DomainID.value`, `Real.IP` and `Real.MAC`) which is later used to verify the packer header information captured at VNI. We have retrieved `src.IP` and `eth.MAC` header values from each packet coming from VM at its actual VNI (backend driver interface) using tshark tool [198]. Each packet information with corresponding domain ID is saved in the `Test_Db`. The information in `Test_Db` is verified with baseline database (`Baseline_Db`) based on their domain ID. For each packet coming from VNI, the IP Spoofing detection module (`IPSpoof_DE()`) extracts the real IP information (`Real.IP`) from (`Baseline_Db`) and compares it with all the obtained IP values (`src.IP`) of the packets originated from TVM. If any of the packet mismatches with the real IP value allocated to VM, packet is detected as IP spoofed. Similarly, for each packet, MAC Spoofing detection module (`MACSpoof_DE()`) extracts the real MAC information

(`Real.MAC`) from (`Baseline_Db`) and compares it with all the obtained MAC values (`eth.MAC`) of the packets originated from TVM. If any of the packet mismatches with the real MAC value allocated to VM, packet is detected as MAC spoofed. The VM generating such packets is declared as suspicious. For each case, a message with details of spoofed values and compromised virtual machine is displayed at administrative interface.

6.3.3 Network Behavior Analyzer

The main goal of Network Behavior Analyzer (NBA) is to detect any abnormality in the network traffic at both the servers. The output of BCE is passed as input to NBA at CNS whereas the traffic which is passed by TrVal is analyzed by NBA at CCoS. NBA has two detection modules: (i) Optimal Feature Statistics Matrix (OFSM) Generation and (ii) Detection Engine which are described below:

6.3.3.1 Optimal Feature Statistics Matrix Generation

Optimal Feature Statistics Matrix (OFSM) represents the behavioral statistics (feature values) of the traffic which are to be analyzed by DE. NBA performs two steps: packet pre-processing and feature selection, to generate the OFSM log which is passed as a input to the detection engine. The packet capture files (.pcap) provided by BCE represent the basic statistics of the packet header values such as destination address, source address, prototype, etc. However, they are not sufficient to detect the attacks. Hence, in packet pre-processing, a set of advanced features such as source bits per sec, sum of SYNACK and destination bits per sec etc. are derived from captured data which represent the association between packet header values. There are some tools which can be used to derive various types of features from PCAP files such as NetMate-flowcalc [199]. All extracted features are stored in Feature Statistics Matrix (FSM) log file. Next, FSM is pre-processed to reduce the dimensionality (features) and to select the most important features which are more relevant for classification. We have used ensemble of Chi Square [193] and Recursive Feature Elimination (RFE) method [194]. Each of the transformer object (feature selection method) is applied to original dataset independently and in parallel. All feature vectors of the output are combined to form a new feature vectors and are stored in optimal FSM (OFSM) log file.

6.3.3.2 Detection Engine

Detection Engine (DE) is responsible for detecting intrusions at Network and VMM-layer. DE operates in two phases: learning and detection. DE employs the learning approach based on ensemble based classifier, particularly Random Forest (RF) [168]. In learning phase, RF trains itself with behavior statistics of prepared attack dataset (OFSM) and learns the attack patterns. It generates a decision model which is used to detect the anomalies in the network traffic at the time of detection. In detection phase, the sniffed packets are processed and prepared in a similar fashion as the packets of attack dataset are processed (explained above). However, the testing log file does not have any labels which are decided by the decision model. RF generates trained multiple decision trees over sub samples of training dataset (bootstrap sample). The output is the average of the predictions made by each tree. The main advantage of considering RF classifier is that it does not require extensive training and can fit with large databases very well. It is also not much sensitive to input parameters like SVM[200]

RF requires the tuning of one parameter i.e. no. of estimators, calculated by Out of Bag (OOB) error plot. Out of Bag (OOB) error is the average error for each instance (Xi, Yi) predicted by n classifiers trained over a bootstrap sample that does not contain (Xi, Yi) [185]. OOB highly affects the classifiers performance and directly relates to generalization error. Some researchers [119] have used Decision Tree (DT) [160] in their detection approach as DT is the simplest tree based classification algorithm. However, a tree can grow very deep and can tend to learn highly irregular patterns which can overfit the dataset instances. Hence, it will produce high classification error and low accuracy for large databases. The publicly available attack datasets such as ITOC [201] and UNSW-NB [189] contain large number of records. Decision Tree is not suitable for them. Instead, the ensemble of decision trees (DTs) (Random Forest) is found to perform well during implementation. The main advantage with Random Forest is that it handles high dimensional spaces as well as large number of training examples very well.

Algorithm 5: Algorithm for Intrusion Profile Generation and Detecting Malicious Packets

```
Initialization:
    X← {x1,x2,x3......xn}
    y← {y1,y2,y3......yk}
Result: $VM_1$ Intrusion Profile, $VM_2$ Intrusion Profile...$VM_n$ Intrusion Profile; Alerts and Logs
Function Profile_Generation_Module()
    for each monitored machine m do
        Log1(PCAP)=BeCap(VNI);
        for Each network packet i in Log1 do
            Label=Extract_label($Packet_i$);
            while $Log1 \neq NULL$ do
                FV=Packet_Pre-Processor($Packet_i$);
                Append.FSM(Label, FV);
            end
        end
        dt=DecisionTreeClassifier();
        rfe=RFE(dt, 8);
        chi2= SelectKBest(chi2, k=8);
        combined_features = FeatureUnion([("RFE", rfe), ("Chi Square", chi2)]);
        OFSM = combined_features.fit(X, y).transform(X);
        clf=RandomForestClassifier(n_estimators=130);
        X=OFSM [:,1:n-1] ;  // All rows from 2nd(index 1) to n (index n-1) columns
        ;
        Y=OFSM[0] ;         // First column contains target class labels (index 0)
        ;
        $Decision\_Model_m$=clf.fit(X, Y);
    end
return
Function Detection_Module()
    for each monitored machine m do
        Log2(PCAP)=BeCap(Interface card);
        for Each network packet i in Log2 do
            while $Log2 \neq NULL$ do
                Test_FV=Packet_Preprocessor($Packet_i$);
                Test_FSM=Write(Test_FV);
            end
        end
        Pred=Load($Decision\_Model_m$);
        Labels=pred.predict(Test_FSM);
        if Lables.find('intrusive') then
            Generate_Alert("Suspicious Activity Detected")–>CloudAdmin;
        end
        CloudAdmin–>Analysis(Log);
        CloudAdmin–>Respond();
    end
return
```

The trained classifier is stored as a baseline detector for matching traffic behavior. Each trained IDS instance is enabled at both CNS and CCoS. The trained profiles are stored in form of decision model in network intrusion profiles database (NIP Db) for each monitored machine as shown in Algorithm 5. In testing time, DE uses a pre-compiled intrusion profile of attacks stored in NIP Db as a decision model trained over attack patterns. The execution flow of MNPD for network attacks detection at CNS and CCoS is as follows:

1. MNPD invokes Behavior Capture Engine and captures the traffic.
2. Test log file is generated and loaded by executing the OFSM module.
3. The baseline detector (decision model) for monitored machine is loaded from NIP Db which is a trained ensemble classifier.
4. DE classifies the packet as anomalous or normal based on learned traffic behavior and generates an alarm to cloud administrator on detection of anomalous class.

As RF (supervised algorithm) is trained for a labeled attack dataset, it can only detect those attacks for which it is trained. It can also detect unknown attacks which are variants of known attack patterns. The attack variants exhibit the similar behavior characteristics of their parent class. The limitation with this approach is that it can not detect those unseen attacks which exhibit a totally different behavior than learned attack patterns.

6.3.4 Alert and Log Generator

A cloud administrator performs further analysis on suspicious traffic based on the output log created by MNPD. The log provides the information about the TVM together with its domain ID, domain name and VNI details generating the malicious traffic. Once, the details of machine generating the malicious traffic is known, cloud administrator can get the other details such as source/destination port details, source addresses, destination addresses etc. A tenant can be informed to close the applications associated with the suspicious ports. The suspicious network connections are terminated. If a tenant TVM generates a large number of malicious traffic, it is isolated for further analysis or restored to a previous checkpoint when it was found to be benign.

TABLE 6.1: Details of normal and intrusive records in the attack datasets

Intrusion Dataset	Total Records	Intrusive Records	Normal Records	No. of Attributes
ITOC	4,00,000	1,67,879	2,32,121	27
UNSW-NB	175,341	119,341	56,000	47

6.4 Experiments

The prototype of our approach has been implemented on a machine with Ubuntu 16.04 as host OS, Xen 4.6 as hypervisor, Core i7 processor with 16 GB RAM, 500 GB HDD and two guest VMs with Ubuntu (one is used as attacker machine and other one as victim machine). Each VM has 8GB RAM, 100GB HDD and Ubuntu 14.04 is installed as guest OS. Python 2.7.10 has been used as a programming language. We have first used UNSW-NB [189], a latest Intrusion Detection dataset, created at Cyber Range Lab of the Australian Centre for Cyber Security (ACCS). The dataset presents a current network threats and modern low frequency attacks. UNSW-NB dataset contains a total 47 features which are explained by Moustafa et al. [202]. There are 10 attack classes in the dataset namely Analysis, Fuzzers, Backdoors, Dos, Exploits, Generic, Reconnaissance, Shellcode and Worms. We further analyzed the performance with another ITOC dataset [201], generated by Information Technology Operations Center at United States Military Academy (USMA) which contains 27 records. ITOC dataset has two attack classes benign and malicious. The details of the dataset is shown in Table 6.1. The above two datasets are processed by MNPD detection components.

6.4.1 Performance Measures

The following parameters are considered for evaluation:

∗ **Accuracy:** This is one of the basic measures for describing the performance of classification algorithms. It describes the degree to which an algorithm can correctly predict the positive and negative instances and is calculated by the formula:

$$Accuracy = \frac{TN + TP}{TN + TP + FN + FP} \tag{6.1}$$

∗ **False Positive Rate (FPR)**: The proportion of incorrectly classified intrusions to the actual size of the attack class and is calculated by the formula:

$$False\,Positive\ Rate = \frac{FP}{FP + TN} \tag{6.2}$$

* **False Negative Rate (FNR)**: The proportion of incorrectly classified normal traffic to the actual size of the normal class and is calculated by the formula:

$$False\,Negative\ Rate = \frac{FN}{TP + FN} \tag{6.3}$$

* **True Positive Rate (TPR)**: The proportion of correctly classified intrusions to the actual size of the attack class and is calculated by the formula:

$$True\,Positive\ Rate = \frac{TP}{TP + FN} \tag{6.4}$$

* **True Negative Rate (TNR)**: The proportion of correctly classified normal traffic to the actual size of the normal class and is calculated by the formula:

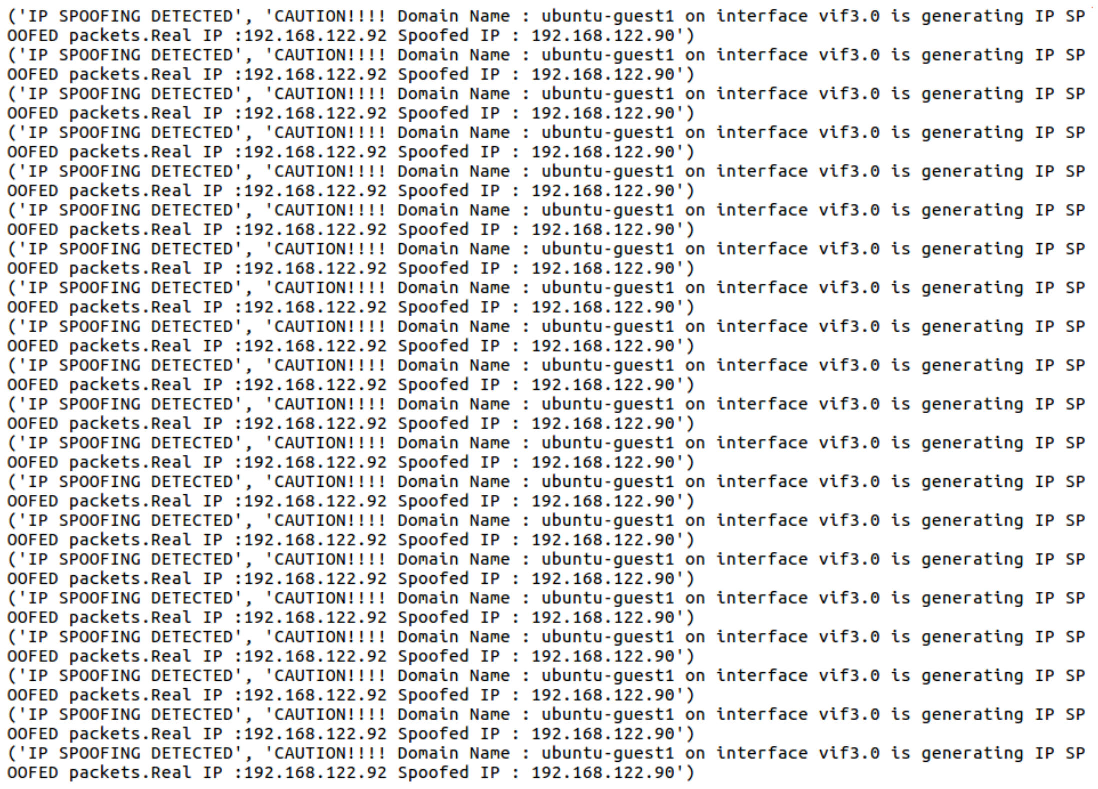
$$True\,Negative\ Rate = \frac{TN}{TN + FP} \tag{6.5}$$

The validation of our technique is described in subsequent section.

