# SDN-based Proactive Defense Mechanism in a Cloud System

by

Chun-Jen Chung

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved July 2015 by the
Graduate Supervisory Committee:

Dijiang Huang, Chair
Gail-Joon Ahn
Guoliang Xue
Yanchao Zhang

ARIZONA STATE UNIVERSITY

August 2015

ABSTRACT

Cloud computing is known as a new and powerful computing paradigm. This new generation of network computing model delivers both software and hardware as on-demand resources and various services over the Internet. However, the security concerns prevent users from adopting the cloud-based solutions to fulfill the IT requirement for many business-critical computing. Due to the resource-sharing and multi-tenant nature of cloud-based solutions, cloud security is especially the most concern in the Infrastructure as a Service (IaaS). It has been attracting a lot of research and development effort in the past few years.

Virtualization is the main technology of cloud computing to enable multi-tenancy. Computing power, storage, and network are all virtualizable to be shared in an IaaS system. This important technology makes abstract infrastructure and resources available to users as isolated *virtual machines* (VMs) and *virtual networks* (VNs). However, it also increases vulnerabilities and possible attack surfaces in the system, since all users in a cloud share these resources with others or even the attackers. The promising protection mechanism is required to ensure strong isolation, mediated sharing, and secure communications between VMs. Technologies for detecting anomalous traffic and protecting normal traffic in VNs are also needed. Therefore, how to secure and protect the private traffic in VNs and how to prevent the malicious traffic from shared resources are major security research challenges in a cloud system.

This dissertation proposes four novel frameworks to address challenges mentioned above. The first work is a new multi-phase distributed vulnerability, measurement, and countermeasure selection mechanism based on the attack graph analytical model. The second work is a hybrid intrusion detection and prevention system to protect VN and VM using virtual machines introspection (VMI) and software defined networking (SDN) technologies. The third work further improves the previous works by introducing a VM profiler and VM Security Index (VSI) to keep track the security status of each VM and suggest the optimal countermeasure to mitigate potential threats. The final work is a SDN-based proactive defense mechanism for a cloud system using a reconfiguration model and moving target defense approaches to actively and dynamically change the virtual network configuration of a cloud system.

To my lovely and supportive family - Annabelcle, Ellsie, Cayden, and Chenchen.

## ACKNOWLEDGEMENTS

Firstly, I would like to express my sincere gratitude to my advisor Prof. Dijiang Huang for the continuous support of my Ph.D study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Gail-Joon Ahn, Prof. Guoliang Xue, and Prof. Yanchao Zhang, for their insightful comments and encouragement, but also for the hard question which incented me to widen my research from various perspectives.

I thank my fellow labmates in for the stimulating discussions, for the sleepless nights we were working together before deadlines, and for all the fun we have had in the pass few years.

Last but not the least, I would like to thank my family: my parents, my wife and to my sisters for supporting me spiritually throughout writing this thesis and my my life in general.

TABLE OF CONTENTS

CHAPTER Page

# LIST OF TABLES

LIST OF FIGURES

Figure Page

Chapter 1

# INTRODUCTION

For the past few years, cloud computing has become a new paradigm for hosting and delivering services over the Internet. In order to reduce the total cost of ownership, more and more business companies have started their business applications on the cloud and acquired infrastructure recourses from the cloud for their business operations. However, the cloud computing technology is still developing and evolving, many issues need to be addressed, especially in security. This is a good opportunity for the IT industry and academic researchers get involved to provide a better architecture and solution.

Typically, the services model provided by the cloud computing can be grouped into three categories [1][2]: *software as service* (*SaaS*), *platform as a service* (*PaaS*), and *infrastructure as a service* (*Iaas*). SaaS is a software deployment model where a provider licenses an application for cloud users as a service on demand. The application service provided by Google App and Flikr is such examples of the SaaS model. PaaS is a model of delivering a software development and computing platform as a service. It provides the facilities required to support the complete lifecycle of building and delivering web applications and services. Google Apps Engine and Microsoft Azure provide this type of services. IaaS is a model of delivering computer infrastructure as a service, a typical platform virtualization environment. Cloud users purchase computing, storage, and networking resources from a cloud service provider to form a fully outsourced service. One such example is the Amazon Elastic Compute Cloud (Amazon EC2).

These three major service models have different level of security requirements and face many different security challenges [3][4]. In the SaaS model, users' data are stored at the cloud service provider's data center, along with other data users. The security issues include data security, data segregation, data access control, identity management and user sign-on process. PaaS service model allows users or developers to build their own applications on top of the required platform. The security issues in a PaaS model include software security, vulnerability and software patch management, as well as the secured and isolated application running environment. Users in an IaaS service model have a better control over the security on the required resources. The cloud service provider only takes care of security issues up to the hypervisor. However, the virtualization security is the most important concern for a multi-tenancy environment. IaaS is the foundation of all cloud services, therefore the security and risk at this layer significantly affects the other service models.

Mobile cloud computing is a new emerging integrated platform to provide the cloud computing service for mobile users. Cloud computing is known as a new and powerful network computing paradigm and mobile cloud computing is an emerging cloud service model following the trend to extend the cloud to the edge of networks. Mobile cloud computing has not simply merged mobile computing and cloud computing technologies. It involves the interdisciplinary research on mobile computing and cloud computing [5][6]. This new generation of the network computing model can deliver both software and hardware as on-demand resources and services over the Internet. It includes numerous mobile devices that are closely associated with their users and dedicated resources in the cloud. They will be directly involved in many cloud activities that extend the cloud boundaries into the entire cyber physical system.

Attackers have shown increased sophistication in their ability to compromise VMs through the exploitation of software vulnerabilities and to control the compromised VMs as zombies [7]. Such attacks are more effective in the cloud environment since cloud users usually share computing resources, e.g., being connected through the same switch, sharing with the same data storage and file systems, even with potential attackers [3]. They usually launch stealthy scans that could also be part of normal cloud activities, and then use self-replication programs to compromise VMs. For example, attackers can explore the vulnerabilities of VMs and use them as stepping stones to deploy future attacks, e.g., Distributed Denial of Service (DDoS) or to gain unauthorized resources from the cloud.

To protect VMs in public clouds, where cloud system administrators usually have full control over the host machines, vulnerabilities can be detected and patched by the system administrator in a centralized manner. However, patching known security holes in cloud data centers, where cloud users usually have the privilege to control software installed on their managed VMs, may not work effectively and can violate the Service Level Agreement (SLA) since protecting Business continuity and service availability from service outages is one of the top concerns in cloud computing systems [8].

The above described security and service situations present a major challenge. The cloud should allow vulnerable VMsc running in the cloud system while performing stringent security monitoring and protections to guard against both potential internal and external attackers from compromising them. This cloud service model is totally different from the traditional security framework where attackers are guarded by network firewalls and host-based antivirus systems.

## 1.1 Problem Statement

Due to the resources sharing nature of the virtualization technology of cloud computing, security and privacy becomes significant concerns for cloud users to employ the cloud computing model. These issues include the threats from the specific features in the cloud environment, the physical and virtual network security problems, and the untrustable running environment of mobile devices for mobile cloud users. To solve these issues, we need to unveil the core technologies of the cloud computing  virtualization and enhance it by a better model. A dynamic and predictive defense mechanism is needed to better protect the resource usage and the network traffic for each cloud user. The trust relationship among cloud services and resources, mobile devices, and mobile users also needs to be addressed.

## 1.2 Research Goals

The goals of this dissertation are to address the security challenges described above and provide a secure, stable, trustable, and an undisturbed virtual running environment for cloud users in a cloud system. The specific subgoals include:

1. design a secure and situation-aware mobile cloud infrastructure,
2. provide a secure and trusted cloud service execution environment for the mobile cloud users,
3. design a dynamic defensive mechanism through Software Defined Networking technology to protect the cloud resources from attacks, abuse and nefarious use,
4. develop analytical models to describe and predict the behavior of both attacker and defender in the mobile cloud environment, and
5. develop security metrics and assessment models for cloud service providers and cloud users to monitor and evaluate the security status of cloud resource usage.

## 1.3 Contributions

The contributions of this dissertation are presented as follows:

- A new multi-phase distributed network intrusion detection and prevention framework called NICE [9] is proposed. It is able to capture and inspect suspicious cloud traffic without interrupting users' applications and cloud services in a virtual networking environment.

- A new hypervisor-based VM process monitor is proposed [10]. It is installed in the hypervisor of the virtualization layer in each cloud server, and is able to monitor and detect hidden malicious processes in each VM non-intrusively using Virtual Machine Introspection (VMI) and semantic reconstruction technologies.

- A comprehensive countermeasure selection algorithm is proposed to match and correlate alerts from intrusion detection engines (NICE-A [9]), vulnerabilities in the attack graph, and the signal from the VM process monitor [10].

- A new security measurement metric called VM Security Index (VSI) is proposed to measure the security status of VMs and helping to select the optimal countermeasure.

- A proactive defense mechanism for a cloud system to actively and dynamically detect and mitigate potential threats by using reconfiguration strategies and the moving target defense approaches [11].

- A distributed attack graph construction algorithm is proposed to solve the scalability issue of the traditional attack graph construction approaches.

## 1.4 Dissertation Organization

This dissertation is structured as follows. Chapter 2 discusses the background of the virtualization, cloud computing and security issues in a cloud system. Chapter 3 discusses the design and implementation of NICE framework. Chapter 4 demonstrates how the VM process monitor, a non-intrusive VMM-based monitor, detect hidden processes in each VM using VMI technology. Chapter 5 demonstrates how to use VM profiler and VSI to track the security status of VMs. Chapter 6 discusses the design and implementation of SDN-based proactive defense mechanism and the distributed attack graph construction model. Finally, chapter 7 outlines future works and chapter 8 summarizes the dissertation.

# Chapter 2

# BACKGROUND

In this chapter, the background knowledge of virtualization, virtual network architecture, and security issues in a virtual network environment are described.

## 2.1 Virtualization Technologies

Virtualization is the core technology for an IaaS service model to create a multi-tenancy environment. The service provider creates a separated virtual infrastructure resource for each tenant. These virtual computing and networking environments are called *virtual machines* (VMs) and *virtual networks* VNs respectively. Tenant users are able to run their own applications, operating systems, or system services in these logical distinct environments without being tied to any specific physical computer system. The logical operating environments that the virtualization focuses on making applications, services, and instances of an operating system portable across different physical computer systems [12]. Several types of virtualization technologies describe as follows.

### *2.1.1 Hypervisor-based Virtualization*

A *hypervisor*, also known as *virtual machine monitor* (VMM), is a small piece of software or firmware that runs on top of machine's hardware. The major functions of the hypervisor are to (1) identify, trap, and respond to the protected or privileged CPU operations made by each virtual machine, and (2) handle queuing, dispatching, and returning the results of hardware requests from virtual machines. An administrative operating system, also known as privilege domain (dom0 in the Xen hypervisor) then runs on top of the hypervisor, just like the virtual machines themselves. This administrative operating system is responsible for managing virtual machines in the same server and communicating with the hypervisor [12].

Two types of hypervisor: Type I and Type II hypervisor. *Type I* hypervisor also called *native* or *bare metal*, the hypervisor is just run above the hardware to control the hardware and to manage the guest operating system. It's also responsible for most communications between all of guest OS and the hardware. A popular instance of this type of virtualization architectures are Xen, VMWare ESX Server and Microsoft Hyper-V. *Type II* hypervisor is also called hosted hypervisor, it runs as an application within the host operating system. The host OS is responsible for providing I/O drivers and managing the *Guest OS* VMs. VMware Workstation, VMWare Server, and Virtual Box are examples of type II hypervisor-based virtualization architectures.

### *2.1.2 Full Virtualization*

Full virtualization also uses a hypervisor, but incorporates code into the hypervisor that emulates the underlying hardware when necessary, enabling unmodified operating systems to run on top of the hypervisor [12]. One of the famous full virtualization software is VMWare ESX server, which uses a customized version of Linux (known as the *Service Console*) as its administrative operating system. The hypervisor in the VMWare ESX server is called *VMKernel*, which is a thick layer hypervisor including hardware driver and VMMs for each VM.

With full virtualization, an unmodified OS hosts a user space program that emulates a machine on which the guest OS runs. This approach doesn't require the guest OS to be changed in any way. It also has the advantage that the virtualized architecture can be completely different from the host architecture. For instance, QEMU can simulate a MIPS processor on an IA-32 host and a startling array of other chips [13]. However, this level of hardware independence comes at the cost of an enormous speed penalty.

### *2.1.3 Paravirtualization*

This type of virtualization is Xen's approach. Xen provides very small and compact hypervisor layer that runs directly on top of hardware, and provides services to the virtualized OS. This thin layer of the hypervisor is different from the thick layer hypervisor of full virtualization. The Xen's design philosophy is keeping hypervisor as small as possible. There is no host and guest separation in this type of virtualization. With Xen, only the hypervisor has full privileges, and it relies on a trusted guest OS to provide hardware drivers, a kernel, and a user space. This management domain is called domain 0 (*dom0*), which uses a customized Linux kernel to support its administrative environment.

Domain 0 is the first domain be created automatically when the system boots and it has special management privileges delegated by the hypervisor. This privileged domain is uniquely distinguished as the domain that the hypervisor allows to access devices and perform control functions. All other guest domains (*domUs*) are unprivileged domains and managed by Dom0. The hardware environment for all guests is not simulated, they are executed in their own isolated domains, as if they are running on a separate system. However, the guest OS needs to be specifically modified to run in this environment. Paravirtualization can provide performance enhancements over other approaches, because the operating system modifications enable the operating system to communicate directly with the hypervisor, and thus does not incur any overhead associated with the emulation that required for the other hypervisor-based virtualization.

### *2.1.4 Kernel-level Virtualization*

This type of virtualization does not require a hypervisor, but runs a specially modified Linux kernel which contains extensions designed to manage and control multiple virtual machines each containing a guest operating system.

Examples of kernel level virtualization technologies include User Mode Linux (UML) and Kernel-based Virtual Machine (KVM). UML has been supported in the mainline Linux kernel for quite a while but requires a special build of the Linux kernel for guest operating systems and KVM was introduced in the 2.6.20 mainline Linux kernel. UML does not require any separate administrative software in order to execute or manage its virtual machines, which can be executed from the Linux command line. KVM uses a device driver in the host's kernel for communication between the main Linux kernel and the virtual machines, requires processor support for virtualization (Intel VT or AMD-v Pacifica), and uses a slightly modified QEMU process as the display and an execution container for its virtual machines. In many ways, KVM's kernel-level virtualization is a specialized version of full virtualization, where the Linux kernel serves as the hypervisor [12].

### *2.1.5 Hardware-assisted Virtualization*

This is a way of improving the efficiency of hardware virtualization. It involves employing specially-designed CPUs and hardware components that helps improve the performance of a guest environment [12]. Hypervisor-based systems such as Xen and VMWare ESX Server, and kernel-level virtualization technologies such as KVM, can take advantage of the hardware support for the virtualization. The latest generation of Intel (Intel-VT) and AMD (AMD-V) processors provide hardware-assisted virtualization. Virtual machines in a hardware virtualization environment can run unmodified operating systems because the hypervisor can use the hardware's support for virtualization to handle privileged and protected operations and hardware access requests, as well as to communicate with and manage the virtual machines.

## 2.2 Virtual Network Architectures

Network is an important resource in the computing and communication system. Cloud system is not just providing computing service or resource only, it also serves communication channels between all of VMs on the same server or external network devices. Obviously, how to secure and isolate these communication channels for each individual VM on the same server is an important issue. In this section, we review some existing virtual network architectures used by current cloud platform providers.

### *2.2.1 Linux Ethernet Bridge*

The general solution to build up a virtual network within the same cloud server is via *Linux Ethernet Bridge*. A bridge is a way to connect two Ethernet segments together in a protocol independent way. Packets are forwarded based on Ethernet address, rather than IP address (like a router). Since forwarding is done at Layer 2, all protocols can go transparently through a bridge [14]. The Linux bridge code implements a subset of the ANSI/IEEE 802.1d standard [15]. The original Linux bridging was first done in Linux 2.2 and the code for bridging has been integrated into 2.4 and 2.6 kernel series.

In a cloud system, the communication happens between different VMs on the same host or among VMs located on different hosts. These traffic will go through virtual networks in order to connect to a virtual network interface of a VM or a physical network interface of the host. In the virtual network of Xen system, the hypervisor provides two modes of virtual networks: *bridge mode* and *route mode*. In the bridge mode, the dom0 creates a software Ethernet bridge attaching to a virtual interface of a VM for forwarding the traffic. Also, it attaches the virtual interface of the VM to a physical interface on the host for the communication with external network and Internet. In the route mode, dom0 creates a routing table which includes a set of MAC and IP addresses in advance to establish a point-to-point link between dom0 and each VM. In both modes, dom0 will create a *virtual interface* (*vif*) for each network interface of VM. For example, *vif1.0* attaches to *eth0* of VM1, *vif2.1* attaches to *eth1* of VM2, and so on. (See Figure 2.1).