6.4.2 Results and Discussion

In this section, MNPD is validated and results are discussed. The performance metrics, considered for evaluation have been discussed in previous section. MNPD detection approach is compared with existing work for network intrusion detection in cloud.

To evaluate the TrVal detection component, we first created two guest VMs (TVMs) running over hypervisor. Both the VMs are configured to run over same tenant subnet, so that they can communicate with each other. The first VM (VM1) is used to create and send IP spoofed packets to victim target VM (VM2). IP spoofed packets are created at VM1 using hping3 tool [102] with spoofed IP: 192.168.122.90 and real IP: 192.168.122.92. Once the MNPD instance was activated at Dom0, alert messages were generated with the details of the VM generating malicious IP spoofed traffic as shown in Figure 6.4. It can be seen



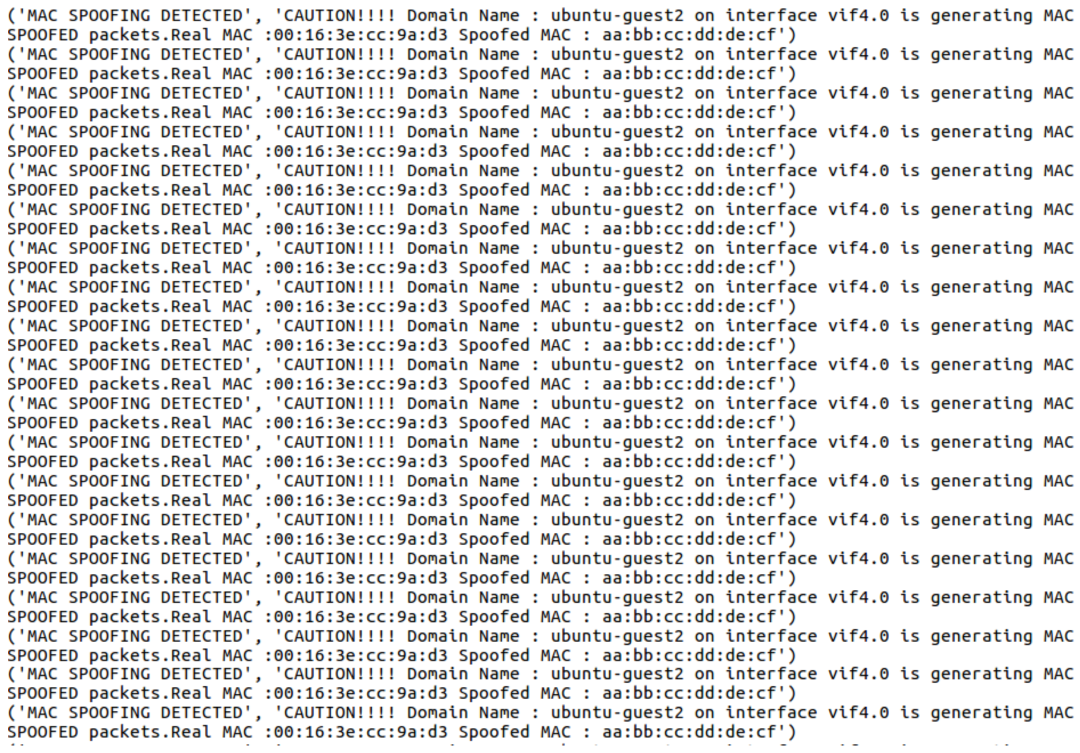
```
('IP SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest1 on interface vif3.0 is generating IP SP
OOFED packets.Real IP :192.168.122.92 Spoofed IP : 192.168.122.90')
('IP SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest1 on interface vif3.0 is generating IP SP
OOFED packets.Real IP :192.168.122.92 Spoofed IP : 192.168.122.90')
('IP SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest1 on interface vif3.0 is generating IP SP
OOFED packets.Real IP :192.168.122.92 Spoofed IP : 192.168.122.90')
('IP SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest1 on interface vif3.0 is generating IP SP
OOFED packets.Real IP :192.168.122.92 Spoofed IP : 192.168.122.90')
('IP SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest1 on interface vif3.0 is generating IP SP
OOFED packets.Real IP :192.168.122.92 Spoofed IP : 192.168.122.90')
('IP SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest1 on interface vif3.0 is generating IP SP
OOFED packets.Real IP :192.168.122.92 Spoofed IP : 192.168.122.90')
('IP SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest1 on interface vif3.0 is generating IP SP
OOFED packets.Real IP :192.168.122.92 Spoofed IP : 192.168.122.90')
('IP SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest1 on interface vif3.0 is generating IP SP
OOFED packets.Real IP :192.168.122.92 Spoofed IP : 192.168.122.90')
('IP SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest1 on interface vif3.0 is generating IP SP
OOFED packets.Real IP :192.168.122.92 Spoofed IP : 192.168.122.90')
('IP SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest1 on interface vif3.0 is generating IP SP
OOFED packets.Real IP :192.168.122.92 Spoofed IP : 192.168.122.90')
('IP SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest1 on interface vif3.0 is generating IP SP
OOFED packets.Real IP :192.168.122.92 Spoofed IP : 192.168.122.90')
('IP SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest1 on interface vif3.0 is generating IP SP
OOFED packets.Real IP :192.168.122.92 Spoofed IP : 192.168.122.90')
('IP SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest1 on interface vif3.0 is generating IP SP
OOFED packets.Real IP :192.168.122.92 Spoofed IP : 192.168.122.90')
('IP SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest1 on interface vif3.0 is generating IP SP
OOFED packets.Real IP :192.168.122.92 Spoofed IP : 192.168.122.90')
('IP SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest1 on interface vif3.0 is generating IP SP
OOFED packets.Real IP :192.168.122.92 Spoofed IP : 192.168.122.90')
('IP SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest1 on interface vif3.0 is generating IP SP
OOFED packets.Real IP :192.168.122.92 Spoofed IP : 192.168.122.90')
('IP SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest1 on interface vif3.0 is generating IP SP
OOFED packets.Real IP :192.168.122.92 Spoofed IP : 192.168.122.90')
('IP SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest1 on interface vif3.0 is generating IP SP
OOFED packets.Real IP :192.168.122.92 Spoofed IP : 192.168.122.90')
('IP SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest1 on interface vif3.0 is generating IP SP
OOFED packets.Real IP :192.168.122.92 Spoofed IP : 192.168.122.90')
('IP SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest1 on interface vif3.0 is generating IP SP
OOFED packets.Real IP :192.168.122.92 Spoofed IP : 192.168.122.90')
```

FIGURE 6.4: Output of MNPD for IP Spoofing Attack

that VM with certain domain name 'ubuntu-guest1' is generating the IP spoofed packet, received at its vif3.0 backend driver. Domain ID of VM can be interpreted from VNI number which is 3 in this case. To block the packets arriving with particular source IP (src.IP), MNPD enables `iptable` filter rules at virbr0 bridge of hypervisor.

In another case, second VM (VM2) is used to create and send MAC spoofed packets to victim target VM (VM1). MAC spoofed packets are created at VM2 using nmap tool [203] with spoofed MAC: aa:bb:cc:dd:de:cf and real MAC: 00:16:3e:cc:9a:d3. Once the MNPD instance is activated at Dom0, alert messages were generated with the details of the VM generating malicious MAC spoofed traffic as shown in Figure 6.5. It can be seen that VM with domain name 'ubuntu-guest2' is generating the MAC spoofed packet, received at its vif4.0 backend driver where 4 is the domain ID of VM2. To block the packets with particular MAC address, MNPD enables `ebtable` filter rule at virbr0 bridge of hypervisor.

To evaluate the behavior analysis functionality, the dataset is preprocessed by extracting the relevant features, normalizing and scaling



```
('MAC SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest2 on interface vif4.0 is generating MAC
SPOOFED packets.Real MAC :00:16:3e:cc:9a:d3 Spoofed MAC : aa:bb:cc:dd:de:cf')
('MAC SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest2 on interface vif4.0 is generating MAC
SPOOFED packets.Real MAC :00:16:3e:cc:9a:d3 Spoofed MAC : aa:bb:cc:dd:de:cf')
('MAC SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest2 on interface vif4.0 is generating MAC
SPOOFED packets.Real MAC :00:16:3e:cc:9a:d3 Spoofed MAC : aa:bb:cc:dd:de:cf')
('MAC SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest2 on interface vif4.0 is generating MAC
SPOOFED packets.Real MAC :00:16:3e:cc:9a:d3 Spoofed MAC : aa:bb:cc:dd:de:cf')
('MAC SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest2 on interface vif4.0 is generating MAC
SPOOFED packets.Real MAC :00:16:3e:cc:9a:d3 Spoofed MAC : aa:bb:cc:dd:de:cf')
('MAC SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest2 on interface vif4.0 is generating MAC
SPOOFED packets.Real MAC :00:16:3e:cc:9a:d3 Spoofed MAC : aa:bb:cc:dd:de:cf')
('MAC SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest2 on interface vif4.0 is generating MAC
SPOOFED packets.Real MAC :00:16:3e:cc:9a:d3 Spoofed MAC : aa:bb:cc:dd:de:cf')
('MAC SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest2 on interface vif4.0 is generating MAC
SPOOFED packets.Real MAC :00:16:3e:cc:9a:d3 Spoofed MAC : aa:bb:cc:dd:de:cf')
('MAC SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest2 on interface vif4.0 is generating MAC
SPOOFED packets.Real MAC :00:16:3e:cc:9a:d3 Spoofed MAC : aa:bb:cc:dd:de:cf')
('MAC SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest2 on interface vif4.0 is generating MAC
SPOOFED packets.Real MAC :00:16:3e:cc:9a:d3 Spoofed MAC : aa:bb:cc:dd:de:cf')
('MAC SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest2 on interface vif4.0 is generating MAC
SPOOFED packets.Real MAC :00:16:3e:cc:9a:d3 Spoofed MAC : aa:bb:cc:dd:de:cf')
('MAC SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest2 on interface vif4.0 is generating MAC
SPOOFED packets.Real MAC :00:16:3e:cc:9a:d3 Spoofed MAC : aa:bb:cc:dd:de:cf')
('MAC SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest2 on interface vif4.0 is generating MAC
SPOOFED packets.Real MAC :00:16:3e:cc:9a:d3 Spoofed MAC : aa:bb:cc:dd:de:cf')
('MAC SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest2 on interface vif4.0 is generating MAC
SPOOFED packets.Real MAC :00:16:3e:cc:9a:d3 Spoofed MAC : aa:bb:cc:dd:de:cf')
('MAC SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest2 on interface vif4.0 is generating MAC
SPOOFED packets.Real MAC :00:16:3e:cc:9a:d3 Spoofed MAC : aa:bb:cc:dd:de:cf')
('MAC SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest2 on interface vif4.0 is generating MAC
SPOOFED packets.Real MAC :00:16:3e:cc:9a:d3 Spoofed MAC : aa:bb:cc:dd:de:cf')
('MAC SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest2 on interface vif4.0 is generating MAC
SPOOFED packets.Real MAC :00:16:3e:cc:9a:d3 Spoofed MAC : aa:bb:cc:dd:de:cf')
('MAC SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest2 on interface vif4.0 is generating MAC
SPOOFED packets.Real MAC :00:16:3e:cc:9a:d3 Spoofed MAC : aa:bb:cc:dd:de:cf')
('MAC SPOOFING DETECTED', 'CAUTION!!!! Domain Name : ubuntu-guest2 on interface vif4.0 is generating MAC
SPOOFED packets.Real MAC :00:16:3e:cc:9a:d3 Spoofed MAC : aa:bb:cc:dd:de:cf')
```

FIGURE 6.5: Output of MNPD for MAC Spoofing Attack

TABLE 6.2: Results of MNPD over UNSW-NB intrusion dataset

Classifiers	**Accuracy** (%)	**False Positive Rate** (%)
DT [204]	85.56	15.78
NB [204]	82.07	18.56
LR [204]	83.15	18.48
ANN [204]	81.34	21.13
EM clustering [204]	78.47	23.79
DT (RFE)	91.570	6.21
DT (RFE+Chi Square)	93.903	5.218
RF (RFE)	92.458	4.121
RF (RFE+Chi Square)	**95.091**	**2.415**

the dataset. Tree based classifiers, particularly Decision Tree and Random Forest are used for analysis using k-fold cross validation where k=10. Feature selection plays a very important role to improve the performance of classifier in a high dimensional dataset. The selected features have a better ability to explain the variance in the training data and improve the accuracy of a classifier. Hence, 16 features are selected by using feature union of RFE [194] and Chi Square [193] methods.