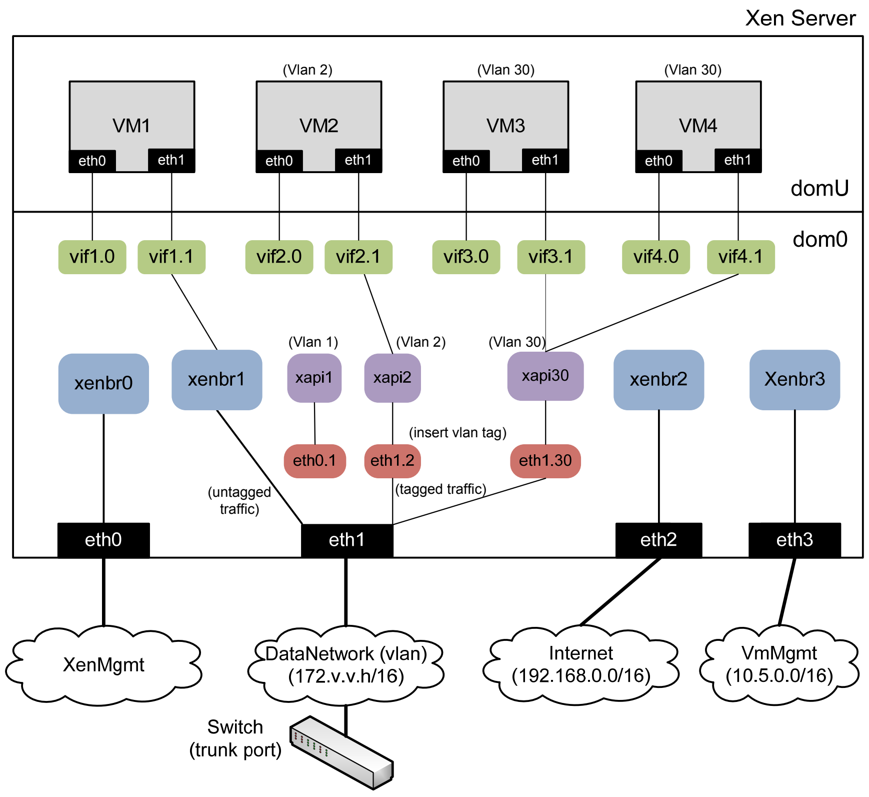


Figure 2.1: Virtual Network Configuration with Linux Bridge

### VLAN Configuration

In order to manage VMs and provide the isolated communication channels for group of VMs, virtual local area network (VLANs) is a standardize implementation in the cloud for this purpose. The VLAN configuration in a Xen system is controlled and managed by dom0. Figure 2.1 illustrates the VLAN structure in a cloud system. For each VLAN, dom0 creates a unique bridge *xapi#* and a corresponding virtual interface *eth1.#*. The symbol # means VLAN ID and *eth1* maps to the physical Ethernet port of the host. All bridges in the Xen system are based on the Linux standard bridge implementation. The *xapi*# is only for the internal network use in the dom0. When dom0 assigns VLAN for a VM, it binds the virtual interface (*vif*#.#) of the VM to the xapi bridge of the VLAN. The *vifs* of all VMs in a same VLAN connect to the same bridge. When the VLAN traffic comes to the bridge, they are forwarded to the corresponding virtual interface *eth*#.#. The virtual interface then inserts the VLAN tag in the Ethernet frame according to the 802.1Q protocol. The tagged VLAN traffic passes through the physical Ethernet port of dom0, and transmit to the trunk port of the external physical switch. If the physical switch enables the trunk port and doesn't set any native VLAN ID (default VLAN ID) or set the default VLAN ID to an unused VLAN ID, the tagged traffic will pass through the switch and remain the tag information untouched. The VLAN traffic also transmits to the same VLAN in another Xen Server or even in the remote site.

The incoming tagged VLAN traffic is processed by dom0 and the packets are passed to the corresponding virtual interface *eth*#.# based on the VLAN ID. This virtual interface will strip off the tag ID and then pass the packet to the destination only. In this way, the VLAN's assigning, tagging, and untagging are all done by dom0. VM does not know which VLAN it belongs to, and the VM also cannot modify the VLAN tag assigned to it. However, It allows an untagged traffic from a non-VLAN VM, but this type of traffic will connect to another type of bridge *xenbr1* and forward to the physical Ethernet port of dom0 directly.

The service provider usually configures the network resources with different VLANs for isolating and separating communication channels and traffic between different groups of clients in a cloud system, such as user groups of different enterprise clients. VLAN provides a mechanism for partitioning the network resources into multiple, disjoint logical broadcast domains [16]. Traffic cannot cross from one VLAN to another except through a router. However, VLAN in the cloud environment faces many vulnerabilities not only from VMs and the servers, but also the traditional network security issues.

### Pros and Cons

All of Linux-based cloud platform native support Linux Ethernet Bridge as a software switch for VMs on the same server, such as Citrix's XenServer and KVM. A Linux bridge is more powerful than a pure hardware bridge because it can also filter and shape traffic. It is easy to configure with a command `brctl`. The new features of the Linux bridge still developing, possible future enhancements are: user space STP filtering, Netlink interface to control bridges, support RSTP and other 802.1d STP extensions, and others.

However, these bridges had no network manageability features and no ability to apply policies to traffic. For this purpose, the most troublesome traffic was that between VMs on the same physical host. This traffic passed directly from VM to VM, without ever traveling over a physical wire, so network administrators had no opportunity to monitor or control it [17].

### *2.2.2 Open vSwitch*

Open vSwitch (OVS) [18] is an OpenFlow-based multilayer software switch licensed under the open source Apache 2 license. It is a default software layer switch for many cloud system implementations, for example, Citrix XenServer [19] and OpenStack [20]. OVS exposes standard control and visibility interfaces to the virtual networking layer, and was designed to support distributed across multiple physical servers. All distributed virtual switches are centralized controlled by an OpenFlow-based network controller, such as NOX/POX, RYU, and floodlight. OVS supports multiple Linux-based virtualization technologies, including Xen, XCP, KVM, and VirtualBox.

**Advanced edge switch**

Open vSwitch utilize an advanced edge switch approach to solving virtual network management problem. This approach takes advantage of the unique insight available to a hypervisor bridge, since as a hypervisor component it can directly associate network packets with VMs and their configuration. These switches add features for visibility and control formerly found only in high-end enterprise switches. They increase visibility into inter-VM traffic through standard methods such as NetFlow and mirroring. Advanced edge switches also implement traffic policies to enforce security and quality-of-service (QoS) requirements [17]. To make management of numerous switches practical, advanced edge switches support centralized policy configuration. This allows administrators to manage a collection of bridges on separate hypervisors as if it was a single distributed virtual switch. Policies and configuration centrally apply to virtual interfaces and migrate with their VMs.

**Features of Open vSwitch**

Open vSwitch is backwards-compatible with the Linux bridge, which many Linux-based hypervisors use for their edge switch. This allows it to be a drop-in replacement in many virtual environments. OVS provides many network control and monitor features. In the monitor visibility, it supports NetFlow, sFlow, Mirroring (SPAN/RSPAN/ERSPAN) protocols. OVS can be configured to send these monitoring traffic to the remote collector or analyzer. In the controllability, OVS provides centralized control through OpenFlow protocol [21]. This is the same as the OpenFlow switch controlled by NOX-based controller for the flow traffic behavior. From the OVS controller, system administrator conducts fine-grained ACLs and OoS policies. For the forwarding features, OVS supports many protocols, such as LACP (Link Aggregate Control Protocol), port bonding for load-balancing, 802.1Q VLAN model with trunk and access ports, 802.1ag connectivity fault management, and several GRE (Generic Routing Encapsulation) protocols. With the GRE tunneling for two bridges of different cloud server on different geographical location, VMs connected to these bridges can build up an end-to-end layer 2 connection, even they are located at different places and have different public IP addresses.

## 2.3 Security Issues in a Virtual Environment

In a cloud system, the possible vulnerabilities for a communication channel may come from the following components: VMs, VNs, cloud servers, and physical network connections. Below are the possible vulnerabilities can be found from these major components.

### *2.3.1 Attack from Virtual Machines*

Virtualization is a key technology in the cloud computing, one of the biggest challenges of security issues in the cloud system is that of VM interconnection [22]. The possible vulnerabilities source from VMs including (1) *VM Hopping*: a process hopping from one VM to another VM. More specifically, an attacker being on one VM can gain access over the other VM], (2) *VM Escape*: one of VM gain access over the hypervisor (VMM) and attacking the rest of other VMs. More specially, if an attacker gains access to the host running multiple VMs, the attacker can then access the resources which are shared by the other VMs, (3) *VM mobility*: the contents of the VM are stored in a file image on the hypervisor. This file image can be moved or copied to another location, the attacker can then modify the contents of this file and alter the VM's activities, and (4) *VM Template tamped*: VMs are cloned by a template for faster instantiated in a cloud system. More specifically, if the template is tamped, the attacker can alter the VM's configuration to monitor all VMs' activities.

### *2.3.2 Attack from Virtual Network*

Both bridge mode and route mode require a dom0 to play a part. In addition to the possible vulnerabilities source of the virtual network, the possibility of a compromised dom0 is another substantial security issue in the network of a cloud system. The possible vulnerabilities source from virtual network and dom0 including (1) *Sniffing*: In the bridge mode, a VM is able to sniff the virtual network on the same bridge using sniff tools such as Wireshark, (2) *Spoofing*: In the route mode, a VM can do an address Resolution Protocol (ARP) spoofing which can intercept packets to a VM and sniff the traffic of the victim VM, and (3) *Compromised dom0*: dom0 can control all of the communication among VMs and the connectivity to external network. Once dom0 is compromised, an attacker can alter the traffic within the virtual network, and steal important information from the traffic.

These security vulnerabilities may contain the following threats: (1) all VMs are monitored by attacking VMs; (2) the communication between VMs are monitored by attackers; and (3) Denial of Service (DoS) against cloud services.

### *2.3.3 VLAN Security*

Ethernet is a broadcast system. To improve the security, VLAN is implemented to strengthen network isolation as well as enhance systems management capabilities. However, with VLAN implementation, every message still can reach all parts of the same VLAN which means these messages can be read by any host on the same VLAN. This allows an attacker to passively eavesdrop packets going through the network. Also, Ethernet does not provide any mechanism to authenticate any sender's identity. This allows an attacker to generate spoofed packets or to replay earlier eavesdropped packets [23].

Most commonly known attacks against end hosts on a layer 2 network are based on Medium Access Control (MAC) Address spoofing or Address Resolution Protocol (ARP) poisoning. This forms the basis of many attacks such as the DoS and Man-In-The-Middle attacks. Other possible attacks in a VLAN-based network [24] including (1) *MAC flooding attack*: this attack is due to the limitation of the switches and bridges work. They possess a finite hardware learning table to store the source addresses of all received packets. When this table becomes full, the traffic that is directed to addresses that cannot be learned anymore will be permanently flooded, (2) *802.1Q tagging attack*: this attack is due to misconfiguration of switches that set a switch port as an unwanted trunk port. This is commonly referred to as “VLAN leaking” which allow a user on a VLAN to get unauthorized access to another VLAN, (3) *Double-Encapsulated 802.1Q/Nested VLAN attack*: this attack is due to misconfiguration of switches that set the native VLAN ID of trunk port same as the VLAN ID of the attacker’s access port. An attacker is able to encapsulate the target VLAN ID as the inner layer of tagging in 802.1Q protocol. After the outer tag is stripped off by misconfigured switch, the switch just forward the packets to the destination of the fake inner tag indicated. Therefore, the attacker on a VLAN is able to get unauthorized access to another VLAN which is referred to as “VLAN Hopping” attack, and (4) *ARP attack*: this attack is due to ARP requests and replies carry the information about layer 2 identity (MAC address) and the layer 3 identity (IP address) of a host or device and there is no verification mechanism of the correctness of these identities. The attacker is able to fool a switch into forwarding packets to a device or host in a different VLAN by sending ARP packets containing appropriately forged identities. On the other hand, within the same VLAN, the so-called ARP poisoning or ARP spoofing attacks are a very effective way to fool end stations or routers into learning counterfeited device identities. This can allow a malicious user to pose as intermediary and perform a Man-In-The-Middle attack.

Chapter 3

# NETWORK INTRUSION DETECTION AND COUNTERMEASURE SELECTION IN VIRTUAL NETWORK SYSTEMS

## 3.1 Introduction

Recent studies have shown that users migrating to the cloud consider security as the most important factor. A recent Cloud Security Alliance (CSA) survey shows that among all security issues, abuse and nefarious use of cloud computing is considered as the top security threat [25], in which attackers can exploit vulnerabilities in clouds and utilize cloud system resources to deploy attacks. In traditional data centers, where system administrators have full control over the host machines, vulnerabilities can be detected and patched by the system administrator in a centralized manner. However, patching known security holes in cloud data centers, where cloud users usually have the privilege to control software installed on their managed VMs, may not work effectively and can violate the *Service Level Agreement* (SLA). Furthermore, cloud users can install vulnerable software on their VMs, which essentially contributes to loopholes in cloud security. The challenge is to establish an effective vulnerability/attack detection and response system for accurately identifying attacks and minimizing the impact of security breach to cloud users.

In [8], M. Armbrust *et al.* addressed that protecting ”Business continuity and services availability” from service outages is one of the top concerns in cloud computing systems. In a cloud system where the infrastructure is shared by potentially millions of users, abuse and nefarious use of the shared infrastructure benefits attackers to exploit vulnerabilities of the cloud and use its resource to deploy attacks in more efficient ways [7]. Such attacks are more effective in the cloud environment since cloud users usually share computing resources, e.g., being connected through the same switch, sharing with the same data storage and file systems, even with potential attackers [3]. The similar setup for VMs in the cloud, e.g., virtualization techniques, VM OS, installed vulnerable software, networking, etc., attracts attackers to compromise multiple VMs.

We proposed a cloud-based detection and monitor framework, called NICE (Network Intrusion detection and Countermeasure sElection in virtual network systems), to establish a defense-in-depth intrusion detection framework. For better attack detection, NICE incorporates attack graph analytical procedures into the intrusion detection processes. We must note that the design of NICE does not intend to improve any of the existing intrusion detection algorithms; indeed, NICE employs a reconfigurable virtual networking approach to detect and counter the attempts to compromise VMs, thus preventing zombie VMs.

In general, NICE includes two main phases: (1) deploy a lightweight mirroring-based network intrusion detection agent (NICE-A) on each cloud server to capture and analyze cloud traffic. A NICE-A periodically scans the virtual system vulnerabilities within a cloud server to establish Scenario Attack Graph (SAGs), and then based on the severity of identified vulnerability towards the collaborative attack goals, NICE will decide whether or not to put a VM in network inspection state. (2) Once a VM enters inspection state, Deep Packet Inspection (DPI) is applied, and/or virtual network reconfigurations can be deployed to the inspecting VM to make the potential attack behaviors prominent.

NICE significantly advances the current network IDS/IPS solutions by employing programmable virtual networking approach that allows the system to construct a dynamic reconfigurable IDS system. By using software switching techniques [18], NICE constructs a mirroring-based traffic capturing framework to minimize the interference on users' traffic compared to traditional bump-in-the-wire (i.e., proxy-based) IDS/IPS. The programmable virtual networking architecture of NICE enables the cloud to establish inspection and quarantine modes for suspicious VMs according to their current vulnerability state in the current

SAG. Based on the collective behavior of VMs in the SAG, NICE can decide appropriate actions, for example DPI or traffic filtering, on the suspicious VMs. Using this approach, NICE does not need to block traffic flows of a suspicious VM in its early attack stage. The contributions of NICE are presented as follows:

- We devise NICE, a new multi-phase distributed network intrusion detection and prevention framework in a virtual networking environment that captures and inspects suspicious cloud traffic without interrupting users' applications and cloud services.
- NICE incorporates a software switching solution to quarantine and inspect suspicious VMs for further investigation and protection. Through programmable network approaches, NICE can improve the attack detection probability and improve the resiliency to VM exploitation attack without interrupting existing normal cloud services.
- NICE employs a novel attack graph approach for attack detection and prevention by correlating attack behavior and also suggests effective countermeasures.
- NICE optimizes the implementation on cloud servers to minimize resource consumption. Our study shows that NICE consumes less computational overhead compared to proxy-based network intrusion detection solutions.

## 3.2 Related Work

The area of detecting malicious behavior has been well explored. The work by Duan *et al.* [26] focuses on the detection of compromised machines that have been recruited to serve as spam zombies. Their approach, SPOT, is based on sequentially scanning outgoing messages while employing a statistical method Sequential Probability Ratio Test (SPRT), to quickly determine whether or not a host has been compromised. BotHunter [27] detects compromised machines based on the fact that a thorough malware infection process has a number of well defined stages that allow correlating the intrusion alarms triggered by inbound traffic with resulting outgoing communication patterns. BotSniffer [28] exploits uniform spatial-temporal behavior characteristics of compromised machines to detect zombies by grouping flows according to server connections and searching for similar behavior in the flow.

An attack graph is able to represent a series of exploits, called *atomic attacks*, that lead to an undesirable state, for example a state where an attacker has obtained administrative access to a machine. There are many automation tools to construct attack graph. O. Sheyner *et al.* [29] proposed a technique based on a modified symbolic model checking NuSMV [30] and Binary Decision Diagrams (BDDs) to construct attack graph. Their model can generate all possible attack paths, however, the scalability is a big issue for this solution. P. Ammann *et al.* [31] introduced the assumption of monotonicity, which states that the precondition of a given exploit is never invalidated by the successful application of another exploit. In other words, attackers never need to backtrack. With this assumption, they can obtain a concise, scalable graph representation for encoding attack tree. X. Ou *et al.* proposed an attack graph tool called MulVAL [32], which adopts a logic programming approach and uses Datalog language to model and analyze network system. The attack graph in the MulVAL is constructed by accumulating true facts of the monitored network system. The attack graph construction process will terminate efficiently because the number of facts is polynomial in system. In order to provide the security assessment and alert correlation features, in this paper, we modified and extended MulVAL's attack graph structure.

Intrusion Detection System (IDS) and firewall are widely used to monitor and detect suspicious events in the network. However, the false alarms and the large volume of raw alerts from IDS are two major problems for any IDS implementations. In order to identify the source or target of the intrusion in the network, especially to detect multi-step attack, the alert correction is a must-have tool. The primary goal of alert correlation is to provide system support for a global and condensed view of network attacks by analyzing raw alerts [33].

Many attack graph based alert correlation techniques have been proposed recently. L. Wang *et al.* [34] devised an in-memory structure, called *queue graph* (QG), to trace alerts matching each exploit in the attack graph. However, the implicit correlations in this design make it difficult to use the correlated alerts in the graph for analysis of similar attack scenarios. Roschke *et al.* [35] proposed a modified attack-graph-based correlation algorithm to create explicit correlations only by matching alerts to specific exploitation nodes in the attack graph with multiple mapping functions, and devised an *alert dependencies graph* (DG) to group related alerts with multiple correlation criteria. Each path in DG represents a subset of alerts that might be part of an attack scenario. However, their algorithm involved all pairs shortest path searching and sorting in DG, which consumes considerable computing power.

After knowing the possible attack scenarios, applying countermeasure is the next important task. Several solutions have been proposed to select optimal countermeasures based on the likelihood of the attack path and cost benefit analysis. A. Roy *et al.* [36] proposed an *attack countermeasure tree* (ACT) to consider attacks and countermeasures together in an attack tree structure. They devised several objective functions based on greedy and branch and bound techniques to minimize the number of countermeasure, reduce investment cost, and maximize the benefit from implementing a certain countermeasure set. In their design, each countermeasure optimization problem could be solved with and without probability assignments to the model. However, their solution focuses on a static attack scenario and predefined countermeasure for each attack. N. Poolsappasit *et al.* [37] proposed a *Bayesian attack graph* (BAG) to address dynamic security risk management problem and applied a genetic algorithm to solve countermeasure optimization problem.

Our solution utilizes a new network control approach called SDN [38], where networking functions can be programmed through software switch and OpenFlow protocol [21], plays a major role in this research. Flow-based switches, such as OVS [18] and OpenFlow

Switch (OFS) [21], support fine-grained and flow-level control for packet switching [39]. With the help of the central controller, all OpenFlow-based switches can be monitored and configured. We take advantage of flow-based switching (OVS) and network controller to apply the selected network countermeasures in our solution.

## 3.3 NICE Models

In this section, we describe how to utilize attack graphs to model security threats and vulnerabilities in a virtual networked system, and propose a VM protection model based on virtual network reconfiguration approaches to prevent VMs from being exploited.

### *3.3.1 Threat Model*

In our attack model, we assume that an attacker can be located either outside or inside of the virtual networking system. The attackers primary goal is to exploit vulnerable VMs and compromise them as zombies.

Our protection model focuses on virtual-network-based attack detection and reconfiguration solutions to improve the resiliency to zombie explorations. Our work does not involve host-based IDS and does not address how to handle encrypted traffic for attack detections. Our proposed solution can be deployed in an Infrastructure-as-a-Service (IaaS) cloud networking system, and we assume that the Cloud Service Provider (CSP) is benign. We also assume that cloud service users are free to install whatever operating systems or applications they want, even

### *3.3.2 Attack Graph Model*

An attack graph is a modeling tool to illustrate all possible multi-stage, multi-host attack paths that are crucial tco understand threats and then to decide appropriate countermeasures [40]. In an attack graph, each node represents either precondition or consequence of an exploit. The actions are not necessarily an active attack since normal protocol interactions can also be used for attacks. Attack graph is helpful in identifying potential threats, possible attacks and known vulnerabilities in a cloud system.

Since the attack graph provides details of all known vulnerabilities in the system and the connectivity information, we get a whole picture of current security situation of the system where we can predict the possible threats and attacks by correlating detected events or activities. If an event is recognized as a potential attack, we can apply specific countermeasures to mitigate its impact or take actions to prevent it from contaminating the cloud system. To represent the attack and the result of such actions, we extend the notation of MulVAL logic attack graph as presented by X. Ou *et al.* [32] and define as Scenario Attack Graph (*SAG*).

**Definition 1 (Scenario Attack Graph)** *An Scenario Attack Graph is a tuple SAG=(V, E), where*

1. $V = N_C \cup N_D \cup N_R$ *denotes a set of vertices that include three types namely conjunction node* $N_C$ *to represent exploit, disjunction node* $N_D$ *to denote result of exploit, and root node* $N_R$ *for showing initial step of an attack scenario.*

2. $E = E_{pre} \cup E_{post}$ *denotes the set of directed edges. An edge* $e \in E_{pre} \subseteq N_D \times N_C$ *represents that* $N_D$ *must be satisfied to achieve* $N_C$*. An edge* $e \in E_{post} \subseteq N_C \times N_D$ *means that the consequence shown by* $N_D$ *can be obtained if* $N_C$ *is satisfied.*

Node $v_c \in N_C$ is defined as a three tuple $(Hosts, vul, alert)$ representing a set of IP addresses, vulnerability information such as CVE [41], and alerts related to $v_c$, respectively. $N_D$ behaves like a logical OR operation and contains details of the results of actions. $N_R$ represents the root node of the scenario attack graph.

For correlating the alerts, we refer to the approach described in [35] and define a new Alert Correlation Graph (*ACG*) to map alerts in ACG to their respective nodes in SAG. To keep track of attack progress, we track the source and destination IP addresses for attack activities.

**Definition 2 (Alert Correlation Graph)** *An ACG is a three tuple $ACG = (A, E, P)$, where*

1. *A is a set of aggregated alerts. An alert $a \in A$ is a data structure $(src, dst, cls, ts)$ representing source IP address, destination IP address, type of the alert, and timestamp of the alert respectively.*

2. *Each alert a maps to a pair of vertices $(v_c, v_d)$ in SAG using $map(a)$ function, i.e., $map(a) : a \mapsto \{(v_c, v_d) | (a.src \in v_c.Hosts) \wedge (a.dst \in v_d.Hosts) \wedge (a.cls = v_c.vul)\}$.*

3. *E is a set of directed edges representing correlation between two alerts $(a, a')$ if criteria below satisfied:*

   *i.* $(a.ts < a'.ts) \wedge (a'.ts - a.ts < threshold)$

   *ii.* $\exists (v_d, v_c) \in E_{pre} : (a.dst \in v_d.Hosts \wedge a'.src \in v_c.Hosts)$

4. *P is set of paths in ACG. A path $S_i \subset P$ is a set of related alerts in chronological order.*

We assume that $A$ contains aggregated alerts rather than raw alerts. Raw alerts having same source and destination IP addresses, attack type and timestamp within a specified window are aggregated as *Meta Alerts*. Each ordered pair $(a, a')$ in *ACG* maps to two neighbor vertices in *SAG* with timestamp difference of two alerts within a predefined *threshold*. *ACG* shows dependency of alerts in chronological order and we can find related alerts in the same attack scenario by searching the alert path in *ACG*. A set $P$ is used to store all paths from root alert to the target alert in the *SAG*, and each path $S_i \subset P$ represents alerts that belong to the same attack scenario.

We explain a method for utilizing *SAG* and *ACG* together so as to predict an attacker's behavior. *Alert_Correlation* algorithm (figure 3.1) is followed for every alert detected and returns one or more paths $S_i$. For every alert $a_c$ that is received from the IDS, it is added to *ACG* if it does not exist. For this new alert $a_c$, the corresponding vertex in the *SAG* is found by using function $map(a_c)$ (line 3). For this vertex in *SAG*, alert related to its parent vertex of type $N_C$ is then correlated with the current alert $a_c$ (line 5). This creates a new set of alerts that belong to a path $S_i$ in *ACG* (line 8) or splits out a new path $S_{i+1}$ from $S_i$ with subset of $S_i$ before the alert $a$ and appends $a_c$ to $S_{i+1}$ (line 10). In the end of this algorithm, the ID of $a_c$ will be added to *alert* attribute of the vertex in *SAG*. Algorithm 1 returns a set of attack paths $S$ in *ACG*.

```
Require: alert $a_c$, $SAG$, $ACG$
 1: if ($a_c$ is a new alert) then
 2:     create node $a_c$ in $ACG$
 3:     $n_1 \leftarrow v_c \in map(a_c)$
 4:     for all $n_2 \in parent(n_1)$ do
 5:         create edge $(n_2.alert, a_c)$
 6:         for all $S_i$ containing $a$ do
 7:             if $a$ is the last element in $S_i$ then
 8:                 append $a_c$ to $S_i$
 9:             else
10:                 create path $S_{i+1} = \{subset(S_i, a), a_c\}$
11:             end if
12:         end for
13:         add $a_c$ to $n_1.alert$
14:     end for
15: end if
16: return $S$
```

Figure 3.1: NICE Algorithm 1 - Alert Correlation

### *3.3.3 VM Protection Model*

The VM protection model of NICE consists of a VM profiler, a security indexer and a state monitor. We specify security index for all the VMs in the network depending upon various factors like connectivity, the number of vulnerabilities present and their impact scores. The impact score of a vulnerability, as defined by the CVSS guide [42], helps judge the confidentiality, integrity, and availability impact of the vulnerability being exploited. Connectivity metric of a VM is decided by evaluating incoming and outgoing connections.

**Definition 3 (VM State)** *Based on the information gathered from the network controller, VM states can be defined as following:*

*1.* Stable*: there does not exist any known vulnerability on the VM.*

*2.* Vulnerable*: presence of one or more vulnerabilities on a VM, which remains unexploited.*

*3.* Exploited*: at least one vulnerability has been exploited and the VM is compromised.*

*4.* Zombie*: VM is under control of attacker.*

## 3.4 NICE System Design

In this section, we present the system design overview of NICE and briefly describe the functionality of its components.

### *3.4.1 System Design Overview*

The proposed NICE framework is illustrated in figure 3.2. It shows the NICE framework within one cloud server cluster. Major components in this framework are distributed and light-weighted NICE-A on each physical cloud server, a network controller, a VM profiling server, and an attack analyzer. The latter three components are located in a centralized control center connected to software switches on each cloud server (i.e., virtual switches built on one or multiple Linux software bridges). NICE-A is a software agent implemented in each cloud server connected to the control center through a dedicated and

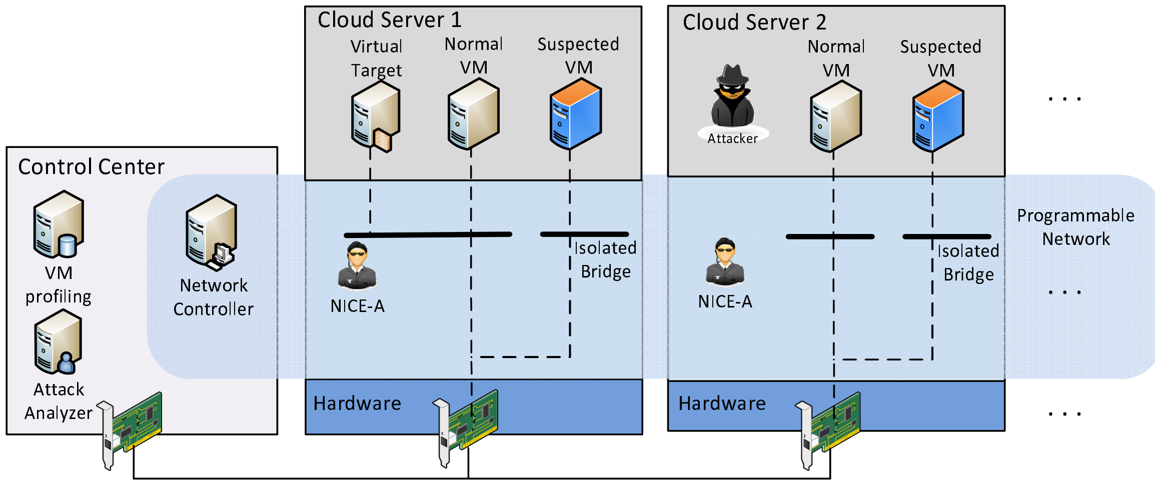


Figure 3.2: NICE Architecture Within One Cloud Server Cluster.

isolated secure channel, which is separated from the normal data packets using OpenFlow tunneling or VLAN approaches. The network controller is responsible for deploying attack countermeasures based on decisions made by the attack analyzer.

In the following description, our terminologies are based on the XEN virtualization technology. NICE-A is a network intrusion detection engine that can be installed in either Dom0 or DomU of a XEN cloud server to capture and filter malicious traffic. Intrusion detection alerts are sent to control center when suspicious or anomalous traffic is detected. After receiving an alert, attack analyzer evaluates the severity of the alert based on the attack graph, decides what countermeasure strategies to take, and then initiates it through the network controller. An attack graph is established according to the vulnerability information derived from both offline and realtime vulnerability scans. Offline scanning can be done by running penetration tests and online realtime vulnerability scanning can be triggered by the network controller (e.g., when new ports are opened and identified by OpenFlow switches) or when new alerts are generated by the NICE-A. Once new vulnerabilities are discovered or countermeasures are deployed, the attack graph will be reconstructed. Countermeasures are initiated by the attack analyzer based on the evaluation results from the cost-benefit analysis of the effectiveness of countermeasures. Then, the network controller initiates countermeasure actions by reconfiguring virtual or physical OpenFlow switches.

### *3.4.2 System Components*

In this section, we explain each component of NICE.

#### NICE-A

The NICE-A is a Network-based Intrusion Detection System (NIDS) agent installed in either Dom0 or DomU in each cloud server. It scans the traffic going through Linux bridges that control all the traffic among VMs and in/out from the physical cloud servers. In our experiment, Snort is used to implement NICE -A in Dom0. It will sniff a mirroring port on each virtual bridge in the Open vSwitch. Each bridge forms an isolated subnet in the virtual network and connects to all related VMs. The traffic generated from the VMs on the mirrored software bridge will be mirrored to a specific port on a specific bridge using SPAN, RSPAN, or ERSPAN methods. The NICE-A sniffing rules have been custom defined to suite our needs. Dom0 in the Xen environment is a privilege domain, that includes a virtual switch for traffic switching among VMs and network drivers for physical network interface of the cloud server. It's more efficient to scan the traffic in Dom0 since all traffic in the cloud server needs go through it, however our design is independent to the installed VM. In the performance evaluation section, we will demonstrate the trade-offs of installing NICE-A in Dom0 and DomU.

We must note that the alert detection quality of NICE-A depends on the implementation of NICE-A which uses Snort. We do not focus on the detection accuracy of Snort in this article. Thus, the individual alert detection's false alarm rate does not change. However, the false alarm rate could be reduced through our architecture design. We will discuss more about this issue in the later section.

### VM Profiling

Virtual machines in the cloud can be profiled to get precise information about their state, services running, open ports, etc. One major factor that counts towards a VM profile is its connectivity with other VMs. Any VM that is connected to more number of machines is more crucial than the one connected to fewer VMs because the effect of compromise of a highly connected VM can cause more damage. Also required is the knowledge of services running on a VM so as to verify the authenticity of alerts pertaining to that VM. An attacker can use port-scanning program to perform an intense examination of the network to look for open ports on any VM. So information about any open ports on a VM and the history of opened ports plays a significant role in determining how vulnerable the VM is. All these factors combined will form the VM profile.

VM profiles are maintained in a database and contain comprehensive information about vulnerabilities, alert and traffic. The data comes from:

- Attack graph generator: while generating the attack graph, every detected vulnerability is added to its corresponding VM entry in the database.
- NICE-A: the alert involving the VM will be recorded in the VM profile database.
- Network controller: the traffic patterns involving the VM are based on 5 tuples (source MAC address, destination MAC address, source IP address, destination IP address, protocol). We can have traffic pattern where packets emanate from a single IP and are delivered to multiple destination IP addresses, and vice-versa.

### Attack Analyzer

The major functions of NICE system are performed by attack analyzer, which includes procedures such as attack graph construction and update, alert correlation and countermeasure selection.

The process of constructing and utilizing the Scenario Attack Graph (*SAG*) consists of three phases: information gathering, attack graph construction, and potential exploit path analysis. With this information, attack paths can be modeled using SAG. Each node in the attack graph represents an exploit by the attacker. Each path from an initial node to a goal node represents a successful attack.

In summary, NICE attack graph is constructed based on the following information:

- *Cloud system information* is collected from the node controller (i.e., Dom0 in XenServer). The information includes the number of VMs in the cloud server, running services on each VM, and VM's Virtual Interface (VIF) information.

- *Virtual network topology and configuration information* is collected from the network controller, which includes virtual network topology, host connectivity, VM connectivity, every VM's IP address, MAC address, port information, and traffic flow information.

- *Vulnerability information* is generated by both on-demand vulnerability scanning (i.e., initiated by the network controller and NICE-A) and regular penetration testing using the well-known vulnerability databases, such as Open Source Vulnerability Database (OSVDB)[43], Common Vulnerabilities and Exposures List (CVE)[41], and NIST National Vulnerability Database (NVD) [44].

The Attack Analyzer also handles alert correlation and analysis operations. This component has two major functions: (1) constructs Alert Correlation Graph (*ACG*), (2) provides threat information and appropriate countermeasures to network controller for virtual network reconfiguration.