First of all, the behavior analysis functionality of MNPD is evaluated with UNSW-NB dataset [189], a latest intrusion detection dataset. The prime motive was to provide an ensemble method which can provide an

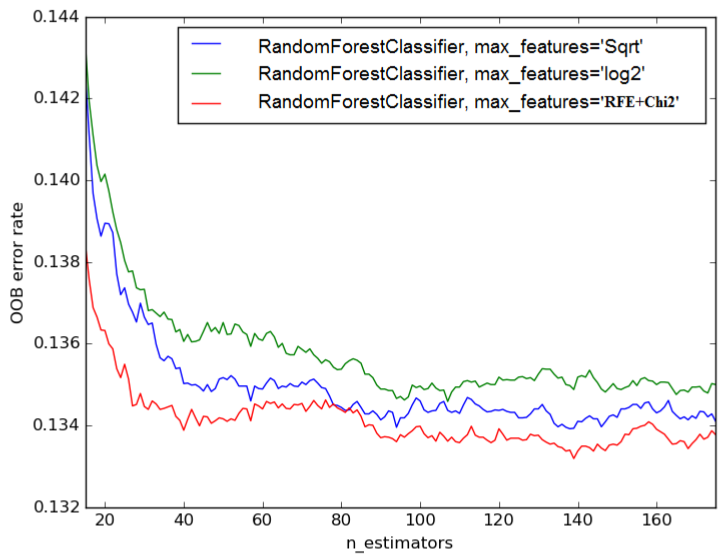


FIGURE 6.6: Estimator prediction based on OOB error for RF (UNSW-NB)

improvement over existing results of Moustafa et al. [204] using UNSW-NB. Moustafa et al. have considered accuracy and FPR for evaluation. The same parameters are considered for evaluating MNPD. They have used different single classifiers DT, Naive Bayes (NB), Artificial Neural Network (ANN), Logistic Regression (LR), Expectation-Maximization (EM) clustering and found that DT is proving best results with 85.56% accuracy with 15.78% FPR as shown in Table 6.2. In our implementation, DT with RFE improves the results with 91.570% accuracy and 6.21% FPR.

Its accuracy is further improved when it is trained and tested over combined feature union method (RFE and Chi Square) which turns out to be 93.903% with 5.218% FPR. Further, Random Forest is executed with 130 decision trees and bagging method is applied. The estimators are predicted based on the OOB error. To perform hyperparameter tuning, RF is trained in iteration over a range of 10-200 estimators. In each iteration OOB error is calculated. A curve plotted between out-of-bag (OOB) error and number of estimators (DTs) as shown in Figure 6.6. Finally the number of estimators (decision trees) are selected as 130 for which OOB error found to be minimum. RF is also tested in both scenarios of feature selection i.e (RF with RFE and RF with RFE & Chi Square). RF with RFE feature selection provides an accuracy of 92.458% with 4.121% FPR. We found the improvement in accuracy when it is trained over the combined feature union method (RFE and

Table 6.3: Classification results of MNPD for intrusion detection using ITOC attack dataset

Classifier+Feature selec.	**Accuracy**	**TPR**	**TNR**	**FNR**	**FPR**
NB (Chi Square)	68.760	51.100	81.544	48.89	18.455
DT (Chi Square)	92.740	88.54	97.7914	11.458	4.208
RF (Chi Square)	97.070	97.052	97.085	2.947	2.914
NB (RFE+Chi Square)	68.723	51.112	81.460	48.887	18.539
DT (RFE+Chi Square)	94.659	89.800	98.17	10.199	1.8266
RF (RFE+Chi Square)	98.888	99.182	98.482	0.8176	1.517
DT+SVM [87]	84.30	86.84	71.66	13.16	28.34

Chi Square). The accuracy comes out to be 95.091% with 2.415% FPR which is better than the results by Moustafa et al. over the same dataset (refer Table 6.2).

We considered another attack dataset (ITOC) for evaluating the behavior analysis functionality of MNPD. The ensemble method of MNPD provides an improvement over existing results of Singh et al. [87] using ITOC dataset as shown in Table 6.3. Singh et al. [87] have considered TPR, TNR, FPR, FNR, Accuracy for evaluation. We have also considered the parameters for evaluation. The performance of Naive Bayes (NB), Decision Tree C 4.5 (DT) and Random Forest (RF) classifiers is also compared. Initially we run NB algorithm in integration with Chi Square as feature selection method. NB gives a low poor accuracy of 68.760% with 51.100% TPR and 81.544% TNR. It provides a high false positives (19.455%-48.89%). There is not much improvement in the performance when running NB with an ensemble of feature selection methods (RFE + Chi Square). There is a very slight variation in all parameters. NB is based on the assumption of that all features are independent from each other. This is not a good criteria for intrusion detection as the network features are derived features and there is definitely some relation between them. We have made a detailed investigation of the intrusion detection using machine learning classifiers and observed that rule based classifiers have been found to perform well for network intrusion detection system applications. The effectiveness of rule based ensemble learning for detecting network intrusions has also been proved by Panda et al [205].

Hence, MNPD applies the ensemble learning particularly RF classifier for attack detection in cloud with ensemble of feature selection approaches. However, we have compared the two popular rule based classifiers DT C 4.5 and RF for attack detection. First of all, both

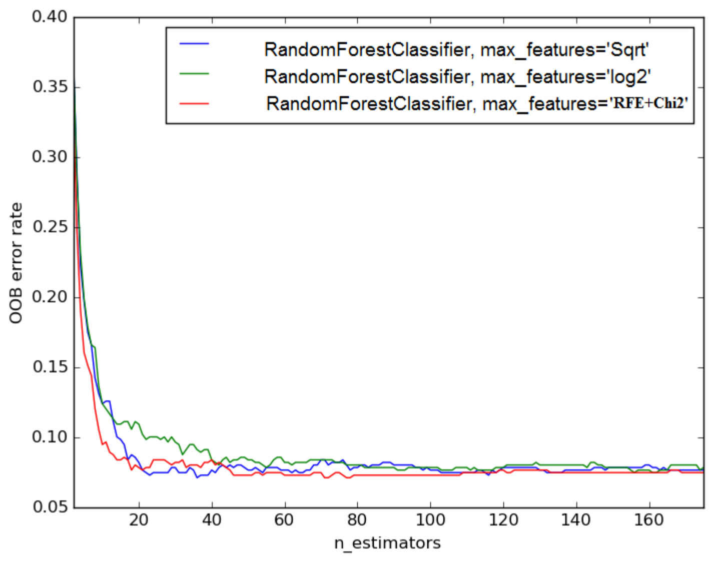


Figure 6.7: Estimator prediction based on OOB error for RF (ITOC)

DT and RF are executed in integration with only Chi Square feature selection method. DT with Chi Square is resulting in a accuracy of 92.740% accuracy with 88.54% TPR, 97.7914% TNR, 11.458% FNR and 4.208% FPR. The performance of DT is improved using a feature union method (union of RFE and Chi Square). It is providing an accuracy of 94.659% with 89.800% TPR, 98.17% TNR, 10.199% FNR and 1.8266 % FPR. There is an improvement in accuracy by $\sim 2\%$. There is a good reduction in the FPR rate by $\sim 2.3814\%$. However, the accuracy has not increased. A single DT classifier tends to overfit over a large size of dataset which contains many records. Hence, we further applied RF, an ensemble classifier with many decision trees. In case of RF, we have considered 80 decision trees and applied bagging method. The estimators are predicted based on the OOB error as shown in Figure 6.7. The parameter tuning is performed in a similar way as discussed above for UNSW-NB dataset. RF with Chi Square is providing 97.070% detection accuracy with 97.052% TPR, 97.085% TNR, 2.947% FNR and 2.914% FPR. RF is providing good detection performance when compared with other classifiers in all the cases discussed above. RF when executed with an ensemble of RFE and Chi Square achieves much better accuracy of 98.888%. The true alarms have improved and false alarms have reduced. We can infer that the irrelevant features having low discrimination power may affect a classifiers performance for classifying instances. It is providing 99.182% TPR, 98.482% TNR, 0.8176% FNR and 1.517% FPR with the ensemble of features selection approaches. The final results of detection engine of MNPD based on RF with (RFE and Chi Square) are compared with the recent work by Singh et al. [87] for network intrusion detection in cloud. Singh et al. achieved a low accuracy of 84.30% for the ITOC dataset by proposing the integration of DT and SVM. MNPD improves its accuracy by $\sim$ 14. DT in integration with SVM also provides low TPR of 86.84% with 71.66% TNR. It produces more false alarms with 28.34% FPR and 13.16% FNR. The integration of DT with a completely different classifier i.e. SVM (based on the concept of hyperplane) is producing the poor results for ITOC. SVM is itself very sensitive to parameters and require tuning of parameters for even any small variation in the dataset. Hence, the rule based classifier applied by MNPD is provided to be effective for network intrusion detection.

We have compared the performance of the proposed MNPD approach with other approaches in cloud environment: ecloudIDS [139], NIDS [90], C-IDS [87] and 'Hypervisor Detector' [126]. All the approaches other than eCloudIDS are misuse detection approaches which use the existing knowledge of attack signatures and/or labeled attack traffic dataset. The eCloudIDS [87] uses traditional Self Organizing Map (SOM) classifier as an anomaly detection module to learn the normal behavioral profile of users. Any deviation from learned profile is notified as abnormal behavior. However, generalizing the clients behavior is difficult in dynamic cloud environment because of the evolving 'normal' traffic flows. NIDS [90], the core module uses SNORT as a primary detection module. The output of SNORT is integrated with the decision tree classifier algorithm which runs only for the traffic, marked as normal by SNORT. As all the benign traffic is forwarded to classifier and the malicious traffic marked as bad, this does not reduce the false positive rate of Snort (only the false negative). Also, there is no verification of an alert; it does not reduce the false alarms, neither for Snort nor for the DT. The parameters considered for comparison are shown in Table 6.4.

C-IDS [87] is a slight variation of NIDS. The only difference being the use of machine learning approaches. C-IDS uses the ensemble of decision tree and SVM classifier in integration with SNORT for detecting the intrusions. It has the same limitation as discussed for NIDS. The proposed MNPD approach is based on the integration of network introspection and machine learning classifier to detect intrusions in cloud environment. It was observed that the Random Forest (RF) technique, which is an ensemble learning classifier, provides better results than when compared with other classification algorithms. An ensemble learning approach generates more than one classifiers and aggregates their results. RF combines the idea of bagging and the random selection of features. 'Hypervisor Detector' [126] uses a network traffic analysis approach which employs misuse detection approach based on the Fuzzy c-means clustering technique integrated with ANN deployed at VMM. However, it involves extensive training due to the complexity of the algorithm. This is not desirable in cloud environment as it may require often retraining of IDS instances.

Most of these approaches (eCloudIDS, NIDS) with some other network intrusion approaches for cloud [74][82][188] are deployed at TVM-layer. This increases the risk of IDS subversion because of the lack of trust in the guest OS kernel. Secondly, spoofing is difficult to be detected at TVM-layer. Hypervisor is the only entity in the VM hosted server which is responsible for the network and TVM configuration and set up of TVMs. Tenant domains (DomUs) are not allowed to retrieve any network related information from hypervisor, set up by cloud admin at the time of VM launch. C-IDS is deployed at cloud servers/cluster nodes. The virtual IP information gets lost once a packet passes the virtual bridge of hypervisor. Hence, detecting attacks below hypervisor, at the host OS of the hypervisor, does not provide the fine granularity. 'Hypervisor Detector' is deployed at the VMM. However, the technical details about the deployment have not been discussed. Moreover, CNS is the prime source of target for network intrusions. None of the approaches discuss the security at the network server. MNPD provides two-levels of security check. The first is done at the CNS and other at VMM of CNS. Unlike other approaches, MNPD performs traffic validation as a second-level security check for checking the IP spoofing and MAC spoofing attacks at hypervisor itself. It blocks the spoofed packets to reach to destination machine (same server or other server). Unlike all other approaches, MNPD considers feature selection using ensemble of RFE and Chi Square which removes the irrelevant features from the dataset and hence improves the accuracy of detection engine.

Table 6.4: Comparison of MNPD with other approaches proposed for Cloud Security

Parameter	MNPD	eCloudIDS [139]	NIDS [90]	C-IDS [87]	Hypervisor Detector [126]
IDS technique	Misuse detec. with VMI	Anomaly detec.	Misuse detec.	Misuse detec.	Misuse detec.
Key tool/ Algorithm	Network Introspection with ML	ML	SNORT and ML	SNORT and ML	ML
Placement of IDS	CNS and VMM (dom0)	VM	VM, cloud servers	cloud servers	VMM
Feature selection	Ensemble (RFE + Chi2)	NA	NA	NA	NA
Traffic validation	Applicable	NA	NA	NA	NA
Behavior analysis	Considered	Considered	Considered	Considered	Considered
IDS subversion	Difficult	Easy	Easy	Easy	Moderate
Details of virtual networking	Specified	Not specified	Not specified	Not specified	Not specified
Robustness	High	Low	Low	Low	Moderate
Dataset	UNSW-NB, ITOC	Self	KDD99	KDD99, ITOC	KDD99
Accuracy	98.88% (ITOC), 95.091% (UNSW-NB)	89%	96.71% (KDD99), 84.31% (NSL-KDD99)	99.98% (KDD99), 84.30% (ITOC)	98%

The TVM-layers approaches (eCloudIDS, NIDS) are prone to guest kernel modification attacks and can be easily compromised by attackers. These approaches are less robust and have high risk of subversion. C-IDS is prone to host manipulation attacks. In fact, it may not be able to capture the traffic from co-located TVMs as the virtual traffic never passes through the physical interface. It is less robust and can also be bypassed by spoofed attack traffic. 'Hypervisor Detector' analyzes the virtual traffic at only VMM. However, it cannot identify the spoofed attack traffic and hence the robustness is moderate.

The eCloudIDS is validated with self generated dataset and accuracy is 89% with 9% FPR. NIDS have been validated with a very older attack dataset, i.e. KDD99 and NSL-KDD9. It achieves an accuracy of 96.71% with 1.91% FPR using KDD99 and 84.31% accuracy with 4.81% FPR with NSL-KDD99. 'Hypervisor Detector' is also validated with very older KDD99 dataset and achieves 98% accuracy. C-IDS achieves an accuracy of 99.98% with 0.01% false alarms using KDD99. Its accuracy is 84.30% with 13.16% false alarm using ITOC dataset which is comparatively newer than KDD99. MNPD is validated with latest intrusion detection dataset (UNSW-NB) and achieves 95.091% accuracy with 2.415% FPR. MNPD is also tested with ITOC dataset. The accuracy is 98.887% with 1.533% false alarms which is better than C-IDS, a recent intrusion detection proposal. Most of the IDS techniques have been with KDD99. As the evaluation is based on very older dataset, no estimation of the real performance of the system can be assessed.

6.5 Conclusion

A robust and learning based security approach Malicious Network Packet Detection (MNPD) is proposed for detecting malicious network traffic in cloud environment. CNS is the primary component which is responsible for routing of cloud network packets from VM to VM, VM to outside world and vice versa. Hence, in the proposed security architecture, one instance of MNPD is deployed at the CNS to monitor the communication of VMs at the network level. CCoS is another critical security component which runs various VMs. Hence, other instances of MNPD are deployed at Dom0 of each of the CCoS which validates all traffic originated by VM for detecting IP and MAC spoofing attacks. Only legitimate packets are forwarded to other interfaces. Behavior analysis of network traffic is performed at both the servers to detect the abnormality in the traffic using pre-compiled intrusion profile and machine learning algorithm. The two-levels of security checks improves the robustness of the system. It also eliminates the need of IDS deployment at each and every VM instance. It achieves 95.091% accuracy with 2.415% FPR using UNSW-NB dataset and an accuracy of 98.88% with 1.517% FPR using ITOC dataset. The results are better than existing recent works for network intrusion detection using same datasets. In future, we plan to provide an extended framework for doing the detailed investigation of attacks by performing memory introspection at VMM of VM hosted servers.

Chapter 7

Conclusion and Scope for Future Work

7.1 Conclusion

In this thesis, a comprehensive intrusion detection framework is developed to detect attacks in cloud environment by provisioning three-lines of defense, covering all three layer of cloud, i.e. TVM-layer, VMM-layer and Network-layer.

We started with a threat model, proposed in cloud environment which helps in analyzing the vulnerabilities and assets that are most likely to be targeted in cloud environment. The proposed threat model addresses the various vulnerable attack surfaces and attack entities in cloud. The attack scenarios are discussed with respect to each attack surface. An attack taxonomy is proposed in cloud which represents a systematic framework to classify attacks and highlight the need for intrusion detection mechanisms. The classification of various attacks is done based on the target cloud components where attacks target specific layers of cloud. Some of these attacks have been addressed in this thesis to detect the intrusion attempts against virtual domains running in cloud. Based on the exhaustive literature study, a classification of detection mechanism is proposed with various examples. A detailed description of various intrusion detection systems is presented to give in-depth view of the various cloud based IDS. Our findings represent the observations after going through various techniques in detail, and they point out the pros and cons of each category, technique and approach. A list of research challenges have also been outlined.

We proposed an efficient, robust and Virtual Machine Introspection(VMI)-based distributed security framework, called **CloudHedge** to detect intrusion at different layers i.e. TVM and VMM and Network-layer in cloud environment. CloudHedge monitors the behavior of both programs and network traffic of monitored TVMs by using the various proposed security approaches integrated with the memory and network introspection capabilities. It provides three line of defense in form of three sub IDS instances which are TVM-based monitoring, Hypervisor-based monitoring and Network-based monitoring. These are named as Malicious System Call Sequence Detection (MSCSD), VM Introspection based Malware Detection (VIMD) and Malicious Network Packet Detection (MNPD) respectively. MSCSD runs inside the TVM (TVM-layer). VIMD runs at the dom0 of the hypervisor (VMM-layer). MNPD runs at both Cloud Network Server (Network-layer) and Dom0 of the hypervisor (VMM-layer). All these sub IDS instances are the part of CloudHedge. We assume that Cloud Service Provider (CSP) is a faithful and reputed organization and VMM platform is trusted. Each of the sub IDS instance of CloudHedge has privilege to access the application specific information of monitored VMs. This privacy concern is clarified between CSP and tenant users at the time of registration in form of Service Level Agreement (SLA).

The sub-IDS instances are distributed and deployed at security-critical positions such as TVM, VMM and Network-server, covering all three layers i.e. **TVM, VMM and Network-layer** in cloud. Each of the sub-IDS is configured, monitored and controlled by the cloud administrator. A tenant member has no control over any of the sub-IDS instance. CloudHedge leverages the use of various VM Introspection libraries to perform the introspection from outside the TVM. The design of CloudHedge is efficient as it provides different security approaches to deal with both network and malware intrusions. It is robust since if an attacker becomes successful in evading the TVM-layer security approach, the other security approaches deployed out-of-the-VM would still be running actively. The design of CloudHedge is motivated from the fact that centralized IDS becomes a bottle neck when the number of TVMs in cloud increase. We also identified that there is less viability and efficiency of distributed IDSes which deploy same security solutions at all regions of cloud. This is because of the limitations and design choices associated with different layers. CloudHedge sub-IDS instances are distributed in nature providing different security approaches at each layer. They do not share the same VM resources for their operations, reducing the overhead at individual TVM. The implementation set up is based on Xen VMM for hosting the TVMs. We have considered the cloud architecture based on OpenStack [16]. The summary of each of the sub-IDS instance is given below.

Malicious System Call Sequence Detection (MSCSD) sub IDS instance provides the first-line of defense which is deployed at TVM-layer in cloud. MSCSD is applicable to traditional physical hosts and all cloud deployments such as SaaS, PaaS and IaaS. MSCSD performs the dynamic analysis (run-time behavioral analysis) of the programs, running in the monitored tenant VMs and centrally controlled and coordinated by cloud administrator. It can access all the contextual information without requiring any complex functions for trace extraction. MSCSD observes the run-time behavior of the programs, hence it is free from anti-detection techniques such as obfuscation and encryption techniques. First of all, it extracts the execution traces of monitored programs (located in program file list) in form of system call logs. MSCSD then applies the proposed 'Bag of n-grams (BonG)' approach for extracting features out of the collected logs. BonG finds out the frequency and structure of various short sequence of system calls, called n-grams. It is therefore successful in maintaining the ordering of the subsequent system calls within each sub-sequence. All the extracted feature vectors represents the behavior of system call sequences in form of the frequency distribution of unique n-grams present in normal and intrusive traces which is learned by machine learning (Decision Tree C 4.5) classifier. In detection phase, the trained model is used to analyze the behavior of running processes and detect suspicious activity. MSCSD has been compared with existing dynamic analysis based intrusion detection approaches [67] [84] for cloud and proved to be efficient with better accuracy. The key advantage with MSCSD is that it improves the accuracy and reduces the storage requirement when compared to other behavior analysis approaches for cloud while maintaining the ordering of system calls. It is validated using University of New Mexico (UNM) dataset [155] and achieves an accuracy of 72.103%-99.812%. It is secure from string manipulation attacks as it uses the numeric feature vector. It can be applied in cloud environment as a first-line of defense mechanism and is better suited to the privacy concerned tenants.

VM Introspection based Malware Detection (VIMD) sub IDS instance provides the second-line of defense which is deployed at VMM-layer. The main motivation behind VIMD is to detect both basic and stealthy advanced attacks at VMM-layer in cloud. VIMD uses the open source VMI libraries extensively. It provides two-levels of security and operates in three phases: memory introspection, behavior analysis of syscalls at hypervisor and alerting & reporting phase. In memory introspection, VIMD performs the primary security check to ensure that all the security-critical processes, running at TVM such as auto-update, auto-scan are enabled. It also detects the presence of hidden processes at TVM memory. If any suspicious activity is detected, cloud administrator is alerted about it. It further captures the execution tracing of the programs, running at TVM, from hypervisor by employing a kernel debugging based VM introspection mechanism. VIMD then performs the secondary security check by doing the detailed behavior analysis on processes in next phase. In behavior analysis phase, VIMD performs the behavior analysis using two core detection components: VMGuard and VAED. In alerting and reporting phase, Alert and Log Generator (ALG) is invoked to create logs and send alerts to the cloud administrator with detailed report. VIMD has been implemented with both the core detection components. The detailed description of core detection mechanism using VMGuard and VAED is given below.

VMGuard is based on the system call sequence analysis which detects attacks at VMM-layer in cloud. It takes the input from the memory introspection phase of VIMD and performs behavior analysis. It is based on the frequency model which integrates the BonG approach with text mining approach, particularly Term Frequency-Inverse Document Frequency (TF-IDF) to extract the rare system call sequences and improve the attack detection rate. TF-IDF considers two factors: frequency and rarity while giving ranks to the extracted n-grams. The extracted relevant features are stored in the Feature Vector Matrix (FVM) log file which is given as input to detection engine. The detection engine of VMGuard is based on the ensemble learning approach (Random Forest) which is used to learn and detect the program semantics of the malware. It has been validated with the UNM dataset [155]. VMGuard achieves an accuracy of 94%-100% in detecting intrusions. UNM dataset is in form of system call traces. VMGuard considers the structural aspects of the traces. VMGuard is found to perform well to detect malware attacks in cloud which do not depend on the system artifacts (e.g. privileged program subversion attacks) at VMM-layer in cloud. However, it does not capture the more complex behavioral aspects of the programs. It is less suitable to detect evasion based attacks which change their behavior on detection of some security tool. The detection of evasive malware attack is addressed by another approach, discussed below.