Figure 3.3 shows the workflow in the attack analyzer component. After receiving an alert from NICE-A, alert analyzer matches the alert in the *ACG*. If the alert already exists

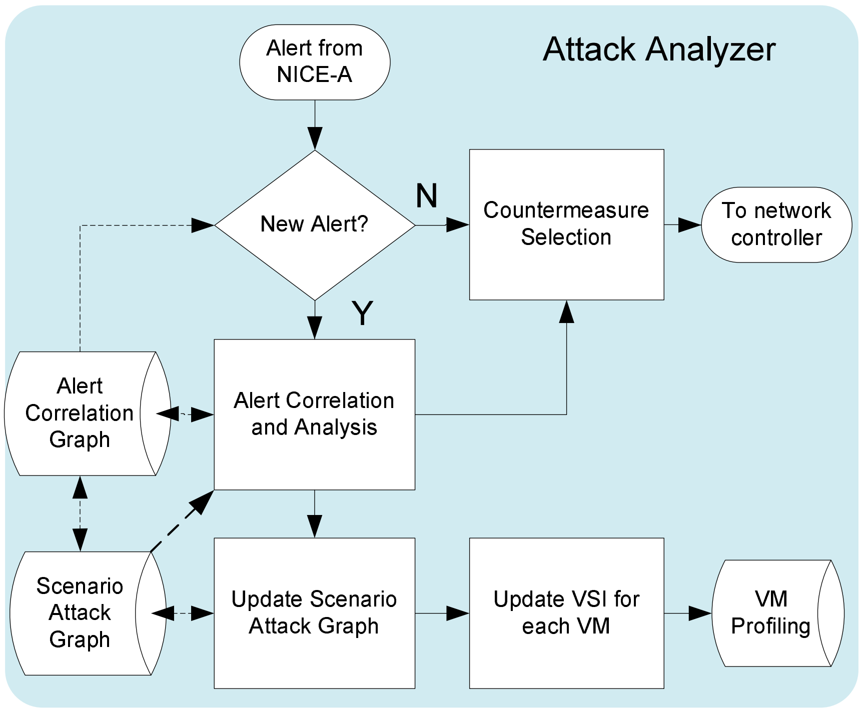


Figure 3.3: Workflow of Attack Analyzer.

in the graph and it is a known attack (i.e., matching the attack signature), the attack analyzer performs countermeasure selection procedure according to Algorithm 2 (figure 3.4), and then notifies network controller immediately to deploy countermeasure or mitigation actions. If the alert is new, attack analyzer will perform alert correlation and analysis according to Algorithm 1 (figure 3.1), and updates *ACG* and *SAG*. This algorithm correlates each new alert to a matching alert correlation set (i.e., in the same attack scenario). A selected countermeasure is applied by the network controller based on the severity of evaluation results. If the alert is a new vulnerability and is not present in the NICE attack graph, the attack analyzer adds it to attack graph and then reconstructs it.

**Network Controller**

The network controller is a key component to support the programmable networking capability to realize the virtual network reconfiguration feature based on OpenFlow protocol [39]. In NICE, within each cloud server there is a software switch, for example, Open vSwitch (OVS) [18], which is used as the edge switch for VMs to handle traffic in & out from VMs. The communication between cloud servers (i.e., physical servers) is handled by physical OpenFlow-capable Switch (OFS). In NICE, we integrated the control functions for both OVS and OFS into the network controller that allows the cloud system to set security/filtering rules in an integrated and comprehensive manner.

The network controller is responsible for collecting network information of current OpenFlow network and provides input to the attack analyzer to construct attack graphs. Through the cloud internal discovery modules that use DNS, DHCP, LLDP and flow-initiations [45], network controller is able to discover the network connectivity information from OVS and OFS. This information includes current data paths on each switch and detailed flow information associated with these paths, such as TCP/IP and MAC header. The network flow and topology change information will be automatically sent to the controller and then delivered to attack analyzer to reconstruct attack graphs.

Another important function of the network controller is to assist the attack analyzer module. According to the OpenFlow protocol [39], when the controller receives the first packet of a flow, it holds the packet and checks the flow table for complying traffic policies. In NICE, the network control also consults with the attack analyzer for the flow access control by setting up the filtering rules on the corresponding OVS and OFS. Once a traffic flow is admitted, the following packets of the flow are not handled by the network controller, but monitored by the NICE-A.

Network controller is also responsible for applying the countermeasure from attack analyzer. Based on *VM Security Index* and severity of an alert, countermeasures are selected by NICE and executed by the network controller. If a severe alert is triggered and identifies some known attacks, or a VM is detected as a zombie, the network controller will block the VM immediately. An alert with medium threat level is triggered by a suspicious compromised VM. Countermeasure in such case is to put the suspicious VM with exploited state into quarantine mode and redirect all its flows to NICE-A Deep Packet Inspection (DPI) mode. An alert with a minor threat level can be generated due to the presence of a vulnerable VM. For this case, in order to intercept the VM's normal traffic, suspicious traffic to/from the VM will be put into inspection mode, in which actions such as restricting its flow bandwidth and changing network configurations will be taken to force the attack exploration behavior to stand out.

## 3.5 NICE Security Measurement, Attack Mitigation and Countermeasures

In this section, we present the methods for selecting the countermeasures for a given attack scenario. When vulnerabilities are discovered or some VMs are identified as suspicious, several countermeasures can be taken to restrict attackers' capabilities and it's important to differentiate between compromised and suspicious VMs. The countermeasure serves the purpose of 1) protecting the target VMs from being compromised; and 2) making attack behavior stand prominent so that the attackers' actions can be identified.

### *3.5.1 Security Measurement Metrics*

The issue of security metrics has attracted much attention and there has been significant effort in the development of quantitative security metrics in recent years. Among different approaches, using attack graph as the security metric model for the evaluation of security risks [46] is a good choice. In order to assess the network security risk condition for the current network configuration, security metrics are needed in the attack graph to measure risk likelihood. After an attack graph is constructed, vulnerability information is included in the graph. For the initial node or external node (i.e., the root of the graph, $N_R \subseteq N_D$), the *priori probability* is assigned on the likelihood of a threat source becoming active and the difficulty of the vulnerability to be exploited. We use $G_V$ to denote the priori risk probability for the root node of the graph and usually the value of $G_V$ is assigned to a high probability, e.g., from 0.7 to 1.

For the internal exploitation node, each attack-step node $e \in N_C$ will have a *probability of vulnerability exploitation* denoted as $G_M[e]$. $G_M[e]$ is assigned according to the Base Score (*BS*) from CVSS (Common Vulnerability Scoring System). The base score as shown in (3.1) [42], is calculated by the impact and exploitability factor of the vulnerability. Base score can be directly obtained from National Vulnerability Database [44] by searching for the vulnerability CVE id.

$$BS = (0.6 \times IV + 0.4 \times E - 1.5) \times f(IV), \tag{3.1}$$

where,

$$IV = 10.41 \times (1 - (1 - C) \times (1 - I) \times (1 - A)),$$

$$E = 20 \times AC \times AU \times AV,$$

and

$$f(IV) = \begin{cases} 0 & \text{if } IV = 0, \\ 1.176 & \text{otherwise.} \end{cases}$$

The impact value ($IV$) is computed from three basic parameters of security namely confidentiality ($C$), integrity ($I$), and availability ($A$). The exploitability ($E$) score consists of access vector ($AV$), access complexity ($AC$), and authentication instances ($AU$). The value of *BS* ranges from 0 to 10. In our attack graph, we assign each internal node with its *BS* value divided by 10, as shown in (3.2).

$$G_M[e] = Pr(e = T) = BS(e)/10, \forall e \in N_C. \tag{3.2}$$

In the attack graph, the relations between exploits can be disjunctive or conjunctive according to how they are related through their dependency conditions [47]. Such relationships can be represented as *conditional probability*, where the risk probability of current node is determined by the relationship with its predecessors and their risk probabilities. We propose the following probability derivation relations:

- for any attack-step node $n \in N_C$ with immediate predecessors set $W = parent(n)$,

$$Pr(n|W) = G_M[n] \times \prod_{s \in W} Pr(s|W); \tag{3.3}$$

- for any privilege node $n \in N_D$ with immediate predecessors set $W = parent(n)$, and then

$$Pr(n|W) = 1 - \prod_{s \in W} (1 - Pr(s|W)). \tag{3.4}$$

Once conditional probabilities have been assigned to all internal nodes in SAG, we can merge risk values from all predecessors to obtain the *cumulative risk probability* or *absolute risk probability* for each node according to (3.5) and (3.6). Based on derived conditional probability assignments on each node, we can then derive an effective security hardening plan or a mitigation strategy:

- for any attack-step node $n \in N_C$ with immediate predecessor set $W = parent(n)$,

$$Pr(n) = Pr(n|W) \times \prod_{s \in W} Pr(s); \tag{3.5}$$

- for any privilege node $n \in N_D$ with immediate predecessor set $W = parent(n)$,

$$Pr(n) = 1 - \prod_{s \in W} (1 - Pr(s)). \tag{3.6}$$

### *3.5.2 Mitigation Strategies*

Based on the security metrics defined in the previous subsection, NICE is able to construct the mitigation strategies in response to detected alerts. First, we define the term *countermeasure pool* as follows:

**Definition 4 (Countermeasure Pool)** *A countermeasure pool $CM = \{cm_1, cm_2, \ldots, cm_n\}$ is a set of countermeasures. Each $cm \in CM$ is a tuple cm = (*cost*,* intrusiveness*,* condition*,* effectiveness*), where*

1. cost *is the unit that describes the expenses required to apply the countermeasure in terms of resources and operational complexity, and it is defined in a range from 1 to 5, and higher metric means higher cost;*

2. intrusiveness *is the negative effect that a countermeasure brings to the Service Level Agreement (SLA) and its value ranges from the least intrusive (1) to the most intrusive (5), and the value of intrusiveness is 0 if the countermeasure has no impacts on the SLA;*

3. condition *is the requirement for the corresponding countermeasure;*

4. effectiveness *is the percentage of probability changes of the node, for which this countermeasure is applied.*

In general, there are many countermeasures that can be applied to the cloud virtual networking system depending on available countermeasure techniques that can be applied. Without losing the generality, several common virtual-networking-based countermeasures are listed in table 3.1. The optimal countermeasure selection is a multi-objective optimization problem, to calculate *MIN*(impact, cost) and *MAX*(benefit).

Table 3.1: Possible Countermeasure Types

| No. | Countermeasure | Intrusiveness | Cost |
|---|---|---|---|
| 1 | Traffic redirection | 3 | 3 |
| 2 | Traffic isolation | 4 | 2 |
| 3 | Deep Packet Inspection | 3 | 3 |
| 4 | Creating filtering rules | 1 | 2 |
| 5 | MAC address change | 2 | 1 |
| 6 | IP address change | 2 | 1 |
| 7 | Block port | 4 | 1 |
| 8 | Software patch | 5 | 4 |
| 9 | Quarantine | 5 | 2 |
| 10 | Network reconfiguration | 0 | 5 |
| 11 | Network topology change | 0 | 5 |

In NICE, the network reconfiguration strategies mainly involve two levels of action: layer-2 and layer-3. At layer-2, virtual bridges (including tunnels that can be established between two bridges) and VLANs are main component in cloud's virtual networking system to connect two VMs directly. A virtual bridge is an entity that attaches Virtual Interfaces (VIFs). Virtual machines on different bridges are isolated at layer 2. VIFs on the same virtual bridge but with different VLAN tags cannot communicate to each other directly. Based on this layer-2 isolation, NICE can deploy layer-2 network reconfiguration to isolate suspicious VMs. For example, vulnerabilities due to Arpspoofing [48] attacks are not possible when the suspicious VM is isolated to a different bridge. As a result, this countermeasure disconnects an attack path in the attack graph causing the attacker to explore an alternate attack path. Layer-3 reconfiguration is another way to disconnect an attack path. Through the network controller, the flow table on each OVS or OFS can be modified to change the network topology.

We must note that using the virtual network reconfiguration approach at lower layer has the advantage in that upper layer applications will experience minimal impact. Especially, this approach is only possible when using software-switching approach to automate the reconfiguration in a highly dynamic networking environment. Countermeasures such as traffic isolation can be implemented by utilizing the traffic engineering capabilities of OVS and OFS to restrict the capacity and reconfigure the virtual network for a suspicious flow. When a suspicious activity such as network and port scanning is detected in the cloud system, it is important to determine whether the detected activity is indeed malicious or not. For example, attackers can purposely hide their scanning behavior to prevent the NIDS from identifying their actions. In such situation, changing the network configuration will force the attacker to perform more explorations, and in turn will make their attacking behavior stand out.

### *3.5.3 Countermeasure Selection*

Algorithm 2 (figure 3.4) presents how to select the optimal countermeasure for a given attack scenario. Input to the algorithm is an $alert$, attack graph $G$, and a pool of countermeasures $CM$. The algorithm starts by selecting the node $v_{Alert}$ that corresponds to the alert generated by a NICE-A. Before selecting the countermeasure, we count the distance of $v_{Alert}$ to the $target\_node$. If the distance is greater than a threshold value, we do not perform countermeasure selection but update the ACG to keep track of alerts in the system (line 3). For the source node $v_{Alert}$, all the reachable nodes (including the source node) are collected into a set $T$ (line 6). Because the alert is generated only after the attacker has performed the action, we set the probability of $v_{Alert}$ to 1 and calculate the new probabilities for all of its child (downstream) nodes in the set $T$ (line 7 & 8). Now for all $t \in T$ the applicable countermeasures in $CM$ are selected and new probabilities are calculated according to the effectiveness of the selected countermeasures (line 13 & 14). The change in probability of $target\_node$ gives the $benefit$ for the applied countermeasure using (3.7). In the next double *for-loop*, we compute the *Return of Investment* (ROI) for each $benefit$ of the applied countermeasure based on (5.5). The countermeasure which when applied on a node gives the least value of ROI, is regarded as the optimal countermeasure. Finally, SAG and ACG are also updated before terminating the algorithm. The complexity of Algorithm 2 is $\mathscr{O}(|V| \times |CM|)$ where $|V|$ is the number of vulnerabilities and $|CM|$ represents the number of countermeasures.

```
Require: $Alert, G(E,V), CM$
1: Let $v_{Alert}$ = Source node of the $Alert$
2: if Distance_to_Target($v_{Alert}$) $> threshold$ then
3:     Update_ACG
4:     return
5: end if
6: Let $T = Descendant(v_{Alert}) \cup v_{Alert}$
7: Set $Pr(v_{Alert}) = 1$
8: Calculate_Risk_Prob($T$)
9: Let $benefit[|T|, |CM|] = \emptyset$
10: for each $t \in T$ do
11:     for each $cm \in CM$ do
12:         if $cm.condition(t)$ then
13:             $Pr(t) = Pr(t) * (1 - cm.effectiveness)$
14:             Calculate_Risk_Prob($Descendant(t)$)
15:                 $$benefit[t, cm] = \Delta Pr(target\_node). \quad (3.7)$$
16:         end if
17:     end for
18: end for
19: Let $ROI[|T|, |CM|] = \emptyset$
20: for each $t \in T$ do
21:     for each $cm \in CM$ do
22:             $$ROI[t, cm] = \frac{benefit[t, cm]}{cost.cm + intrusiveness.cm}. \quad (3.8)$$
23:     end for
24: end for
25: Update_SAG and Update_ACG
26: return Select_Optimal_CM($ROI$)
```

Figure 3.4: NICE Algorithm 2 - Countermeasure Selection

## 3.6 Performance Evaluation

In this section, we evaluate the security performance, the system computing, and the network reconfiguration overhead due to the introduced security mechanism.

### *3.6.1 Security Performance Analysis*

To demonstrate the security performance of NICE, we created a virtual network testing environment consisting of all the presented components of NICE.

#### Environment and Configuration

To evaluate the security performance, a demonstrative virtual cloud system consisting of public (public virtual servers) and private (VMs) virtual domains is established as shown in Figure 3.5. Cloud Servers 1 and 2 are connected to Internet through the external firewall. In the Demilitarized Zone (DMZ) on Server 1, there is one Mail server, one DNS server and one Web server. Public network on Server 2 houses SQL server and NAT Gateway Server. Remote access to VMs in the private network is controlled through SSHD (i.e., SSH Daemon) from the NAT Gateway Server. Table 3.2 shows the vulnerabilities present in this network and table 3.3 shows the corresponding network connectivity that can be explored based on the identified vulnerabilities.