VAED is based on the system call transition analysis which can be opted by CSP to detect the evasive malware attacks at VMM-layer in cloud. It is based on the probability model which considers both structural and behavioral aspects of traces. VAED captures the program semantics in different execution paths of the program in form of system call dependency graph (SCDG). The semantics are based on the ordered sequence of system calls with the analysis of the transition probabilities from one system call to other possible system call. The transition probabilities are calculated by using the Markov Chain property. It considers the frequency of a system call transition with respect to all other system call transitions. This helps to capture the more complex behavioral aspects of the evasive malware. The extracted features for each SCDG, are stored in form of <transition-value pair> in Feature Transition Matrix (FTM). The transitions having good information centric paths are chosen by applying Information Gain Ratio (IGR) over FTM and stored in Optimal FTM (OFTM). OFTM is learned by ensemble classifier, based on fusion of diverse classifiers. Here, weighted voting scheme is used as a fusion rule to fuse the diverse classifiers results. The trained classifier captures the behavior semantics of evasive malware from OFTM. It is used as baseline information to detect abnormality. It has been validated with evasive malware dataset [158] (in form of malicious programs), obtained from University of California. Evasive malware dataset is first injected in clone VMs for extracting the behavior using PET component. The extracted traces are now analyzed using VAED approach for evasion detection. VAED achieves an accuracy of 97.50%-98.833% for detecting evasive malware.

The core detection components of VIMD has been successfully validated and seems to provide good results. To validate other components of VIMD specially PV and PET, evasive malware dataset [158] has been used. VIMD detected the hidden processes successfully. We have compared VIMD with other intrusion detection frameworks in a virtualization environment: Maitland [91], Xenini-IDS [69] and ShadowContext [93]. It is proved to be better while considering various parameters for comparison. The key advantage with VIMD is that its runs as a security application out-of-the-VM at Dom0 of hypervisor and provides a complete solution to detect both basic and evasive malware attacks in cloud by providing efficient detection approaches. It is better suited to provide second-line of security defense mechanism for the privacy cum security concerned tenants.

Malicious Network Packet Detection (**MNPD**) sub IDS instance provides the third-line of defense which is deployed at Network and VMM-layer in cloud. The security approaches as discussed above, perform the careful examination of the system calls which is the key in identifying the interaction between program and guest OS. This information is very important to detect malware attacks. However, system call analysis based approaches are not sufficient to detect network based intrusions. In order to facilitate the detection of network intrusions outside the guest machine, MNPD performs the network traffic analysis of tenant' network. MNPD ensures the security from network intrusions by monitoring the tenant virtual network traffic with two-levels of security check. It provides the primary security from attackers, targeting Network-layer of Cloud Network Server in cloud. The secondary security check detects the attacks against virtual domains at VMM-layer of Cloud Compute Server. It leverages the network introspection features to gain the VM related information using open source tools such as Libvirt [77], XenStore [191], dnsmasq server [192] from Dom0 of hypervisor. The information is later used to validate the network traffic using proposed algorithm. The non-spoofed packets are further analyzed passed to network behavior analyzer (NBA) to learn and detect any abnormality in the virtual traffic. NBA learns the attack patterns in learning phase using Random Forest classifier. The trained model is used to detect any suspicious network pattern in detection phase. MNPD runs two popular feature selection approaches i.e. Chi Square and Recursive Feature Elimination (RFE) methods [194] in parallel and combines their results to transform the original dataset into new dataset. This reduces the dimension of dataset and removes the less relevant features which improve the classifiers' performance. Alerts and logs are sent to cloud administrator if any suspicious activity is detected by NBA detection component. Only legitimate packets are allowed to pass from physical interface of CCoS and forwarded to other servers (CNS/other CCoS). MNPD does not incur overhead in monitoring extensive memory writes or instruction-level traces. It is a more secure solution to detect attacks which never pass through physical interface and hence are not detected by traditional IDS. Secondly, it is difficult to get compromised by malicious tenant users as it is deployed outside the TVM at both the VMM and Network layer. We have compared the performance of the proposed MNPD approach with other approaches in cloud environment: ecloudIDS [139], NIDS [90], CIDS [87] and 'Hypervisor Detector' [126]. Most of the IDS techniques have been validated with KDD99. As the evaluation is based on very older dataset, no estimation of the real performance of the system can be assessed. MNPD has been validated with UNSW-NB and ITOC datasets. It provides an accuracy of 98.88% using ITOC dataset and 95.091% using UNSW-NB dataset which are better than the existing approaches.

In summary, this thesis presented a threat model, attack taxonomy, classification of IDS with detailed analysis of their detection mechanism for detecting attacks that are possible in the cloud. A set of key research challenges have been identified. To address some of these challenges, we proposed a comprehensive intrusion detection framework called CloudHedge with three lines of defense against different types of attacks in the cloud. We have validated the proposed techniques using different datasets and the results seem to be promising.

7.2 Scope for Future Work

The design of CloudHedge overcomes the limitations associated with the existing security proposals. However, a number of issues crop up in the working of CloudHedge. Some of them are as follows:

- MSCSD is currently controlled by cloud administrator, for monitoring system specific and configuration files of TVMs. However, in future MSCSD can be provided as a service to the tenants for monitoring their user space files which will be completely under control of tenants.
- CloudHedge can also be improved by making use of the clustering approaches for intrusion detection. Clustering approaches (unsupervised learning) can be used to detect the unseen attack patterns.

* MNPD can also be improved by taking the advantage of both classification and clustering approaches to detect the attacks in cloud.
* Correlation of alerts from IDS instances deployed at different layers can be done using appropriate mechanism to detect the distributed attacks in cloud.
* In this stage, CloudHedge is designed to detect attack launched by malicious tenants. Cloud Hedge does not provide the detection of attacks from malicious cloud service provider. However, CloudHedge can be extended to detect these types of attacks in future.
* Efficient hypervisor security solutions can also be integrated with the existing frameworks by incorporating hypervisor introspection approaches to address the attacks against VMM.
* Severity of attacks can also be decided based on the some appropriate mechanism for anomalous score calculation based on user's behavior profile.
* Hypercall analysis can also be done at the hypervisor-layer to improve the attack detection accuracy of IDS.

Appendix A

List of Publications

▪ Publications in International Journals

1. P. Mishra, E. S. Pilli, V. Varadharajan, and U. Tupakula, "Intrusion detection techniques in cloud environment: A survey," *Journal of Network and Computer Applications*, Elsevier, vol. 77, pp. 18-47, 2017 (SCI Expanded with 3.5 (IF)).
2. P. Mishra, E. S. Pilli, V. Varadharajan, and U. Tupakula, "VAED: VMI-assisted evasion detection approach for infrastructure as a service cloud," *Concurrency and Computation: Practice and Experience*, Wiley, vol. 29, no. 12, pp. 1-21, 2017, (SCI Expanded with 1.133 (IF)).
3. P. Mishra, V. Varadharajan, E. S. Pilli, and U. Tupakula, "VMGuard: VMI-assisted security architecture for intrusion detection in cloud environment," *IEEE Transactions on Cloud Computing*, pp 1-14, 2017 [Under minor revision].
4. P. Mishra, V. Varadharajan, E. S. Pilli, and U. Tupakula, "Detailed Investigation and Analysis of using Machine Learning Techniques for Intrusion Detection," *IEEE Communication Surveys and Tutorials*, pp. 1-35, 2017, (SCI with 17.177 (IF)) [Submitted revised version].

▪ Publications in International Conferences

1. P. Mishra, E. S. Pilli, V. Varadharajan, and U. Tupakula, "Out-VM monitoring for malicious network packet detection in cloud environment," in *IEEE ISEA Asia Security and Privacy Conference*, IEEE, NIT Surat, India, Jan 2017, pp. 1-10.

2. P. Mishra, E. S. Pilli, V. Varadharajan, and U. Tupakula, "Securing Virtual Machines from Anomalies Using Program-Behavior Analysis in Cloud Environment," in *IEEE 18th International Conference on High Performance Computing and Communications (HPCC 2016)*, Sydney, Australia, Dec 2016, pp. 991-998.
3. P. Mishra, E. S. Pilli, V. Varadharajan, and U. Tupakula, "Efficient approaches for intrusion detection in cloud environment," in *IEEE International Conference on Computing, Communication and Automation (ICCCA 2016)*, Greater Noida, India, 2016, pp. 1211-1216.

■ Other Publications (Journal/Conference)

1. P. Mishra, E. S. Pilli, V. Varadharajan, and U. Tupakula, "PSI-NetVisor: Program Semantic Aware Intrusion Detection at Network and Hypervisor Layer in Cloud," Journal of Intelligent and Fuzzy Systems, vol. 32, no. 4, pp. 2909-2921, 2017. [SCI Expanded with 1.261 IF]
2. P. Mishra, E. S. Pilli, V. Varadharajan, and U. Tupakula, "NvCloudIDS: A Security Architecture to Detect Intrusions at Network and Virtualization Layer in Cloud Environment," in *IEEE 5th International Conference on Advances in Computing, Communications and Informatics (ICACCI 2016)*, Jaipur, India, Sept 2016, pp. 56-62.

References

[1] S. Garfinkel, *Architects of the Information Society: 35 Years of the Laboratory for Computer Science at MIT*, 1st ed. Cambridge, Massachusetts: MIT Press, 1999.

[2] M. Bakery and R. Buyya, "Cluster computing at a glance," *High Performance Cluster Computing: Architectures and Systems*, vol. 1, pp. 3–47, 1999.

[3] M. Chetty and R. Buyya, "Weaving computational Grids: How analogous are they with electrical Grids?" *Computing in Science & Engineering*, vol. 4, no. 4, pp. 61–71, 2002.

[4] R. Chellappa, "Intermediaries in Cloud-Computing: A New Computing Paradigm," 1997. [Online]. Available: http://meetings2.informs.org/DAL97/TALKS/C20.html

[5] NIST, "NIST cloud computing standards roadmap," National Institute of Standards and Technology, Tech. Rep. 500-291, 2011.

[6] S. Subashini and V. Kavitha, "A survey on security issues in service delivery models of cloud computing," *J. of Network and Computer Applications*, vol. 34, no. 1, pp. 1–11, 2011.

[7] A. Sinha, "Cloud Computing in Libraries: Opportunities and Challenges," *Pearl: A J. of Library and Information Science*, vol. 10, no. 2, pp. 113–118, 2016.

[8] J. Sugerman, G. Venkitachalam, and B.-H. Lim, "Virtualizing i/o devices on vmware workstation's hosted virtual machine monitor." in *USENIX Annual Technical Conference*, 2001, pp. 1–14.

[9] M. Pearce, S. Zeadally, and R. Hunt, "Virtualization: Issues, security threats, and solutions," *ACM Computing Surveys (CSUR)*, vol. 45, no. 2, p. 17, 2013.

[10] 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," in *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, 2003, pp. 164–177.

[11] Microsoft, *Using Windows 8 Client Hyper-V*, 2017. [Online]. Available: https://www.microsoft.com/en-in/download/details.aspx?id=36188

[12] VMware, *VMware vSphere*, 2017. [Online]. Available: https://my.vmware.com/en/web/vmware/info/slug/datacenter_cloud_infrastructure/vmware_vsphere/6_5

[13] vmware, *Download VMware Workstation Pro 12.5*, 2017, Available:http://www.vmware.com/products/workstation/workstation-evaluation.html.

[14] Orcle, *Download Oracle VirtualBox*, 2017, Available:https://www.virtualbox.org/wiki/Downloads.

[15] D. Milojičić, I. M. Llorente, and R. S. Montero, "Opennebula: A cloud management tool," *IEEE Internet Computing*, vol. 15, no. 2, pp. 11–14, 2011.

[16] Openstack, *OpenStack: Open Source Cloud Computing Software*, 2015. [Online]. Available: https://www.openstack.org/software/

[17] T. A. S. Foundation, *Apache CloudStack: Downloads*, 2017. [Online]. Available: https://cloudstack.apache.org/downloads.html

[18] Citrix, *Download - XenServer*, May 2016. [Online]. Available: http://xenserver.org/open-source-virtualization-download.html

[19] H. P. Enterprise, *Download HPE Helion Eucalyptus*, April 2016. [Online]. Available: http://www8.hp.com/in/en/cloud/helion-eucalyptus-downloads.html#C4

[20] R. Wojtczuk, "Subverting the xen hypervisor," *Black Hat USA*, vol. 2008, pp. 1–9, 2008.

[21] CloudVPS, *The open cloud*, 2011. [Online]. Available: https://www.cloudvps.com/openstack

[22] J.-M. Kim, H.-Y. Jeong, I. Cho, S. M. Kang, and J. H. Park, "A secure smart-work service model based OpenStack for cloud computing," *Cluster computing*, vol. 17, no. 3, pp. 691–702, 2014.

[23] VEXXHOST, *VEXXHOST: OpenStack Public Cloud*, 2015. [Online]. Available: https://vexxhost.com/public-cloud

[24] AURO, *AURO Public Cloud*, 2016. [Online]. Available: https://auro.io/public_cloud_hosting/overview

[25] Rackspace, *Introducing the Rackspace cloud*, 2015. [Online]. Available: https://developer.rackspace.com/docs/user-guides/infrastructure/cloud-intro/

[26] M. Kelly, *DataCentred World First: OpenStack Public Cloud on 64-bit ARM Servers*, 2014. [Online]. Available: http://www.datacentred.co.uk/news/datacentred-world-first-openstack-public-cloud-on-64-bit-arm-servers/

[27] Elastx, *About Elastx*, 2012. [Online]. Available: https://elastx.se/en/about-elastx

[28] OpenStack, *OpenStack Marketplace: Dualtec Public Cloud*, 2011. [Online]. Available: https://www.openstack.org/marketplace/public-clouds/uol-diveo/dualtec-public-cloud

[29] Internap, *AgileCLOUD: High-performance cloud Infrastructure-as-a-Service powered by OpenStack*, 2016. [Online]. Available: http://www.internap.com/cloud/public-cloud/

[30] Rackspace, *Rackspace Private Cloud Powered by OpenStack*, 2017. [Online]. Available: https://www.rackspace.com/en-in/openstack/private/openstack

[31] B. B. Group, *IBM Blue Box Continues its Path Toward Keeping Your Cloud Environment Safe*, Oct 2016. [Online]. Available: https://www.blueboxcloud.com/insight/blog-article/ibm-blue-box-continues-its-path-toward-keeping-your-cloud-environment-safe

[32] Platform9, *Platform9 Announces Hybrid Cloud-as-a-Service for OpenStack Community*, Oct 2016. [Online]. Available: https://platform9.com/press/openstack-hybrid/

[33] L. et al., "Cloud Adoption and Security in India Survey Report," Cloud Security Alliance, Tech. Rep., Nov 2016.

[34] M. Mimiso, *Virtual Machine Escape Exploit Targets Xen*, Sept 2012. [Online]. Available: http://threatpost.com/virtual-machine-escape-exploit-targets-xen-090612/76979

[35] M. Dekker, D. Liveri, and M. Lakka, "Cloud Security Incident Reporting, Framework for Reporting about Major Cloud Security Incidents," ENISA, Tech. Rep. doi:10.2788/14231, Dec 2013.

[36] J. Dee, *Amazon cloud infested with DDoS botnets*, July 2014. [Online]. Available: http://www.techwalls.com/amazon-cloud-infested-ddos-botnets/

[37] www.cyber edge.com, "Cyber threat defense report," CYBEREDGE Group, Tech. Rep., 2014.

[38] J. Bourne, *Code Spaces RIP: Code hosting provider ceases trading after well orchestratedDDoS attack*, May 2014. [Online]. Available: http://www.cloudcomputing-news.net/news/2014/jun/19/code-spaces-rip-code-hosting-provider-ceases-trading-after-well-orchestrated-ddos-at

[39] Symantec, "Internet Security Threat Report," Tech. Rep., April 2016.

[40] ——, "Symantec Intelligence Report," Tech. Rep., June 2015.

[41] Verizon, *Data Breach Investigations Report (DBIR)*, 2015. [Online]. Available: https://www.verizonenterprise.com/DBIR/2015/

[42] Cisco, *Cisco Annual Security Report*, 2015. [Online]. Available: https://www.cisco.com/web/offergist_ty2_asset/Cisco_2015_ASR.pdf

[43] CISCO, "Cisco Annual Cyber Security Report," Tech. Rep., 2017.

[44] CloudPassage, "Cloud Security Spotlight Report," Tech. Rep., 2016.

[45] S. VivinSandar and S. Shenai, "Economic denial of sustainability (EDoS) in cloud services using available: HTTP and XML based ddos attacks," *International Journal of Computer Applications*, vol. 41, no. 20, 2012.

[46] Z. Xiao and Y. Xiao, "Security and privacy in cloud computing," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 2, pp. 843–859, 2013.

[47] S. R. Krishna and B. P. Rani, "Virtualization Security Issues and Mitigations in Cloud Computing," in *2nd Int. Conf. on Computational Intelligence and Informatics, Hyderabad, India*, 2017, pp. 117–128.

[48] C. Metz, *Amazon outage spans clouds 'insulated' from each other*, Apr 2011, Available:http://www.theregister.co.uk/2011/04/21/amazon_web_services_outages_spans_zones/.

[49] J. Li, X. Chen, M. Li, J. Li, P. P. Lee, and W. Lou, "Secure deduplication with efficient and reliable convergent key management," *IEEE Trans. on Parallel and Distributed Systems*, vol. 25, no. 6, pp. 1615–1625, 2014.

[50] E. Maler, P. Mishra, and R. Philpott, "Assertions and Protocol for the Oasis Security Assertion Markup Language (SAML)," *OASIS*, 2003.

[51] U. Habiba, R. Masood, M. A. Shibli, and M. A. Niazi, "Cloud identity management security issues & solutions: a taxonomy," *Complex Adaptive Systems Modeling*, vol. 2, no. 1, pp. 1–37, 2014.

[52] E. E. Mon and T. T. Naing, "The privacy-aware access control system using attribute-and role-based access control in private cloud," in *4th IEEE Int. Conf. on Broadband Network and Multimedia Technology (IC-BNMT), Beijing, China*, 2011, pp. 447–451.

[53] CSA, "The treacherous 12: Cloud computing top threats in 2016," Cloud Security Alliance, Tech. Rep. 500-291, Feb 2016.

[54] R. Chandramouli, M. Iorga, and S. Chokhani, "Cryptographic key management issues and challenges in cloud services," Tech. Rep. 7956, 2013.

[55] C. Modi, D. Patel, B. Borisaniya, A. Patel, and M. Rajarajan, “A survey on security issues and solutions at different layers of cloud computing,” *The J. of Supercomputing*, vol. 63, no. 2, pp. 561–592, 2013.

[56] ISO, *ISO 38500 IT Governance Standard*, May 2008. [Online]. Available: http://www.38500.org/

[57] ISACA, *COBIT 5: Regulatory and Compliance*, april 2012. [Online]. Available: https://cobitonline.isaca.org/

[58] S. Cordero, *Cloud Controls Matrix Working Group*, June 2016. [Online]. Available: https://cloudsecurityalliance.org/group/cloud-controls-matrix/

[59] HHS.gov, *Health Information Privacy: Enforcement Highlights*, August 1996. [Online]. Available: https://www.hhs.gov/hipaa/index.html

[60] L. PCI Security Standards Council, *PCI Security*, 2006. [Online]. Available: https://www.pcisecuritystandards.org/pci_security/

[61] NIST, *Federal Information Security Modernization Act (FISMA) Implementation*, 2014. [Online]. Available: http://csrc.nist.gov/groups/SMA/fisma/index.html

[62] soxlaw.com, *The Sarbanes-Oxley Act*, 2002. [Online]. Available: https://www.ftc.gov/tips-advice/business-center/privacy-and-security/gramm-leach-bliley-act

[63] M. K. Srinivasan, K. Sarukesi, P. Rodrigues, M. S. Manoj, and P. Revathy, “State-of-the-art cloud computing security taxonomies: a classification of security challenges in the present cloud computing environment,” in *Int. Conf. on Advances in Computing, Communications and Informatics, Chennai, India*, 2012, pp. 470–476.

[64] D. Barbara and S. Jajodia, *Applications of data mining in computer security*, 1st ed. Springer Science & Business Media, 2002, vol. 6.

[65] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, “Anomaly-based network intrusion detection: Techniques, systems and challenges,” *computers & security*, vol. 28, no. 1, pp. 18–28, 2009.

[66] Y. Hebbal, S. Laniepce, and J.-M. Menaud, “Virtual Machine Introspection: Techniques and Applications,” in *10th Int. Conf. on Availability, Reliability and Security (ARES), Toulouse, France*, 2015, pp. 676–685.

[67] S. S. Alarifi and S. D. Wolthusen, “Detecting anomalies in IaaS environments through virtual machine host system call analysis,” in *Int.*

Conf. for Internet Technology And Secured Transactions, London, UK, 2012, pp. 211–218.

[68] M. I. Sharif, W. Lee, W. Cui, and A. Lanzi, "Secure in-vm monitoring using hardware virtualization," in *16th ACM conf. on Computer and communications security, Chicago, Illinois, USA*, 2009, pp. 477–487.

[69] C. Maiero and M. Miculan, "Unobservable intrusion detection based on call traces in paravirtualized systems," in *Int. Conf. on Security and Cryptography, Seville, Spain*, 2011, pp. 300–306.

[70] X. Jiang, X. Wang, and D. Xu, "Stealthy malware detection through VMM-based out-of-the-box semantic view reconstruction," in *14th ACM conf. on Computer and communications security, New York, NY, USA*, 2007, pp. 128–138.

[71] I. Gul and M. Hussain, "Distributed cloud intrusion detection model," *Int. J. of Advanced Science and Technology*, vol. 34, no. 38, p. 135, 2011.

[72] S. Bharadwaja, W. Sun, M. Niamat, and F. Shen, "Collabra: A Xen hypervisor based collaborative intrusion detection system," in *8th Int. conf. on Information technology: New generations (ITNG), Las Vegas, Nevada, USA*, 2011, pp. 695–700.

[73] S. Gupta and P. Kumar, "System cum Program-Wide Lightweight Malicious Program Execution Detection Scheme for Cloud," *Information Security J.: A Global Perspective*, vol. 23, no. 3, pp. 86–99, 2014.

[74] C. C. Lo, C. C. Huang, and J. Ku, "A cooperative intrusion detection system framework for cloud computing networks," in *39th Int. Conf. on Parallel Processing Workshops (ICPPW), San Diego, CA*, 2010, pp. 280–284.

[75] A. Dinaburg et al., "Ether: Malware Analysis via Hardware Virtualization Extensions," in *15th ACM Conf. on Computer and communications security, VA, USA*, 2008, pp. 51–62.

[76] H. Xiong, Z. Liu, W. Xu, and S. Jiao, "Libvmi: a library for bridging the semantic gap between guest os and vmm," in *12th Int. Conf. on Computer and Information Technology (CIT), Chengdu, Sichuan, China*, 2012, pp. 549–556.

[77] R. Hat, *Libvirt Virtualization API*, 2016. [Online]. Available: http://libvirt.org/news.html

[78] B. D. Payne, M. De Carbone, and W. Lee, "Secure and flexible monitoring of virtual machines," in *Computer Security Applications Conference, Florida, USA*, 2007, pp. 385–397.

[79] S. Kim, B. Chanana *et al.*, "An Introduction to VMsafe: Architecture, Performance, and Solutions," 2009. [Online]. Available: https://www.vmworld.com/docs/DOC-2406

[80] C. Schneider, J. Pfoh, and C. Eckert, "A universal semantic bridge for virtual machine introspection," in *Information Systems Security*, 2011, pp. 370–373.

[81] C. N. Modi, D. R. Patel, A. Patel, and R. Muttukrishnan, "Bayesian Classifier and snort based network intrusion detection system in cloud computing," in *3rd Int. Conf. on Computing Communication & Networking Technologies (ICCCNT)*, 2012, pp. 1–7.

[82] S. Roschke et al., "Intrusion Detection in the Cloud," in *IEEE 8th Int. Conf. on Dependable, Autonomic and Secure Computing, Berkeley, CA*, pp. 729–734.