Table 3.2: Vulnerabilities in the Virtual Networked System.

| Host | Vulnerability | Node | CVE | Base Score |
|---|---|---|---|---|
| VM group | LICQ buffer overflow | 10 | CVE 2001-0439 | 0.75 |
| | MS Video ActiveX Stack buffer overflow | 5 | CVE 2008-0015 | 0.93 |
| | GNU C Library loader flaw | 22 | CVE-2010-3847 | 0.69 |
| Admin Server | MS SMV service Stack buffer overflow | 2 | CVE 2008-4050 | 0.93 |
| Gateway server | OpenSSL uses predictable random variable | 15 | CVE 2008-0166 | 0.78 |
| | Heap corruption in OpenSSH | 4 | CVE 2003-0693 | 1 |
| | Improper cookies handler in OpenSSH | 9 | CVE 2007-4752 | 0.75 |
| Mail server | Remote code execution in SMTP | 21 | CVE 2004-0840 | 1 |
| | Squid port scan | 19 | CVE 2001-1030 | 0.75 |
| Web server | WebDAV vulnerability in IIS | 13 | CVE 2009-1535 | 0.76 |

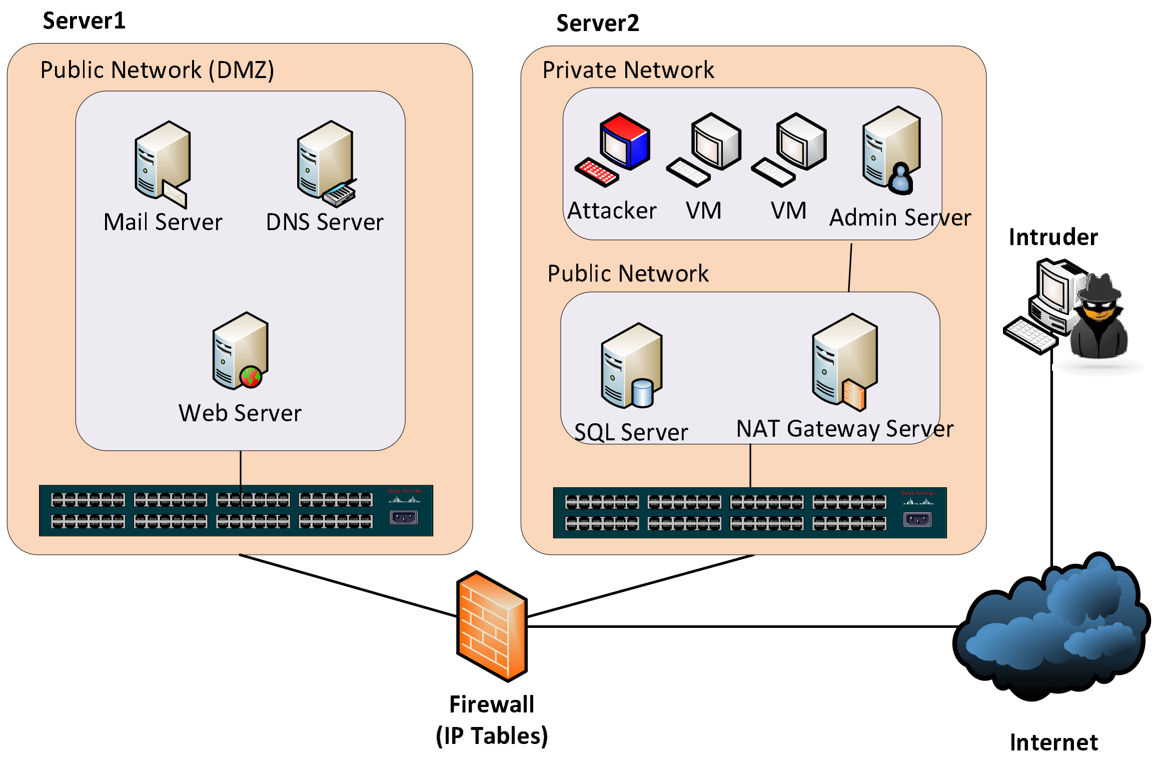
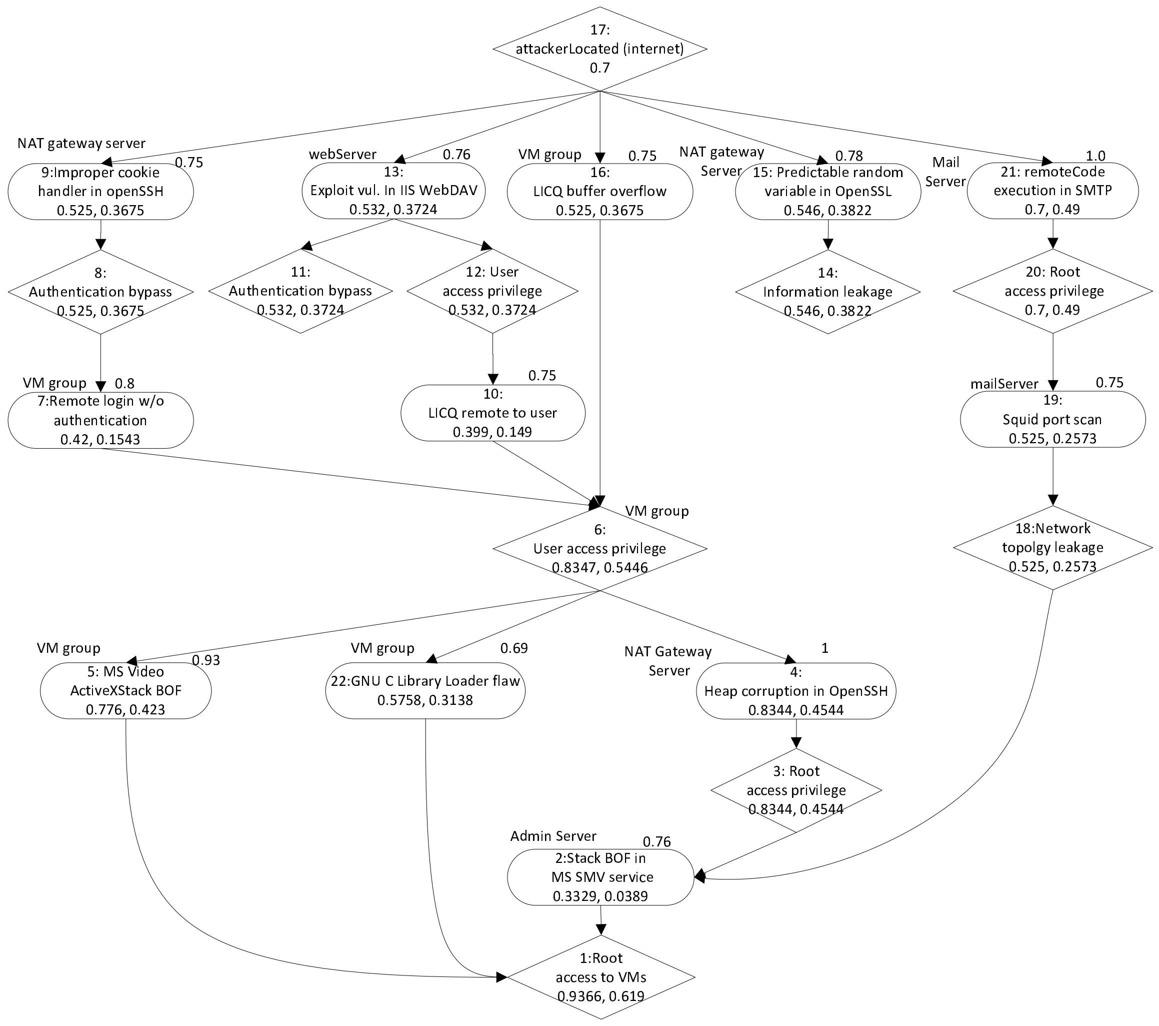


Figure 3.5: Virtual Network Topology for Security Evaluation.

### Attack Graph and Alert Correlation

The attack graph can be generated by utilizing network topology and the vulnerability information, and it is shown in Figure 3.6. As the attack progresses, the system generates various alerts that can be related to the nodes in the attack graph.

Creating an attack graph requires knowledge of network connectivity, running services and their vulnerability information. This information is provided to the attack graph generator as the input. Whenever a new vulnerability is discovered or there are changes in the network connectivity and services running through them, the updated information is provided to attack graph generator and old attack graph is updated to a new one. *SAG* provides information about the possible paths that an attacker can follow. *ACG* serves the purpose of confirming attackers' behavior, and helps in determining false positive and false negative. *ACG* can also be helpful in predicting attackers' next steps.



17: attackerLocated (internet) 0.7
NAT gateway server
0.75
9:Improper cookie handler in openSSH 0.525, 0.3675
webServer
0.76
13: Exploit vul. In IIS WebDAV 0.532, 0.3724
VM group
0.75
16: LICQ buffer overflow 0.525, 0.3675
NAT gateway Server
0.78
15: Predictable random variable in OpenSSL 0.546, 0.3822
Mail Server
1.0
21: remoteCode execution in SMTP 0.7, 0.49
8: Authentication bypass 0.525, 0.3675
11: Authentication bypass 0.532, 0.3724
12: User access privilege 0.532, 0.3724
14: Information leakage 0.546, 0.3822
20: Root access privilege 0.7, 0.49
VM group
0.8
7:Remote login w/o authentication 0.42, 0.1543
0.75
10: LICQ remote to user 0.399, 0.149
mailServer
0.75
19: Squid port scan 0.525, 0.2573
VM group
6: User access privilege 0.8347, 0.5446
18:Network topolgy leakage 0.525, 0.2573
VM group
0.93
5: MS Video ActiveXStack BOF 0.776, 0.423
VM group
0.69
22:GNU C Library Loader flaw 0.5758, 0.3138
NAT Gateway Server
1
4: Heap corruption in OpenSSH 0.8344, 0.4544
3: Root access privilege 0.8344, 0.4544
Admin Server
0.76
2:Stack BOF in MS SMV service 0.3329, 0.0389
1:Root access to VMs 0.9366, 0.619

Figure 3.6: Attack Graph for the Test Network.

### Countermeasure Selection

To illustrate how NICE works, let us consider for example, an alert is generated for node 16 ($v_{Alert} = 16$) when the system detects LICQ Buffer overflow. After the alert is generated, the cumulative probability of node 16 becomes 1 because that attacker has already compromised that node. This triggers a change in cumulative probabilities of child nodes of node 16. Now the next step is to select the countermeasures from the pool of countermeasures *CM*. If the countermeasure *CM4: create filtering rules* is applied to node 5 and we assume

Table 3.3: Virtual Network Connectivity.

| From | To | Protocol |
|---|---|---|
| Internet | NAT Gateway server | SSHD |
|  | Mail server | IMAP, SMTP |
|  | Web server | HTTP |
| Web server | SQL server | SQL |
| NAT Gateway server | VM group | Basic network protocols |
|  | Admin server | Basic network protocols |
| VM Group | NAT Gateway server | Basin network protocols |
|  | Mail server | IMAP, SMTP |
|  | SQL server | SQL |
|  | Web server | HTTP |
|  | DNS server | DNS |

that this countermeasure has effectiveness of 85%, the probability of node 5 will change to 0.1164, which causes change in probability values of all child nodes of node 5 thereby accumulating to a decrease of 28.5% for the target node 1. Following the same approach for all possible countermeasures that can be applied, the percentage change in the cumulative probability of node 1, i.e., *benefit* computed using (3.7) are shown in Figure 3.7.

Apart from calculating the *benefit* measurements, we also present the evaluation based on *Return of Investment (ROI)* using (5.5) and represent a comprehensive evaluation considering *benefit*, *cost* and *intrusiveness* of countermeasure. Figure 3.8 shows the *ROI* evaluations for presented countermeasures. Results show that countermeasures CM2 and CM8 on node 5 have the maximum *benefit* evaluation, however their cost and intrusiveness scores indicate that they might not be good candidates for the optimal countermeasure and *ROI* evaluation results confirm this. The *ROI* evaluations demonstrate that CM4 on node 5 is the optimal solution.

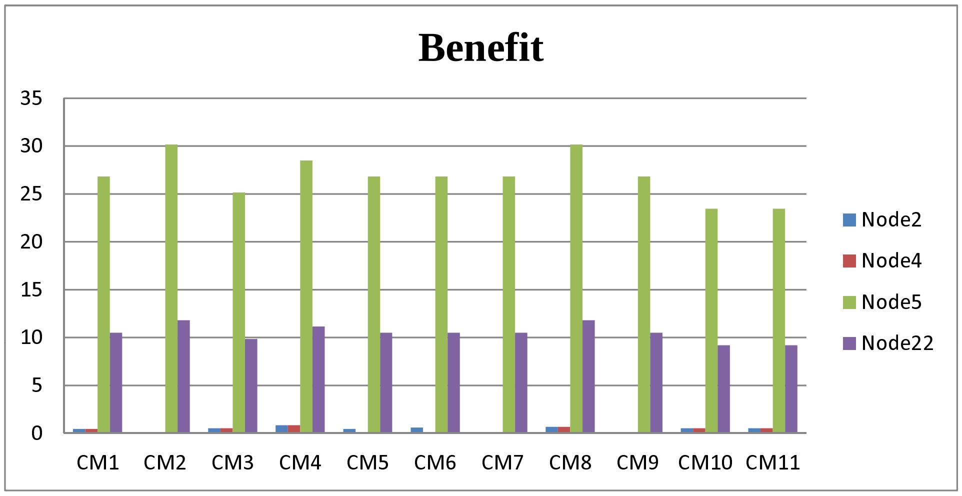


Figure 3.7: Benefit Evaluation Chart.

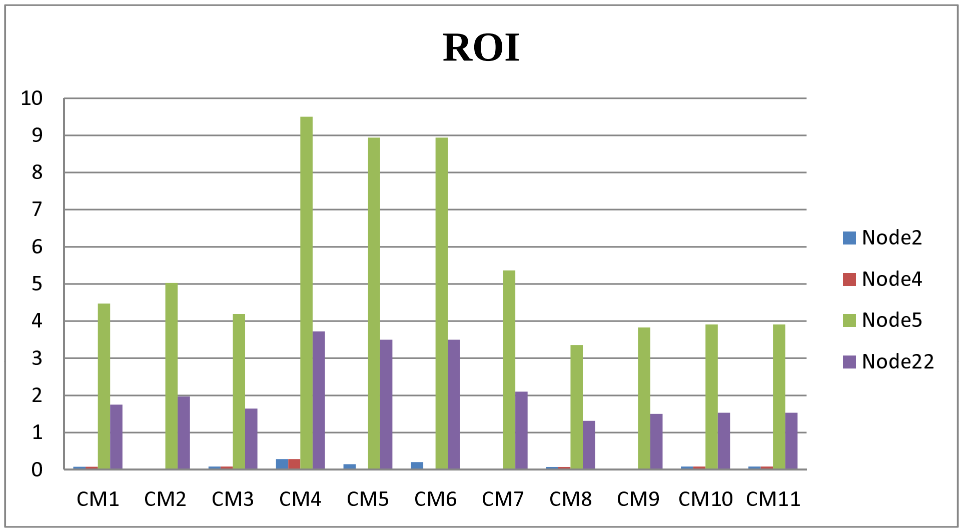


Figure 3.8: ROI Evaluation Chart.

**Experiment in Private Cloud Environment**

Previously we presented an example where the attackers target is VM in the private network. For performance analysis and capacity test, we extended the configuration in figure 3.3 to create another test environment which includes 14 VMs across 3 cloud servers and configured each VM as a target node to create a dedicated SAG for each VM. These

VMs consist of Windows (W) and Linux (L) machines in the private network 172.16.11.0/24, and contains more number of vulnerabilities related to their OSes and applications. We created penetration testing scripts with Metasploit framework [49] and Armitage [50] as attackers in our test environment. These scripts emulate attackers from different places in the internal and external sources, and launch diversity of attacks based on the vulnerabilities in each VM.

In order to evaluate security level of a VM, we define a VM Security Index (VSI) to represent the security level of each VM in the current virtual network environment. This VSI refers to the VEA-bility metric [51] and utilizes two parameters that include Vulnerability and Exploitability as security metrics for a VM. The VSI value ranges from 0 to 10, where lower value means better security. We now define VSI:

**Definition 5 (VM Security Index)** *VSI for a virtual machine k is defined as* $VSI_k = (V_k + E_k)/2$*, where*

1. $V_k$ *is vulnerability score for VM k. The score is the exponential average of base score from each vulnerability in the VM or a maximum 10, i.e.,* $V_k = \min\{10, \ln \sum e^{BaseScore(v)}\}$.
2. $E_k$ *is exploitability score for VM k. It is the exponential average of exploitability score for all vulnerabilities or a maximum 10 multiplied by the ratio of network services on the VM, i.e.,* $E_k = (\min\{10, \ln \sum e^{ExploitabilityScore(v)}\}) \times (\frac{S_k}{NS_k})$. $S_k$ *represents the number of services provided by VM k.* $NS_k$ *represents the number of network services the VM k can connect to.*

Basically, vulnerability score considers the base scores of all the vulnerabilities on a VM. The base score depicts how easy it is for an attacker to exploit the vulnerability and how much damage it may incur. The exponential addition of base scores allows the vulnerability score to incline towards higher base score values and increases in logarithm-scale based on the number of vulnerabilities. The exploitability score on the other hand shows the accessibility of a target VM, and depends on the ratio of the number of services to the number of network services as defined in [51]. Higher exploitability score means that there are many possible paths for that attacker to reach the target.

VSI can be used to measure the security level of each VM in the virtual network in the cloud system. It can be also used as an indicator to demonstrate the security status of a VM, i.e., a VM with higher value of VSI means is easier to be attacked. In order to prevent attackers from exploiting other vulnerable VMs, the VMs with higher VSI values need to be monitored closely by the system (e.g., using DPI) and mitigation strategies may be needed to reduce the VSI value when necessary.

Figure 3.9 shows the plotting of *VSI* for these virtual machines before countermeasure selection and application. Figure 3.10 compares *VSI* values before and after applying the countermeasure CM4, i.e., creating filtering rules. It shows the percentage change in *VSI* after applying countermeasure on all of the VMs. Applying CM4 avoids vulnerabilities and causes *VSI* to drop without blocking normal services and ports.