[83] C. Modi and D. Patel, "A novel hybrid-network intrusion detection system (H-NIDS) in cloud computing," in *IEEE Symposium on Computational Intelligence in Cyber Security (CICS), Singapore, Southern Malaysia*, April 2013, pp. 23–30.

[84] S. Gupta and P. Kumar, "An Immediate System Call Sequence Based Approach for Detecting Malicious Program Executions in Cloud Environment," *Wireless Personal Communications*, vol. 81, no. 1, pp. 405–425, 2015.

[85] S. Forrest, S. Hofmeyr, A. Somayaji, T. Longstaff *et al.*, "A sense of self for unix processes," in *IEEE Symposium on Security and Privacy, Oakland, CA, USA*, 1996, pp. 120–128.

[86] Z. Li, W. Sun, and L. Wang, "A neural network based distributed intrusion detection system on cloud platform," in *IEEE 2nd Int. Conf. on Cloud Computing and Intelligent Systems (CCIS), Hangzhou, China*, 2012, pp. 75–79.

[87] S. et al., "Collaborative IDS Framework for Cloud," *Int. J. of Network Security*, vol. 18, no. 4, pp. 699–709, 2016.

[88] D. Baysa, R. M. Low, and M. Stamp, "Structural entropy and metamorphic malware," *J. of computer virology and hacking techniques*, vol. 9, no. 4, pp. 179–192, 2013.

[89] G. Wang, "Exploiting timing side-channels against VM monitoring," 2014.

[90] C. Modi, D. Patel, B. Borisanya, A. Patel, and M. Rajarajan, "A novel framework for intrusion detection in cloud," in *5th Int. Conf. on Security of Information and Networks, Jaipur, India*, 2012, pp. 67–74.

[91] C. Benninger et al., "Maitland: Lighter-Weight VM Introspection to Support Cyber-Security in the Cloud," in *5th Int. Conf. on Cloud Computing, Hawaii, USA*, 2012, pp. 471–478.

[92] Y. Fu and Z. Lin, "Space traveling across VM: Automatically bridging the semantic gap in virtual machine introspection via online kernel data redirection," in *IEEE Symposium on Security and Privacy, San Francisco, CA*, 2012, pp. 586–600.

[93] R. Wu, P. Chen, P. Liu, and B. Mao, "System Call Redirection: A Practical Approach to Meeting Real-World Virtual Machine Introspection Needs," in *44th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks, Atlanta, GA, USA*, 2014, pp. 574–585.

[94] B. Grobauer, T. Walloschek, and E. Stöcker, "Understanding cloud computing vulnerabilities," *IEEE Security & privacy*, vol. 9, no. 2, pp. 50–57, 2011.

[95] M. M. S. M. Habib, V. Varadharajan, "A framework for evaluating trust of service providers in cloud marketplaces," in *28th ACM Symp. Applied Computing, Coimbra, Portugal*, 2013, pp. 1–3.

[96] M. Lindorfer, C. Kolbitsch, and P. M. Comparetti, "Detecting environment-sensitive malware," in *Int. Workshop on Recent Advances in Intrusion Detection, CA, USA*, 2011, pp. 338–357.

[97] Y. Fratantonio, C. Kruegel, and G. Vigna, "Shellzer: a tool for the dynamic analysis of malicious shellcode," in *Int. Workshop on Recent Advances in Intrusion Detection, Salzburg, Austria.* Springer, 2011, pp. 61–80.

[98] G. Pék et al., "nEther: in-guest detection of out-of-the-guest malware analyzers," in *4th European Workshop on System Security, Salzburg, Austria*, 2011, p. 3.

[99] AMD, "Cpuid specification," Tech. Rep., Sept 2014.

[100] T. Hwang, Y. Shin, K. Son, and H. Park, "Design of a hypervisor-based rootkit detection method for virtualized systems in cloud computing environments," in *AASRI Winter International Conference on Engineering and Technology (AASRI-WIET), Saipan, Northern Mariana Islands*, 2013, pp. 27–32.

[101] V. Varadharajan and U. Tupakula, "Security as a Service Model for Cloud Environment," *IEEE Trans. on Network and Service Management*, vol. 11, no. 1, pp. 60–75, March 2014.

[102] S. Sanfilippo, *HPING 3*, 2005. [Online]. Available: http://www.hping.org/hping3.html

[103] G. F. Lyon, *Nmap network scanning: The official Nmap project guide to network discovery and security scanning.* Nmap Project, 2009.

[104] S. Gupta and P. Kumar, "Taxonomy of cloud security," *Int. J. of Computer Science, Engineering and Applications*, vol. 3, no. 5, 2013.

[105] P. Ferrie, "Attacks on more virtual machine emulators," *Symantec Technology Exchange*, vol. 55, 2007.

[106] M. Mulazzani, S. Schrittwieser, M. Leithner, M. Huber, and E. Weippl, "Dark clouds on the horizon: Using cloud storage as attack vector and online slack space," in *20th USENIX Security Symposium, San Francisco, CA*, 2011, pp. 1–11.

[107] A.-S. K. Pathan, *The State of the Art in Intrusion Prevention and Detection.* Boca Raton, Florida: CRC Press, 2014.

[108] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Int. Joint Conf. on Neural Networks*, 1989, pp. 593–605.

[109] J. R. Quinlan, *C4.5: programs for machine learning.* Elsevier, 2014.

[110] C. Cortes and V. Vapnik, "Support vector machine," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[111] P. Kumar, N. Nitin, V. Sehgal, K. Shah, S. S. P. Shukla, and D. S. Chauhan, "A novel approach for security in cloud computing using hidden markov model and clustering," in *World Congress on Information and Communication Technologies (WICT)*, 2011, pp. 810–815.

[112] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.

[113] P. Casas, J. Mazel, and P. Owezarski, "Unsupervised network intrusion detection systems: Detecting the unknown without knowledge," *Computer Communications*, vol. 35, no. 7, pp. 772–783, 2012.

[114] J. Yang, T. Deng, and R. Sui, "An Adaptive Weighted One-Class SVM for Robust Outlier Detection," in *Chinese Intelligent Systems Conference, Shanghai, China*, 2016, pp. 475–484.

[115] S. A. Hofmeyr, S. Forrest, and A. Somayaji, "Intrusion detection using sequences of system calls," *J. of Computer Security*, vol. 6, no. 3, pp. 151–180, 1998.

[116] C. Warrender, S. Forrest, and B. Pearlmutter, "Detecting Intrusions using System Calls: Alternative Data Models," in *IEEE Symposium on Security and Privacy, Oakland, California*, 1999, pp. 133–145.

[117] S. Alarifi and S. Wolthusen, "Anomaly Detection for Ephemeral Cloud Iaas Virtual Machines," in *7th Internaltional Network and System Security, Madrid, Spain*, 2013, pp. 321–335.

[118] S.-B. Cho and H.-J. Park, “Efficient anomaly detection by modeling privilege flows using hidden markov model,” *Computers & Security*, vol. 22, no. 1, pp. 45–55, 2003.

[119] J. Arshad, P. Townend, and J. Xu, “A Novel Intrusion Severity Analysis Approach for Clouds,” *Future Generation Computer Systems*, vol. 29, no. 1, pp. 416–428, 2013.

[120] C. Ko, G. Fink, and K. Levitt, “Automated detection of vulnerabilities in privileged programs by execution monitoring,” in *Computer Security Applications Conference, New Orleans, Louisiana*, 1994, pp. 134–144.

[121] M. M. Masud, L. Khan, and B. Thuraisingham, “A hybrid model to detect malicious executables,” in *IEEE Int. Conf. on Communications, Glasgow, Scotland*, 2007, pp. 1443–1448.

[122] B. D. Payne, “Simplifying virtual machine introspection using libvmi,” Sandia Report, California, USA, Tech. Rep. SAND2012-7818, Nov. 2012.

[123] T. Garfinkel, M. Rosenblum *et al.*, “A Virtual Machine Introspection Based Architecture for Intrusion Detection,” in *Network and Distributed System Security Symposium, San Diego, California*, 2003, pp. 191–206.

[124] T. K. Lengyel et al., “Scalability, fidelity and stealth in the drakvuf dynamic malware analysis system,” in *30th Annual Computer Security Applications Conf., New York, NY, USA*, 2014, pp. 386–395.

[125] P. M. Chen and B. D. Noble, “When virtual is better than real [operating system relocation to virtual machines],” in *8th Workshop on Hot Topics in Operating Systems, Elmau, Germany*, 2001, pp. 133–138.

[126] N. Pandeeswari and G. Kumar, “Anomaly Detection System in Cloud Environment Using Fuzzy Clustering Based ANN,” *Mob. Netw. Appl.*, vol. 21, no. 3, pp. 494–505, Jun 2016.

[127] C. Modi, D. Patel, B. Borisaniya, H. Patel, A. Patel, and M. Rajarajan, “A survey of intrusion detection techniques in cloud,” *J. of Network and Computer Applications*, vol. 36, no. 1, pp. 42–57, 2013.

[128] D.-K. Kang, D. Fuller, and V. Honavar, “Learning classifiers for misuse and anomaly detection using a bag of system calls representation,” in *IEEE Information Assurance Workshop, New York, USA*, 2005, pp. 118–125.

[129] Q.-B. Yin, L.-R. Shen, R.-B. Zhang, X.-Y. Li, and H.-Q. Wang, “Intrusion detection based on hidden markov model,” in *Int. Conf. on Machine Learning and Cybernetics, Xi’an, China*, 2003, pp. 3115–3118.

[130] J. J. Flores, A. Antolino, J. M. Garcia, and F. C. Solorio, “Hybrid network anomaly detection–learning HMMs through evolutionary computation,” 2012.

[131] B. Ward, *The book of VMware: the complete guide to VMware workstation.* No Starch Press San Francisco, 2002, vol. 1.

[132] K. Kourai and S. Chiba, “HyperSpector: virtual distributed monitoring environments for secure intrusion detection,” in *1st ACM Int. Conf. on Virtual Execution Environments, Chicago, IL, USA*, 2005, pp. 197–207.

[133] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, “kvm: The Linux virtual machine monitor,” in *Proceedings of the Linux Symposium*, vol. 1, 2007, pp. 225–230.

[134] B. D. Payne, M. Carbone, M. Sharif, and W. Lee, “Lares: An architecture for secure active monitoring using virtualization,” in *IEEE Symposium on Security and Privacy, California, USA*, 2008, pp. 233–247.

[135] J. Pfoh et al., “Nitro: Hardware-based System Call Tracing for Virtual Machines,” in *Advances in Information and Computer Security, Tokyo, Japan*, 2011, pp. 96–112.

[136] M. Bernaschi, E. Gabrielli, and L. V. Mancini, “Remus: a security-enhanced operating system,” *ACM Trans. on Information and System Security (TISSEC)*, vol. 5, no. 1, pp. 36–61, 2002.

[137] C. Mazzariello, R. Bifulco, and R. Canonico, “Integrating a network IDS into an open source cloud computing environment,” in *6th Int. Conf. on Information Assurance and Security (IAS), Atlanta, GA, USA*, 2010, pp. 265–270.

[138] J.-H. Lee, M.-W. Park, J.-H. Eom, and T.-M. Chung, “Multi-level intrusion detection system and log management in cloud computing,” in *13th Int. Conf. on Advanced Communication Technology (ICACT), Gangwon-Do, South Korea*, 2011, pp. 552–555.

[139] M. K. Srinivasan, K. Sarukesi, A. Keshava, and P. Revathy, “eCloudIDS Tier-1 uX-Engine Subsystem Design and Implementation Using Self-Organizing Map (SOM) for Secure Cloud Computing Environment,” in *Recent Trends in Computer Networks and Distributed Systems Security, Trivandrum, India*, 2012, pp. 432–443.

[140] C. N. Modi, D. R. Patel, A. Patel, and M. Rajarajan, “Integrating Signature Apriori based Network Intrusion Detection System (NIDS) in Cloud Computing,” *Procedia Technology*, vol. 6, pp. 905–912, 2012.

[141] C.-H. Lin, C.-W. Tien, and H.-K. Pao, “Efficient and Effective NIDS for Cloud Virtualization Environment,” in *Int. Conf. on Cloud Computing Technology and Science (CloudCom), Taipei, Taiwan*, 2012, pp. 249–254.

[142] U. Tupakula, V. Varadharajan, and N. Akku, “Intrusion detection techniques for infrastructure as a service cloud,” in *IEEE 9th Int. Conf. on Dependable, Autonomic and Secure Computing (DASC), Sydney, Australia*, 2011, pp. 744–751.