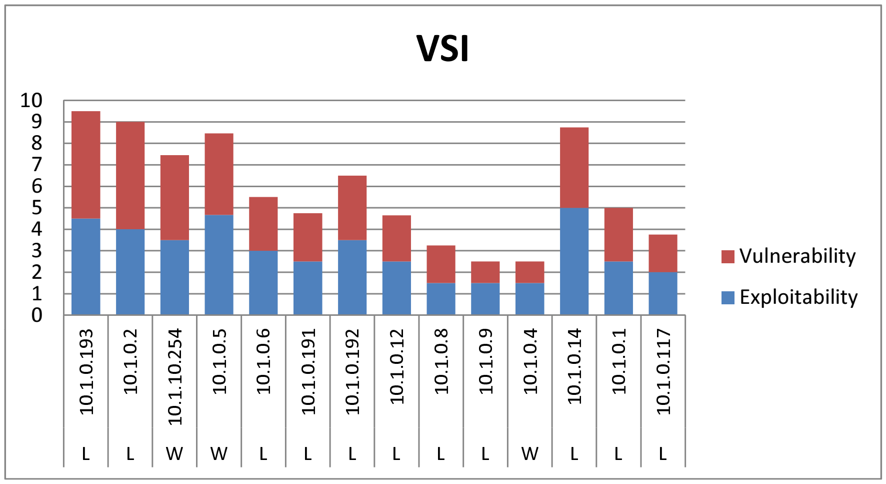


Figure 3.9: VM Security Index.

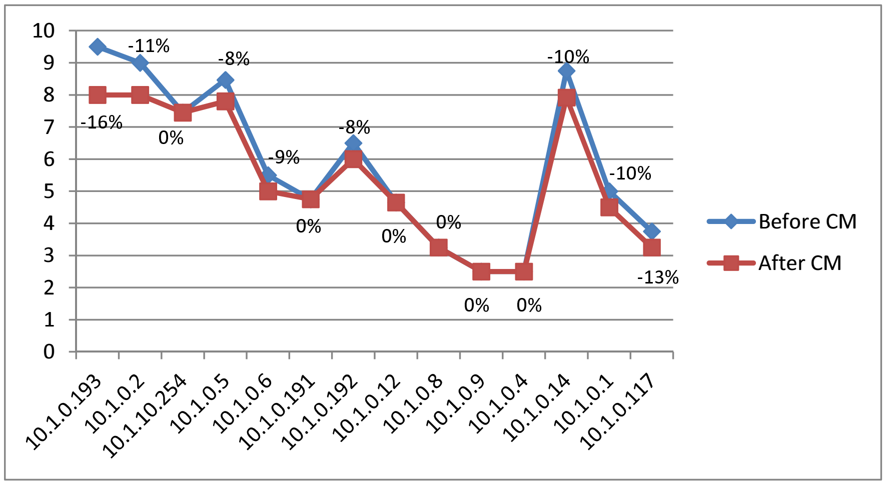


Figure 3.10: Change in VM Security Index.

**False Alarms**

A cloud system with hundreds of nodes will have huge amount of alerts raised by Snort. Not all of these alerts can be relied upon, and an effective mechanism is needed to verify if such alerts need to be addressed. Since Snort can be programmed to generate alerts with CVE id, one approach that our work provides is to match if the alert is actually related to some vulnerability being exploited. If so, the existence of that vulnerability in SAG means that the alert is more likely to be a real attack. Thus, the false positive rate will be the joint probability of the correlated alerts, which will not increase the false positive rate compared to each individual false positive rate.

Moreover, we cannot keep aside the case of zero-day attack where the vulnerability is discovered by the attacker but is not detected by vulnerability scanner. In such case, the alert being real will be regarded as false, given that there does not exist corresponding node in SAG. Thus, current research does not address how to reduce the false negative rate. It is important to note that vulnerability scanner should be able to detect most recent vulnerabilities and sync with the latest vulnerability database to reduce the chance of Zero-day attacks.

### *3.6.2 NICE System Performance*

We evaluate system performance to provide guidance on how much traffic NICE can handle for one cloud server and use the evaluation metric to scale up to a large cloud system. In a real cloud system, traffic planning is needed to run NICE, which is beyond the scope of this paper. Due to the space limitation, we will investigate the research involving multiple cloud clusters in the future.

To demonstrate the feasibility of our solution, comparative studies were conducted on several virtualization approaches. We evaluated NICE based on Dom0 and DomU implementations with mirroring-based and proxy-based attack detection agents (i.e., NICE-A). In mirror-based IDS scenario, we established two virtual networks in each cloud server: normal network and monitoring network. NICE-A is connected to the monitoring network. Traffic on the normal network is mirrored to the monitoring network using Switched Port Analyzer (SPAN) approach. In the proxy-based IDS solution, NICE-A interfaces two VMs and the traffic goes through NICE-A. Additionally, we have deployed the NICE-A in Dom0 and it removes the traffic duplication function in mirroring and proxy-based solutions.

NICE-A running in Dom0 is more efficient since it can sniff the traffic directly on the virtual bridge. However, in DomU, the traffic need to be duplicated on the VM's virtual interface (vif), causing overhead. When the IDS is running in Intrusion Prevention System (IPS) mode, it needs to intercept all the traffic and perform packet checking, which consumes more system resources as compared to IDS mode. To demonstrate performance evaluations we used four metrics namely CPU utilization, network capacity, agent processing capacity, and communication delay. We performed the evaluation on cloud servers with Intel quad-core Xeon 2.4Ghz CPU and 32G memory.

We used packet generator to mimic real traffic in the Cloud system. As shown in Figure 3.11, the traffic load, in form of packet sending speed, increases from 1 to 3000 packets per second. The performance at Dom0 consumes less CPU and the IPS mode consumes the maximum CPU resources. It can be observed that when the packet rate reaches to 3000 packets per second; the CPU utilization of IPS at DomU reaches its limitation, while the IDS mode at DomU only occupies about 68%.

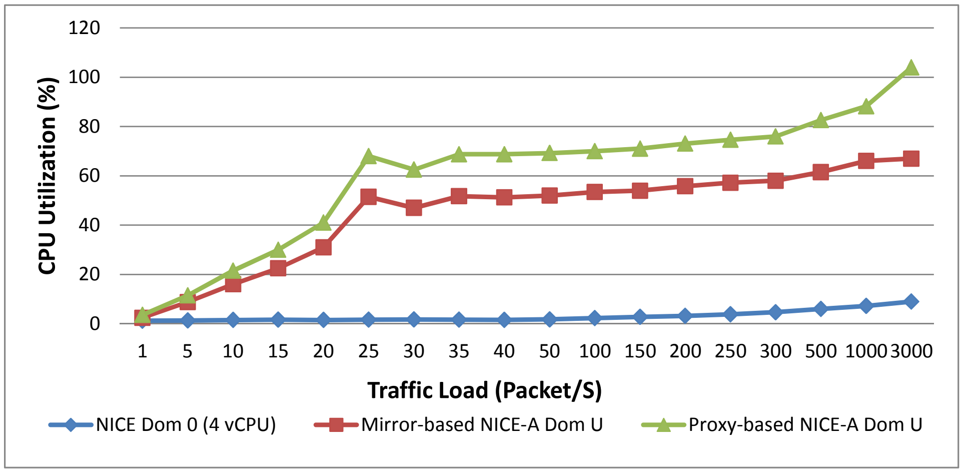


Figure 3.11: CPU Utilization of NICE-A.

Figure 5.7, represents the performance of NICE-A in terms of percentage of successfully analyzed packets,i.e., the number of the analyzed packets divided by the total number of packets received. The higher this value is, more packets this agent can handle. It can be observed from the result that IPS agent demonstrates 100% performance because every packet captured by the IPS is cached in the detection agent buffer. However, 100% success analyzing rate of IPS is at the cost of the analyzing delay. For other two types of agents, the detection agent does not store the captured packets and thus no delay is introduced. However, they all experience packet drop when traffic load is huge.

In Figure 3.13, the communication delay with the system under different NICE-A is presented. We generated 100 consecutive normal packets with the speed of 1 packet per second to test the end-to-end delay of two VMs compared by using NICE-A running in mirroring and proxy modes in DomU and NICE running in Dom0. We record the minimal,

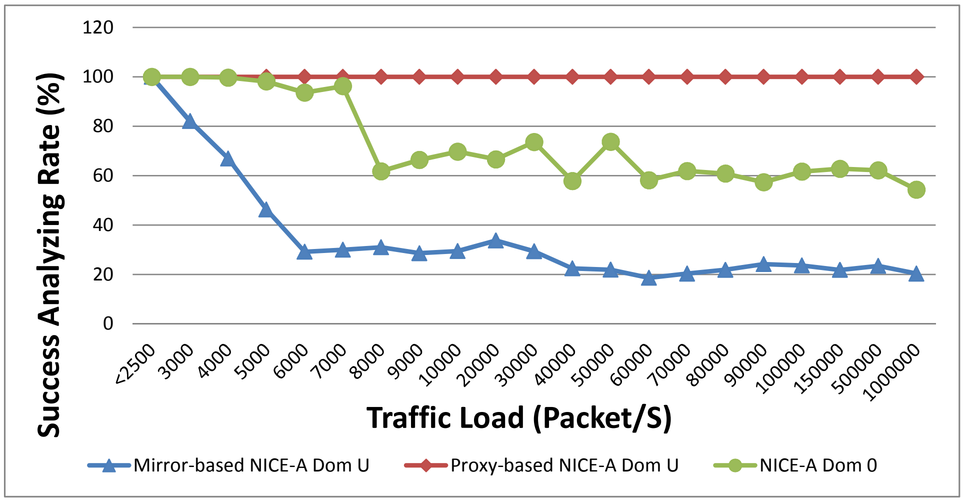


Figure 3.12: NICE-A Success Analyzing Rate.

average, and maximum communication delay in the comparative study. Results show that the delay of proxy-based NICE-A is the highest because every packet has to pass through it. Mirror-based NICE-A at DomU and NICE-A at Dom0 do not have noticeable differences in the delay. In summary, the NICE-A at Dom0 and Mirror-based NICE-A at DomU have better performance in terms of delay.

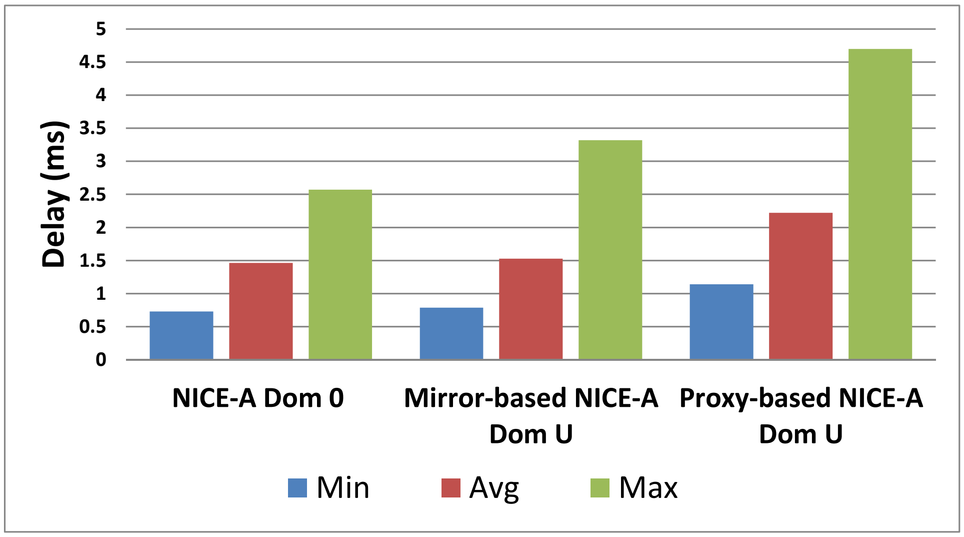


Figure 3.13: Network Communication Delay of NICE-A.

From this test we expected to prove the proposed solution, thus achieving our goal ”establish a dynamic defensive mechanism based software defined networking approach that involves multiphase intrusion detections”. The experiments prove that for a small-scale cloud system, our approach works well. The performance evaluation includes two parts. First, security performance evaluation. It shows that the our approach achieves the design security goals: to prevent vulnerable VMs from being compromised and to do so in less intrusive and cost effective manner. Second, CPU and throughput performance evaluation. It shows the limits of using the proposed solution in terms of networking throughputs based on software switches and CPU usage when running detection engines on Dom 0 and Dom U. The performance results provide us a benchmark for the given hardware setup and shows how much traffic can be handled by using a single detection domain. To scale up to a data center level intrusion detection system, a decentralized approach must be devised, which is scheduled in our future research.

## 3.7 Conclusion and Future Work

This framework is able to detect and mitigate collaborative attacks in the cloud virtual networking environment. NICE utilizes the attack graph model to conduct attack detection and prediction. The proposed solution investigates how to use the programmability of software switches-based solutions to improve the detection accuracy and defeat victim exploitation phases of collaborative attacks. The system performance evaluation demonstrates the feasibility of NICE and shows that the proposed solution can significantly reduce the risk of the cloud system from being exploited and abused by internal and external attackers.

NICE only investigates the network IDS approach to counter zombie explorative attacks. To improve the detection accuracy, host-based IDS solutions are needed to be incorporated and to cover the whole spectrum of IDS in the cloud system. This should be investigated in the future work. Additionally, as indicated in the paper, we will investigate the scalability of the proposed NICE solution by investigating the decentralized network control and attack analysis model based on current study.

Chapter 4

# NON-INTRUSIVE PROCESS-BASED MONITORING SYSTEM TO MITIGATE AND PREVENT VM VULNERABILITY EXPLORATIONS

## 4.1 Introduction

Security is one of the concerns that still make people think twice before migrating to the cloud. Virtualization introduces several attack surfaces for the cloud, like hypervisor, virtual machines (VMs), virtual network [52] to name a few. Among them, VMs are the most important resources for user and the most vulnerable target for the attacker. In traditional data centers, where system administrators have some control over the host machines, vulnerabilities can be detected and patched. However, patching known security holes in a cloud, where customers usually have the privilege to control the software installed on their VMs, may not work effectively and any action by the administrator might violate the Service Level Agreement (SLA) [53]. Furthermore, these vulnerable VMs are not only harmful to their users, but also pose a threat to other VMs. The challenge is to establish an effective detection and response system for accurately identifying vulnerabilities and malicious processes on users' VMs, rapidly detecting attacks from internal and external network, and efficaciously minimizing the impact of security breaches to cloud users.

Detection and removal of malicious code is not easy unless each VM is equipped with malicious code detection and monitoring tool. In worse case, malicious code will reside stealthy somewhere in the cloud system and such a hidden malicious code becomes a ticking bomb. If the malicious code is controlled by a remote attacker, s(he) is able to recruit other vulnerable VMs and launch a multi-step or coordinated Distributed Denied of Service (DDoS) attack. Goal for the attacker is to get control of a target VM. After controlling the VM, a remote attacker is able to monitor, intercept, and change the state and actions of other software on the system. The controlled malicious code is also able to hide itself and even disable the host intrusion detection system, which is a significant threat to a cloud. Therefore, malicious code detection and mitigation is a very important security tool for protecting VMs from being controlled by attackers.

Vulnerable VMs are ports of entry for a variety of security threats such as DDoS attacks, spamming, and malware distribution. Owners of compromised VMs are most often unaware that their systems are being used by someone else. Such a virtual machine whose security has been breached, either by a virus or other means, and consequently allows an unauthorized user to remotely control the system for malicious activities without the owner's knowledge is known as a *Zombie VM*. While the proliferation of Zombie VMs presents a substantial threat to the cloud system and network security, BotNets represent an even grave danger. In recent years, many resources have been dedicated towards the detection of compromised hosts in various domains, but there has been very little focus on the detection and prevention of zombies in the cloud environment.

Traditional methods to detect malware rely primarily on anti-malware agent installed on a VM to search the executable malicious code using pattern matching techniques. With such "in-the-box" agent-based approach, both detecting agent and detected results are visible and vulnerable to an attack, where the malware can temper with the result and disable the agent to hide the malicious code. To address this problem, more and more solutions have been proposed recently to place the malware detection engine outside the VM, then using *VM Introspection* (VMI) technology to detect and monitor the malicious process in the VM [54][55][56]. Such "out-of-box" agent-free approach significantly improves the tamper resistance of malware detection engine [55]. However, introspection causes the so-called *semantic gap*, where the high-level semantic view of guest operating system are missing, e.g. active process, loaded kernel module, and system calls.

To address the problems described above, we propose a hybrid defense-in-depth intrusion detection framework based on our previous work NICE [9] to detect and monitor the vulnerability and attacks in the cloud. For better detection of attacks, our framework incorporates attack graph analytical model to describe the detected vulnerabilities in each VM and creates the vulnerability dependency based on VM's reachability in the virtual network. For the VM-based malicious process detection, we propose a non-intrusive agent-free detection and monitoring tool using VMI technology. In order to address semantic gap challenge, we reconstruct the semantic view to traverse thread dispatched database. We also reconstruct the complete process list of kernel and compare the user-level processes with the kernel-level process using cross-view technology to identify the hidden process.

Using Software Defined Networking (SDN) has been gradually adopted by commercial companies such as Citrix XenServer [19] and VMWare NSX [57]. SDN provides the ability to control the traffic in the virtual network for the QoS purpose, it also can be used to improve security and mitigate the attacks in a virtual cloud networking environment, for example, to build the basic firewalls, VPN, and network-based intrusion framework. We leverage SDN to build a monitor and control plane over distributed programmable virtual switches in order to significantly improve the attack detection and mitigate the attack consequences.

The proposed framework does not intend to improve any of the existing intrusion detection algorithms. Instead, we create this framework based on Xen virtualization platform and establish a system having following features:

- A light-weight network based intrusion detection engine in the dom0 of each cloud server for capturing and analyzing the network traffic.
- Attack graph analytical model for describing vulnerabilities and their dependencies in the cloud system.

- VM process monitor for monitoring and detecting hidden malicious processes in each VM using VMI and semantic reconstruction technologies.
- Countermeasure selection by matching and correlating alerts from intrusion detection engine, vulnerabilities in the attack graph and signal from VM process monitor.
- Deployment of virtual network reconfiguration-based countermeasure through network controller using Software Defined Networking (SDN).

## 4.2 Related Work

Detection of compromised machines currently takes place largely at host level and/or network level. At the host level, while anti-virus and anti-spyware systems are effective in catching and preventing the spread of known threats [58], majority of users do not keep these security software updated or properly configured. At the network level, IDSs and network firewalls fail to address already compromised vulnerabilities within the networks they protect. For example, when a system within the firewall becomes compromised by the careless actions of an authorized user, like allowing a trojan horse access to the internal network, then once such a system has been compromised, a personal firewall loses effectiveness because the attacker has already gained some level of control over the system.

Intrusion Detection System (IDS) attempt to detect and prevent the spread of compromised machines or their attacks by developing characteristic network traffic profiles, called signatures, and using them to identify attacks. However, similar to anti-virus solutions, IDSs are generally effective only as far as the malicious traffic pattern is detectable. In most cases, the intruder is able to continue to evade detection by blending malicious actions and activity with legitimate usage. Recent trends have shown intruders exploiting limitations and vulnerabilities within firewalls and IDS to better conceal the identities of zombies, thus making it harder to detect attacks. While system and network administrators attempt to combat this problem by addressing vulnerabilities in firewalls and anti-virus software as soon as they become known, zombies and their agents have been evolving even faster.

In a virtualized environment, such as a cloud system, it becomes easy for the attacker magnify the loss. In order to prevent the compromised VM from attacking vulnerable VMs due to the possible security hole cased by the resource sharing, the detection and monitor tools are necessary to secure each VM in a cloud system. For the VMM based malware detection, Livewire [54] proposed first concept of placing an“out-of-VM” monitor and applying VMI technology to reconstruct the semantic view of the internal structure of the VM. However, it can only reconstruct low-level VM states (e.g., disk blocks and memory pages). The high-level VM states (e.g., processes, kernel module, and files) still require an intrusive way to bridge the semantic gap. VMwatcher [55] is another “out-of-box” approach that overcomes the semantic gap created by the missing information about detailed internal view of the system by “in-the-box” approaches. To close the semantic gap, it applies a technique called ‘guest view casting’ and non-intrusively reconstructs the high-level internal VM semantic views from outside. However, it VMwatch only focuses on the malware detection in VMs and cannot monitor or detect the security status in the virtual network.

Antfarm [59] is an implementation of VMM-based introspection techniques that tracks the activities of processes in VMs by monitoring low-level interactions between guest operating systems (OS) and their memory management structures. It can determine when a guest OS creates, destroys, or context-switches the processes without explicit information about the OS. Lycosid [60] is a VMM-based hidden process detection and identification service. It uses Antfarm to obtain a trusted view of guest OS processes, invokes a user-level program to obtain a untrusted view of these processes, and then uses cross-view validation principle to detect malicious hidden OS processes. The drawback of Lycosid is that it uses an intrusive approach to obtain the processes information from guest OS. Maitland [61] introduced a light-weight introspection technique in absence of hypervisor, that provides the same levels of within-VM observability. To address isolation challenges in out-of-VM approaches, Out-Grafting [62] was proposed. It allows fine-grained process-level execution monitoring. Our framework uses non-intrusive approach to monitor and detect the internal process of VMs and identifies the suspicious processes with the help of attack graph model.

To model possible vulnerabilities and their dependencies in a cloud, we use attack graph. An attack graph is able to represent atomic attacks in a network and describe possible attack paths for attackers to reach his/her goal, which we consider is to get root privileges on a particular VM. Various tools for attack graph generation and analysis have been proposed. One such tool was proposed by O. Sheyner *et al.* [29][63]. To tackle scalability issues with attack representations, P. Ammann *et al.* [64] assumed attacks to be monotonic in nature, which allows attackers not to backtrack. TVA (Topological Vulnerability Analysis) tool developed by Jajodia [65] was another attack graph generation tool but it required some initial input by hand. MulVAL [32] was proposed by X. Ou *et al.* in which they utilized Datalog representation of network state and attack scenarios. We adapt the MulVAL approach to create our version of attack graph, Scenario Attack Graph (SAG).

## 4.3 System Overview and Models

In this section, we present the design goals, discuss the assumption, and describe models proposed in this work.

### *4.3.1 Design Goals and Assumption*

We establish a hybrid intrusion detection framework to detect and monitor the malicious traffic in the network and malicious process in each VM using VMI technology. To achieve that, we have following design goals:

- The framework should be able to capture all of vulnerabilities in the cloud system and enumerate all possible attack paths after analyzing the vulnerabilities and their dependencies.
- The framework should be able to detect malicious processes in VMs immediately when the process is created, but such detection shall not be done by any piece of code running on VM.
- The framework should be able to select the optimal countermeasure and deploy it before the attacker takes the next exploitation step.

In this work, we assume that the hypervisor is trusted and secured, which means the hypervisor properly isolates the resources and environment for the running VMs. The hypervisor is protected against any exploits launched by the attacker and the detection engine installed in the hypervisor is invisible to the attacker. We also assume that users are able to install vulnerable software and execute any malware or malicious code in their VM. System administrator cannot patch the software or remove the malicious code without users agreement. However, the Cloud Service Provider (CSP) allows to block the traffic issued by such processes. Furthermore, the VM outage caused by cloud system reliability [66] is out the scope of this work.

### *4.3.2 Attack Graph Model*

An attack graph is a modeling paradigm to illustrate all possible attack paths in a network that can be exploited by internal or external attackers. It is a crucial model to understand threats and then to decide appropriate countermeasures in a protected network [20]. In an attack graph, each node represents a condition or an action. A condition node is a precondition and/or a consequence of an exploit. It represents a system configuration or an access privilege that should be true in order to exploit any other vulnerabilities. An action node is a step that attacker exploits an existing vulnerability in order to compromise a VM. It depends upon existence of one or more conditions along the path and is not necessarily an active attack since a normal protocol interaction can also be used for an attack.

As the attack graph lists all known vulnerabilities in the system and the connectivity information, one can get a whole picture of current security situation of the system. We can then see the possible threats and attacks by correlating detected events or activities with that depicted by the attack graph. Attack graph is thus helpful in identifying potential threats, possible attacks and known vulnerabilities in a cloud system. Once an event is recognized as a potential attack, attack graph tells important information about how the attacker can utilize that event the damage it can cause to other machines. With this information in hand we can apply specific countermeasures to mitigate a malicious event impact and take actions to prevent it from contaminating other virtual machines.

Figure 4.1 shows a simple example of an attack graph. The left hand side of the figure shows a simple network topology with three VMs. VM $A$ contains two vulnerabilities, $v1$ and $v2$. $v2$ can only exploited by an attacker if (s)he has exploited $v1$ and obtained the required privilege to exploit $v2$. VM $B$ and $C$ has $v3$ and $v4$ vulnerabilities respectively. The right hand side of the figure shows the generated attack graph corresponding to the network topology in the figure. Oval nodes represent attacker's action to exploit a vulnerability. Diamond nodes represent precondition of exploiting next vulnerability on the path and/or the consequence (post-condition) of an exploit. It shows that there are two possible attack paths to reach the goal which means to compromise VM $C$.

### *4.3.3 Suspend, Check, and Forwarding Model*

The attack graph model enumerates all of possible threats in the system, and presents an analogy of a map to list different paths from source or attacker to destination or target. With the help of intrusion detection agent, we can identify where currently the attacker is on

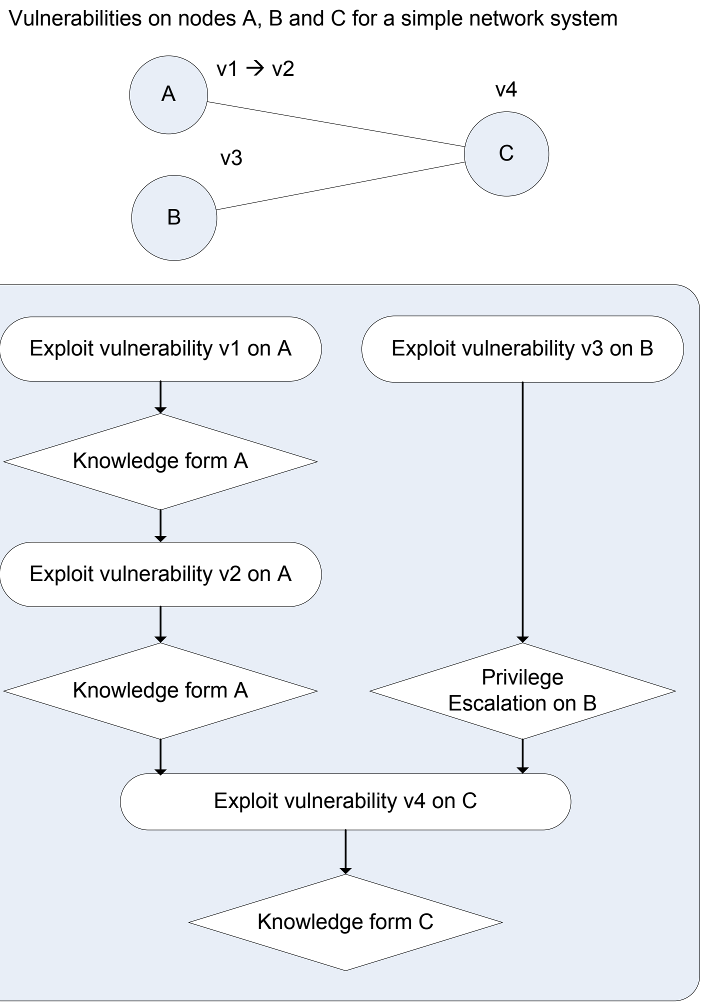

Vulnerabilities on nodes A, B and C for a simple network system
A
v1 → v2
B
v3
C
v4
Exploit vulnerability v1 on A
Exploit vulnerability v3 on B
Knowledge form A
Exploit vulnerability v2 on A
Knowledge form A
Privilege Escalation on B
Exploit vulnerability v4 on C
Knowledge form C

Figure 4.1: A Simple Attack Graph Example.

the attack graph. The alert raised by intrusion detection agent reveals that the traffic from a source to a destination with a certain protocol is suspicious. In order to detect the malicious payload, we propose a *suspend-check-forwarding* model to deter attacker's actions. Out system does so by detecting the infected process and stopping its communication with other processes in different VMs. We integrate the attack graph, VMI technology, and programmability feature of SDN to design our inspection model for tracking the available attack paths, monitoring the internal process of a guest VM, and suspending the traffic to a destination VM for further inspection.

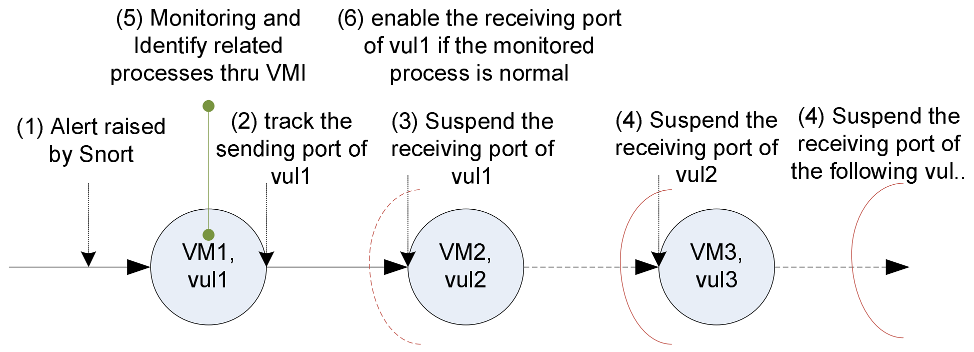

(5) Monitoring and Identify related processes thru VMI
(6) enable the receiving port of vul1 if the monitored process is normal
(1) Alert raised by Snort
(2) track the sending port of vul1
(3) Suspend the receiving port of vul1
(4) Suspend the receiving port of vul2
(4) Suspend the receiving port of the following vul..
VM1, vul1
VM2, vul2
VM3, vul3

Figure 4.2: Suspend, Check, and Forwarding Model.

The concept of this model is shown through figure 4.2. In order to verify the activities of internal process inside a possible victim and prevent the malicious code from further spreading out to infect other VMs, our system tracks the traffic that is generated from the port of the malicious process and whose IP address is the source of the alert message raised by a detection agent (step 1 and 2 in figure 4.2). After that, the system suspends the traffic on the receiving port of vulnerability on the destination VM after the suspicious node (step 3), and continually suspends the following interaction ports of vulnerabilities lying on the same attack path (step 4). Such suspended ports belong to applications susceptible to attacks. After suspending the traffic, monitoring on the infected VM begins to look for malicious processes (step 5). If a malicious process is detected, the traffic is blocked by the network controller in SDN, otherwise the traffic suspension is cleared and the receiving port on the destination VM is enabled (step 6).

## 4.4 System Design

In this section, we first present the system architecture of our design and then detailed descriptions of its components.

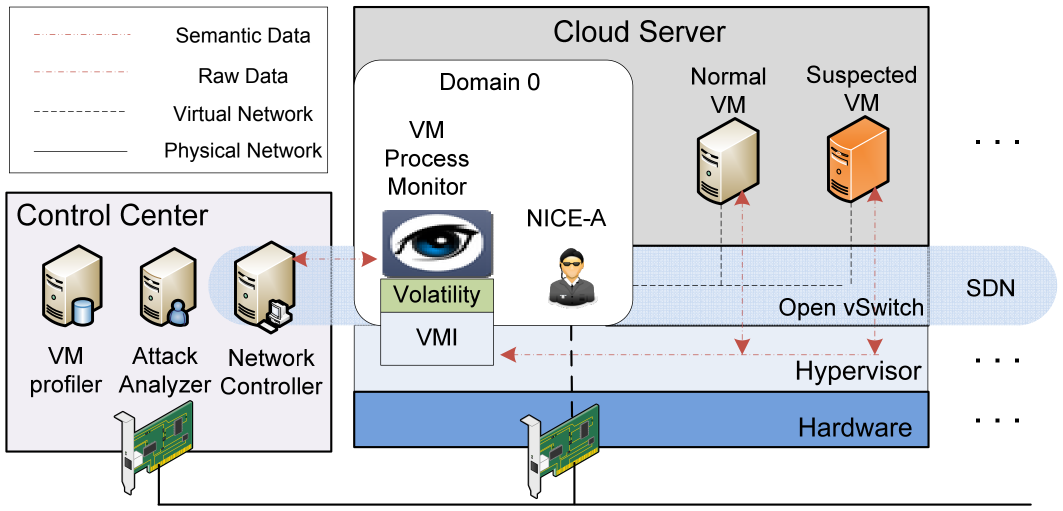


Figure 4.3: System Architecture.

### *4.4.1 System Architecture*

The proposed system is designed to work in a cloud virtual networking environment. It consists of a cluster of cloud servers and their interconnections. We assume that the latest virtualization solutions are deployed on cloud servers. The virtual environment can be classified as Privilege Domains, e.g., the dom0 of XEN Servers [19] and the host domain of KVM [67], and Unprivileged Domains, e.g., VMs. Cloud servers are interconnected through programmable networking switches, such as physical OpenFlow Switches (OFS) [21] and software-based Open vSwitches (OVS) [68] deployed in the Privilege Domains. In this work, we refer OFSs and OVSs and their controllers as to the Software Defined Network (SDN). The deployed security mechanism focuses on providing a non-intrusive approach to prevent attackers from exploring vulnerable VMs and use them as a stepping stone for further attacks.

The system architecture of our solution is illustrated in Figure 4.3. The control center consists of a network controller, a VM profiler, and an attack analyzer.

A network intrusion detection engine NICE-A can be installed in either Dom0 or DomU of a XEN cloud server. Its job is to capture and filter malicious traffic. Alerts from NICE-A are sent to control center upon detection of anomalous traffic. After receiving an alert, attack analyzer evaluates the severity of the alert based on the attack graph. It then initiates countermeasures through the network controller after deciding what countermeasure strategies to take.

As described in [9], countermeasures initiated by the attack analyzer are based on the evaluation results from the cost-benefit analysis of the effectiveness of countermeasures. The network controller initiates countermeasure actions by reconfiguring virtual or physical OpenFlow switches. We must note that the alert detection quality of NICE-A depends on the implementation of NICE-A that uses Snort. We do not focus on the detection accuracy of Snort in this paper. Dom0 consists of VM Process Monitor which uses VMI to monitor running processes on VMs.

### *4.4.2 Attack Analyzer*

Attack Analyzer is a centralized information process center to process the security-related information and has the whole picture of the security status of the monitoring cloud server cluster. The major tasks of this component include collecting and processing information about the identified alerts, suspicious traffic and suspected processes from each VM Process Monitor, selecting the best countermeasure based on the knowledge of current attacks and system status, and sending the commands to the Network Controller for countering or mitigating the attack. These functionalities are realized by four subcomponents: Attack Graph analysis model, countermeasure selection, and VM profiler.

The Attack Analyzer handles alert correlation and analysis. This component has two major functions: (1) constructs Attack Graph, (2) provide threat information and appropriate countermeasures to network controller for virtual network reconfiguration.