[143] M. R. Watson, A. K. Marnerides, A. Mauthe, D. Hutchison *et al.*, “Malware detection in cloud computing infrastructures,” *IEEE Trans. on Dependable and Secure Computing*, vol. 13, no. 2, pp. 192–205, 2016.

[144] C.-M. Chen, D. Guan, Y.-Z. Huang, and Y.-H. Ou, “State-based attack detection for cloud,” in *IEEE Int. Symposium on Next-Generation Electronics (ISNE), Kaohsiung, Taiwan*, 2013, pp. 177–180.

[145] V. Varadharajan and U. Tupakula, “On the design and implementation of an integrated security architecture for cloud with improved resilience,” *IEEE Trans. on Cloud Computing*, vol. PP, 2016.

[146] H. A. Kholidy, A. Erradi, S. Abdelwahed, and F. Baiardi, “Ha-cids: A hierarchical and autonomous ids for cloud systems,” in *5th Int. Conf. on Computational Intelligence, Communication Systems and Networks, Madrid, Spain*, 2013, pp. 179–184.

[147] Z. Al Haddad, M. Hanoune, and A. Mamouni, “A Collaborative Network Intrusion Detection System (C-NIDS) in Cloud Computing,” *Int. J. of Communication Networks and Information Security*, vol. 8, no. 3, p. 130, 2016.

[148] A. S. Abed, T. C. Clancy, and D. S. Levy, “Applying Bag of System Calls for Anomalous Behavior Detection of Applications in Linux Containers,” in *IEEE Globecom Workshops (GC Wkshps), San Diego, CA, USA*, Dec. 2015, pp. 1–5.

[149] S. Gupta, P. Kumar, and A. Abraham, “A profile based network intrusion detection and prevention system for securing cloud environment,” *Int. J. of Distributed Sensor Networks*, vol. 2013, no. 364575, pp. 1–12, 2013.

[150] KDD99, *KDD Cup 1999 Data*, 1999. [Online]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[151] M. Ficco, L. Tasquier, and R. Aversa, “Intrusion detection in cloud computing,” in *8th Int. Conf. on P2P, Parallel, Grid, Cloud and Internet Computing,Compiegne, France*, 2013, pp. 276–283.

[152] U. Steinberg and B. Kauer, "NOVA: a microhypervisor-based secure virtualization architecture," in *5th European conf. on Computer systems, Paris, France*, 2010, pp. 209–222.

[153] F. Zhang, J. Wang, K. Sun, and A. Stavrou, "Hypercheck: A hardware-assisted integrity monitor," *IEEE Transactions on Dependable and Secure Computing*, vol. 11, no. 4, pp. 332–344, 2014.

[154] S. Gupta, P. Kumar, A. Sardana, and A. Abraham, "A fingerprinting system calls approach for intrusion detection in a cloud environment," in *4th Int. Conf. on Computational aspects of social networks (CASoN), Sao Carlos, Brazil*, 2012, pp. 309–314.

[155] UNM, *UNM Dataset*, 1998. [Online]. Available: http://www.cs.unm.edu/~immsec/systemcalls.htm

[156] Z. Deng, X. Zhang, and D. Xu, "SPIDER: Stealthy binary program instrumentation and debugging via hardware virtualization," in *29th Annual Computer Security Applications Conference, New Orleans, Louisiana*, 2013, pp. 289–298.

[157] A. Sindelar, A. Moser, J. Stuettgen, J. Sanchez, M. Cohen, and M. Bushkov, *Rekall Memory Forensic*, 2013, Available:http://www.rekall-forensic.com/.

[158] D. Kirat et al., "Barecloud: Bare-metal analysis-based evasive malware detection," in *In the proc. of the 23rd USENIX Security Symposium, San Diego, CA*, 2014, pp. 287–301.

[159] N. Ye, X. Li, and S. M. Emran, "Decision Tree for Signature Recognition and State Classification," in *IEEE Systems, Man and Cybernetics Information Assurance & Security Workshop, New York, USA*, 2000, pp. 194–199.

[160] T. M. Mitchell, "Machine learning," 1997.

[161] A. H. Bhat et al., "Machine Learning Approach for Intrusion Detection on Cloud Virtual Machines," *Int. J. of Application or Innovation in Engineering & Management*, vol. 2, no. 6, pp. 56–66, 2013.

[162] Amazon, *Amazon EC2 - Virtual Server Hosting*, 2011, available: https://aws.amazon.com/ec2/.

[163] L. Youseff, M. Butrico, and D. Da Silva, "Toward a Unified Ontology of Cloud Computing," in *Grid Computing Environments Workshop, Austin, Texax*, 2008, pp. 1–10.

[164] T. K. Lengyel, *Stealthy monitoring with Xen altp2m*, 2016. [Online]. Available: https://blog.xenproject.org/2016/04/13/stealthy-monitoring-with-xen-altp2m/#comments

[165] B. Anderson, D. Quist, J. Neil, C. Storlie, and T. Lane, “Graph-based malware detection using dynamic analysis,” *J. in Computer Virology*, vol. 7, no. 4, pp. 247–258, 2011.

[166] R. Jones, *libguestfs: Tools for accessing and modifying virtual machine disk images*, 2016. [Online]. Available: http://libguestfs.org/

[167] D. Yuxin et al., “Feature representation and selection in malicious code detection methods based on static system calls,” *Computers & Security*, vol. 30, no. 6, pp. 514–524, 2011.

[168] L. Breiman, “Random forests,” *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[169] R. Yuan et al., “An SVM-based machine learning method for accurate internet traffic classification,” *Information Systems Frontiers*, vol. 12, no. 2, pp. 149–156, 2010.

[170] D. Herrmann et al., “Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier,” in *ACM workshop on Cloud computing security, hicago, Illinois, US*, 2009, pp. 31–42.

[171] X. Li et al., “AdaBoost with SVM-based component classifiers,” *Engineering Applications of Artificial Intelligence*, vol. 21, no. 5, pp. 785–795, 2008.

[172] W. R. Gilks, *Markov chain monte carlo.* New York City, United States: Wiley Online Library, 2005.

[173] F. Karbalaie, A. Sami, and M. Ahmadi, “Semantic malware detection by deploying graph mining,” *Int. J. of Computer Science Issues*, vol. 9, no. 1, pp. 373–379, 2012.

[174] M. Iqbal, S. U. Rehman, S. Gillani, and S. Asghar, “An Empirical Evaluation of Feature Selection Methods,” *Improving Knowledge Discovery through the Integration of Data Mining Techniques*, pp. 233–258, 2015.

[175] M. Wozniak and K. Jackowski, “Some remarks on chosen methods of classifier fusion based on weighted voting,” in *Int. Conf. on Hybrid Artificial Intelligence Systems, Salamanca, Spain*, 2009, pp. 541–548.

[176] M. Beynon, B. Curry, and P. Morgan, “The Dempster–Shafer theory of evidence: an alternative approach to multicriteria decision modelling,” *Omega*, vol. 28, no. 1, pp. 37–50, 2000.

[177] M. J. Roemer, G. J. Kacprzynski, and R. F. Orsagh, “Assessment of data and knowledge fusion strategies for prognostics and health management,” in *Aerospace Conference, Big Sky, Montana*, vol. 6, 2001, pp. 2979–2988.

[178] F. Moreno-Seco, J. M. Inesta, P. J. P. De León, and L. Micó, "Comparison of classifier fusion methods for classification in pattern recognition tasks," in *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, 2006, pp. 705–713.

[179] S. Dua and X. Du, *Data mining and machine learning in cybersecurity.* CRC press, 2016.

[180] D. T. Larose, "Decision trees," *Discovering Knowledge in Data: An Introduction to Data Mining*, pp. 107–127, 2004.

[181] S. Raschka, *Python Machine Learning.* Packt Publishing Ltd, 2015.

[182] H. He, Y. Bai, E. A. Garcia, and S. Li, "ADASYN: Adaptive synthetic sampling approach for imbalanced learning," in *IEEE Int. Joint Conf. on Neural Networks (IEEE World Congress on Computational Intelligence), Hong Kong, CHINA*, 2008, pp. 1322–1328.

[183] N. V. Chawla, A. Lazarevic, L. O. Hall, and K. W. Bowyer, "SMOTEBoost: Improving prediction of the minority class in boosting," in *European Conference on Principles of Data Mining and Knowledge Discovery, Cavtat-Dubrovnik, Croatia*, 2003, pp. 107–119.

[184] H. Guo and H. L. Viktor, "Learning from imbalanced data sets with boosting and data generation: the databoost-im approach," *ACM Sigkdd Explorations Newsletter*, vol. 6, no. 1, pp. 30–39, 2004.

[185] T. Hastie, R. Tibshirani, J. Friedman, and J. Franklin, "The Elements of Statistical Learning: Data Mining, Inference and Prediction," *The Mathematical Intelligencer*, vol. 27, no. 2, pp. 83–85, 2005.

[186] Cisco, "Cisco annual security report," 2014.

[187] J. Dee, *Amazon cloud infested with DDoS botnets*, July 2014. [Online]. Available: http://www.techwalls.com/amazon-cloud-infested-ddos-botnets/

[188] H. B. Baraka and H. Tianfield, "Intrusion Detection System for Cloud Environment," in *7th Int. Conf. on Security of Information and Networks, Glasgow, Scotland, UK*, 2014, pp. 1–6.

[189] N. Moustafa and J. Slay, "UNSW-NB15: A Comprehensive Data Set for Network Intrusion Detection Systems (UNSW-NB15 Network Data Set)," in *Military Communications and Information Systems Conference (MilCIS), Canberra, Australia*, 2015, pp. 1–6.

[190] C. Harrison, D. Cook, R. McGraw, and J. A. Hamilton Jr, "Constructing a Cloud-based IDS by Merging VMI with FMA," in *11th Int. Conf. on Trust, Security and Privacy in Computing and Communications, Liverpool, United Kingdom*, 2012, pp. 163–169.

[191] V. Hanquez, *The XenStore package*, 2013. [Online]. Available: https://hackage.haskell.org/package/xenstore

[192] S. Kelley, *dnsmasq*, 2016. [Online]. Available: http://www.thekelleys.org.uk/dnsmasq/docs/dnsmasq-man.html

[193] F. Sebastiani, "Machine learning in automated text categorization," *ACM computing surveys (CSUR)*, vol. 34, no. 1, pp. 1–47, 2002.

[194] I. Guyon, S. Gunn, M. Nikravesh, and L. A. Zadeh, *Feature Extraction: Foundations and Applications.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, vol. 207.

[195] D. Chisnall, *The Definitive Guide to the Xen Hypervisor.* Pearson Education, 2008.

[196] Xenproject.org, *OpenStack: XAPI*, 2015. [Online]. Available: https://wiki.xenproject.org/wiki/OpenStack

[197] V. Jacobson, C. Leres, and S. McCanne, *TCPDUMP public repository*, 2003. [Online]. Available: http://www.tcpdump.org

[198] B. Merino, *Instant Traffic Analysis with Tshark How-to.* Birmingham, UK: Packt Publishing Ltd, 2013.

[199] D. Arndt, *NetMate-flowcalc*, 2013. [Online]. Available: Available: https://dan.arndt.ca/projects/netmate-flowcalc/

[200] C. C. Aggarwal and C. Zhai, *Mining text data.* New York, USA: Springer Science & Business Media, 2012.

[201] B. Sangster, T. O'Connor, T. Cook, R. Fanelli, E. Dean, C. Morrell, and G. J. Conti, *ITOC dataset*, 2009. [Online]. Available: http://www.itoc.usma.edu/research/dataset/

[202] ACCS, *The UNSW-NB15 data set description*, 2015. [Online]. Available: https://www.unsw.adfa.edu.au/australian-centre-for-cyber-security/cybersecurity/ADFA-NB15-Datasets/

[203] G. Lyon, *Nmap: The Network Mapper*, 1997. [Online]. Available: https://github.com/nmap/nmap

[204] N. Moustafa and J. Slay, "The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set)," in *Information Security J.: A Global Perspective*, vol. 25, 2016, pp. 18–31.

[205] M. Panda and M. R. Patra, "Ensembling rule based classifiers for detecting network intrusions," in *Int. Conf. on Advances in Recent Technologies in Communication and Computing.* IEEE, 2009, pp. 19–22.