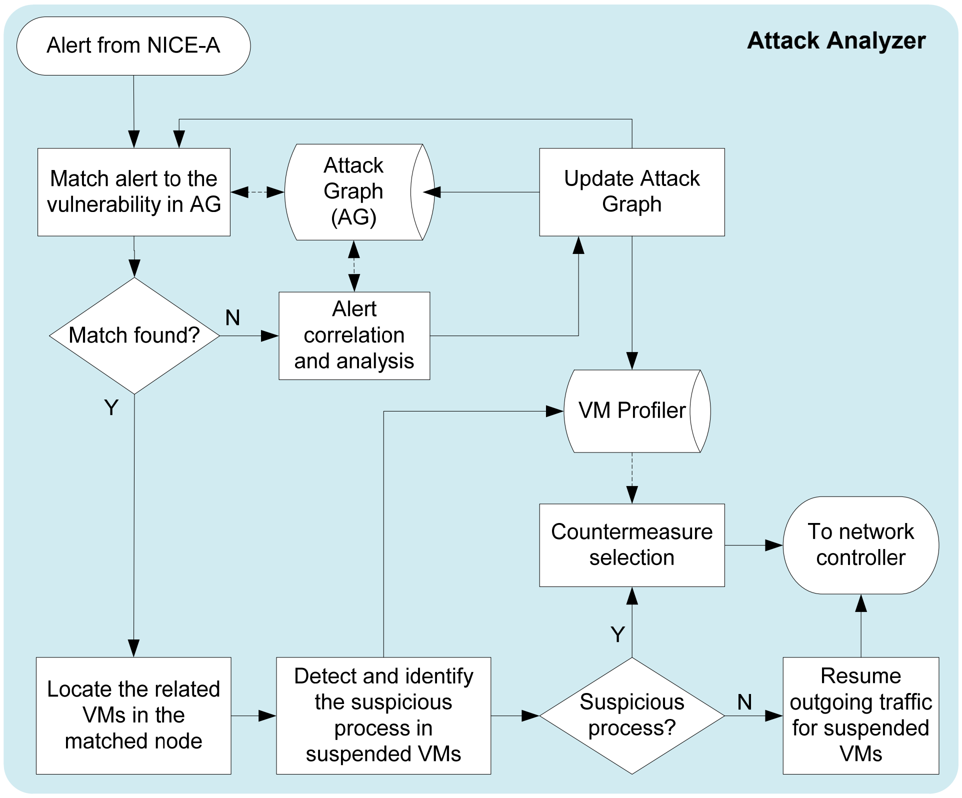

Attack Analyzer
Alert from NICE-A
Match alert to the vulnerability in AG
Attack Graph (AG)
Update Attack Graph
Match found?
N
Y
Alert correlation and analysis
VM Profiler
Countermeasure selection
To network controller
Y
Locate the related VMs in the matched node
Detect and identify the suspicious process in suspended VMs
Suspicious process?
N
Resume outgoing traffic for suspended VMs

Figure 4.4: Workflow of the Attack Analyzer.

Figure 4.4 shows the workflow in the attack analyzer component. Alert received from NICE-A is checked against the vulnerability in the attack graph (*AG*). If a match is found, i.e. the vulnerability corresponding to the alert already exists in the attack graph then it can be regarded as a known attack matching with signature in the alert message. If no alert matches in the AG, then alert correlation and analysis is performed and AG is updated. However, for a matching alert, Attack analyzer locates the VM in the matched node based on the destination IP address in the alert. The VM Process Monitor then performs the inspection action on the corresponding VM using VMI to detect and identify the suspicious process with reference to the VM profiler. If a process is identified as suspicious, a selected countermeasure is applied by the network controller based on the severity of evaluation results. If the inspected process is found to be harmful, appropriate countermeasure is applied by the network controller, otherwise the outgoing traffic is resumed from the suspended VM.

### *4.4.3 Attack Graph*

To keep track of all possible attack in the cloud, AA maintains an attack graph analysis model to analyze vulnerabilities and their relationship from all monitored VMs. The related tasks to the attack graph include constructing and updating the attack graph when a vulnerability is patched or the network topology is changed, correlating alerts with attack graph, predicting attacks, managing the VM Profiler, and selecting the optimal countermeasure. The attack graph is pre-constructed based on the following information:

- *Cloud system information*: it is collected from each PD. The information includes the number of VMs in each cloud server, the running services on each VM, and VM's Virtual Interfaces (VIFs) information.

- *Virtual network topology and configuration information*: NC collects this information, that includes virtual network topology, host connectivity, VM connectivity, every VM's IP address, MAC address, port information, and traffic flow information.

- *Vulnerability information*: it is generated by both on-demand vulnerability scanning and regular penetration testing using the well-known vulnerability databases, such as Open Source Vulnerability Database (OSVDB)[43], Common Vulnerabilities and Exposures List (CVE)[41], NIST National Vulnerability Database (NVD) [44], etc. Such scanning can be initiated by the NC and VM process monitor.

Many alert correlation techniques have been proposed [34][35][69] to reduce the false detection rate. In our framework, alert correlations and analysis are also handled by Attack Analyzer in the control center. This component has two major functions: (1) correlate alerts and integrate them into the attack graph model, (2) provide threat information or countermeasure to Network Controller for virtual network reconfiguration or further inspection.

### *4.4.4 Network Controller*

Network controller is the main component to conduct the VM oriented (high level) countermeasure on the suspicious and malicious traffic based on the decision from Attack Analyzer. Network controller is the key component to support programmable network using OpenFlow protocol [39]. In our framework, each cloud server has a software switch, i.e., implemented by using Open vSwitch (OVS) [68] as the edge switch to handle all of traffic to and from VMs. The communication between cloud servers (i.e., physical servers) is handled by OFS. Both OVS and OFS are controlled by the Network Controller, allowing the controller to set security/filtering rules on both OVS and OFS. Network Controller is also responsible for collecting network information of current OpenFlow network, and provides inputs to Attack Analyzer to construct attack graphs.

### *4.4.5 VM Profiler*

VM Profiler keeps tracking the security-related status of each VM. These profiles are necessary for the AA to identity suspicious events. We use three lists to record the security status of the processes for VMs in the cloud.

- *Frequently Compromised Process List (FCP)*: FCP is a list of processes related to well-known vulnerabilities in CVE, NVD, and OSVDB because these vulnerabilities are easy to be compromised by zero-day attacks, for example, IExplorer.exe, Acrobat.exe, WinRAR.exe, WINWORD.exe and so on. FCP is a public list for all VMs.
- *Blacklist (BL)*: BL is a list of malicious processes that have been identified by a PI. The process in BL is not allowed to establish a communication channel to other VMs.

- *Whitelist (WL)*: WL is a list of processes that have not been identified as suspicious. It maintains the process which without any malicious behavior has been found for a certain period of time.

### *4.4.6 VM Process Monitor*

Detecting the hidden malicious process is a very important task for securing the system. For the malicious process detection, traditional methods primarily rely on the detection and monitoring agent installed in the protected system. However, such systems have drawbacks like system integrity is not preserved, detection and monitor agent is easy to be attacked, and the correctness of the detection result cannot be guaranteed. To address these problems, placing the detection and monitor agent out of the protected VM is reasonable.

Virtualization Technology equips VM with three properties: isolation, encapsulation, and privilege. Due to these properties, it is easy to deploy the detection and monitoring agent in the virtualization platform. In this article, we focus on XEN virtualization platform only. We place the detection and monitor agent out of the protected VM, use VMI technology and semantic reconstruction to traverse thread dispatched database, and to reconstruct the complete process list of kernel and compare the user-level process with the kernel-level process using cross-view technology to identify the hidden process.

Figure 4.5 shows VM Process Monitor. We develop five sub-modules in this monitor: security console, daemon, VMI, semantic reconstruction, extractor and executor. Security console is an interactive console for administrators to setup the VM to be monitored and configure the monitor strategies and rules. These configuration messages will be sent to the daemon module with a command and displayed on the console to notify the administrators or serve as a log. Daemon, extractor and executor are all supporting functions for VMI and reconstruction modules to pass the messages and commands between VMM and VM Process Monitor.

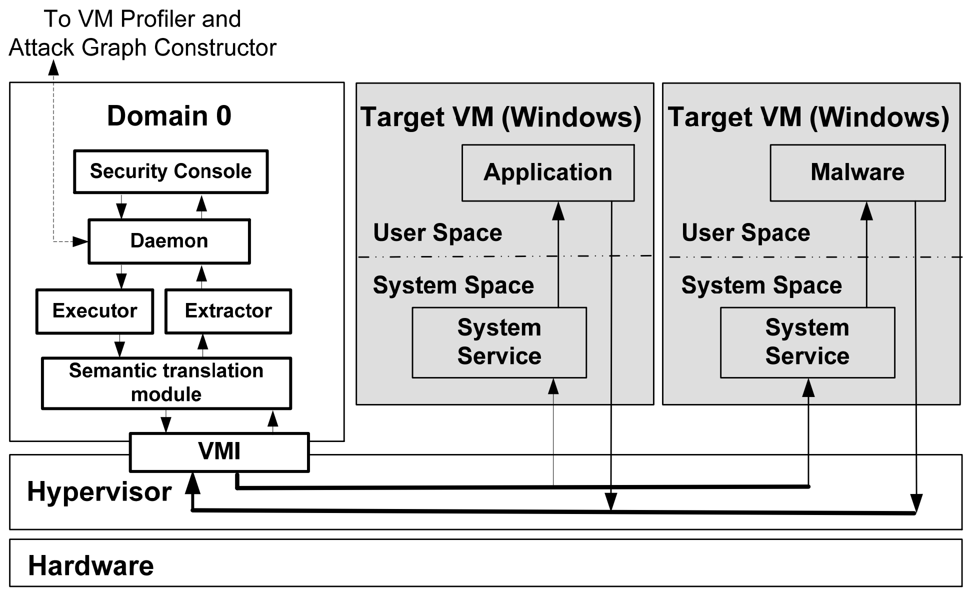


Figure 4.5: VM Process Monitor.

VMI module is responsible for mapping the addresses between the application process' virtual address space in the VM and the actual physical address space through two-layer mappings. The first layer translates the Guest Virtual Address (GVA) to the Guest Physical Address (GPA). The second layer translates the GPA to the Host Physical Address (HPA). By reading the GVA from the outside of VM, VMI is able to read the content of the memory information in the VM through the translation from VM's GVA to VM's HPA.

In a virtualized platform, the virtual machine monitor (VMM) or hypervisor can only read VM's internal virtual hardware information, which is a large volume of unreadable raw data. It lacks meaningful semantic view of internal structures in the guest operating system and it is also hard to directly map the current running status of the system. This situation is often referred as "sematic gap" problem. The semantic translation module is the one for addressing the sematic gap issue. It is responsible for reconstructing the kernel-level process of a VM and recovering the VM's virtual hardware information from the information captured by the VMI.

For example, in order to retrieve the socket and network connection information of processes in a MS Windows VM, the VM process monitor locates the _ADDRESS_OBJECT and _TCPT_OBJECT data structures in VM's memory, which is pointed by _AddrObjTable and _TCBTable in PE (Portable and Executable) header of the kernel module tcpip.sys. The VM process monitor traverses the linked list of _ADDRESS_OBJECT and _TCPT_OBJECT data structures to identify all the in-use sockets and active network connections, and most importantly, the processes they belong to.

## 4.5 Countermeasure Strategies

We consider the countermeasure strategies at both network level and host level. The network level countermeasures primarily rely on network reconfiguration strategies through SDN. It contains countermeasures such as traffic isolation, deep packet inspection (DPI), MAC address rewrite, IP address rewrite, network topology change, port blocking, and rate-limiting, as well as traffic drop, redirect, and suspend. In this work, we use port blocking and traffic suspension as network level countermeasure strategies for the proof of concept.

As for the host-level countermeasure strategies, our system mainly involves two levels of actions: VM-level network reconfiguration and process-level network reconfiguration. Virtual switch, i.e., the OVS in a XEN system, are the main component in the virtual networking of a cloud system for the VM connectivity. A VM in the XEN environment is connected to a virtual bridge in the OVS through the Virtual Interfaces (VIFs) attaching to the VM. VMs on different bridges are isolated at layer-2. Even the traffic between VMs on the same bridge is under the control of the virtual switch. Our framework utilizes this layer-2 traffic management capability to propose a VM-level network reconfiguration strategy. The VM-level reconfiguration strategy can either disable the suspicious VM's VIF to isolate it from other VMs or force the suspicious traffic redirect to an in-line mode intrusion detection system for further deep-packet inspection.

Process-level network reconfiguration provides a fine-grained and scrutinized control over the network connections. With the processes and sockets information of a VM reported from VM Process Monitor, the Network Controller is able to make OVS to isolate the traffic issued by the inspected processes or to redirect the traffic to an in-line mode intrusion detection system for further inspection before delivering to the destination. The advantage of the process-level network reconfiguration is that all other network services remain untouched while the activity of the suspicious process is monitored or isolated.

In summary, the host level countermeasure strategies include:

1. *VMIsp*: VM-level Inspection put a suspected VM under the inspection of a in-line mode intrusion prevention system (IPS).

2. *VMIso*: VM-level Isolation disables all network traffic to and from a suspected VM.

3. *PrIsp*: Process-level Inspection redirects the traffic of a suspicious process in the VM to a in-line mode IPS.

4. *PrIso*: Process-level Isolation prevents the suspicious process in a suspected VM from communicating with other VMs while the network services from other processes stay unaffected.

## 4.6 Evaluation

### *4.6.1 Case Study for Security Performance Analysis*

To demonstrate the security performance of our framework, we created an attack scenario to evaluate our network intrusion detection system with attack graph model and non-intrusive process-based monitoring system with VMI technology.

Figure 4.6 shows the test network topology consisting of three users. User1 and User2 acquire a VM for their workstations respectively. They are all connect to the common vir-

Table 4.1: Vulnerabilities in the Test Network.

| Owner | Host | Vulnerability | CVE ID |
|---|---|---|---|
| User1 | Workstation | Internet Explorer | CVE2009-1918 |
| User2 | Workstation | none | none |
| User3 | Web Server | Apache HTTP service | CVE2006-3747 |
| User3 | Database Server | MySQL database service | CVE2009-2446 |

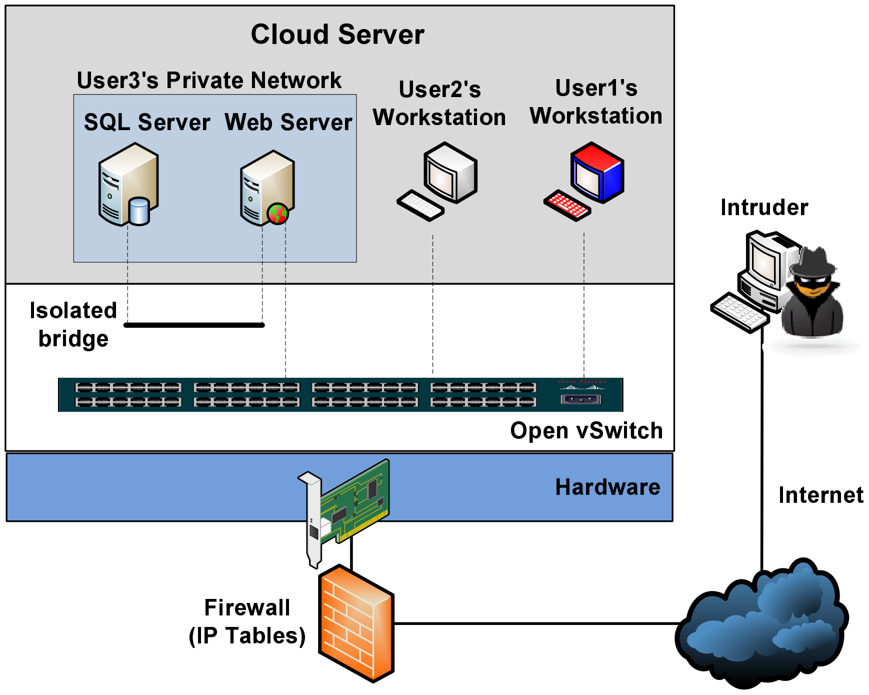


Figure 4.6: Network Topology for the Case Study.

tual network trough OVS. User3 creates a private network to host a database server which can not be accessed directly from external network and a web server which can be accessed from Internet through firewall and virtual network through OVS. Attacker is assumed to be outside the network and has access to the network through Internet. The target for attacker is to get root access on the database server. Table 4.1 lists the vulnerabilities present on the VMs inside the test network.

Attack graph for the test network is shown in Figure 4.7. The original attack graph contains only one attack path which is the path on the right side from node 1 to node 16. Node

1 in attack graph represents the attacker. Node 16 represents the situation where attacker obtains root privilege on database server in the private network of User3 and allows to execute any code on the server which is the attacker's goal. After node 4 has been exploited, it can lead to node 6 and allows attacker to remotely execute malicious code on user VM. The execution of malicious script allows the attacker network access to the database server through tcp on port 3306, as denoted by node 8. From here, the attacker can exploit another vulnerability on node 16 and can gain root access to the database server. Another possible exploitation sequence the attacker can follow is to go for exploitation of vulnerability denoted by node 12 which can lead the attacker to node 13 allowing permission to execute code on the apache webserver.

The attack path on the left side is dynamically created by the attack graph constructor when the vulnerability on the node 4 (CVE-2012-0158) is detected by NICE-A, which means a user in the virtual network has downloaded a malicious file containing the vulnerability of MSCOMCTL.OCX from attacker's website. Whenever the victim opens the downloaded malicious file,a hidden connection is established to the remote attacker. In order to detect if user on a VM has executed the file, we need to enable the suspicious process monitoring and detection module.

For better understanding we grouped attacker's actions as follows:

### Attack Preparation

Attacker places a malicious file on a website and waits for a novice user to download it. The malicious file is capable of connecting back to the attacker to provide a shell control, if it is opened by the victim.

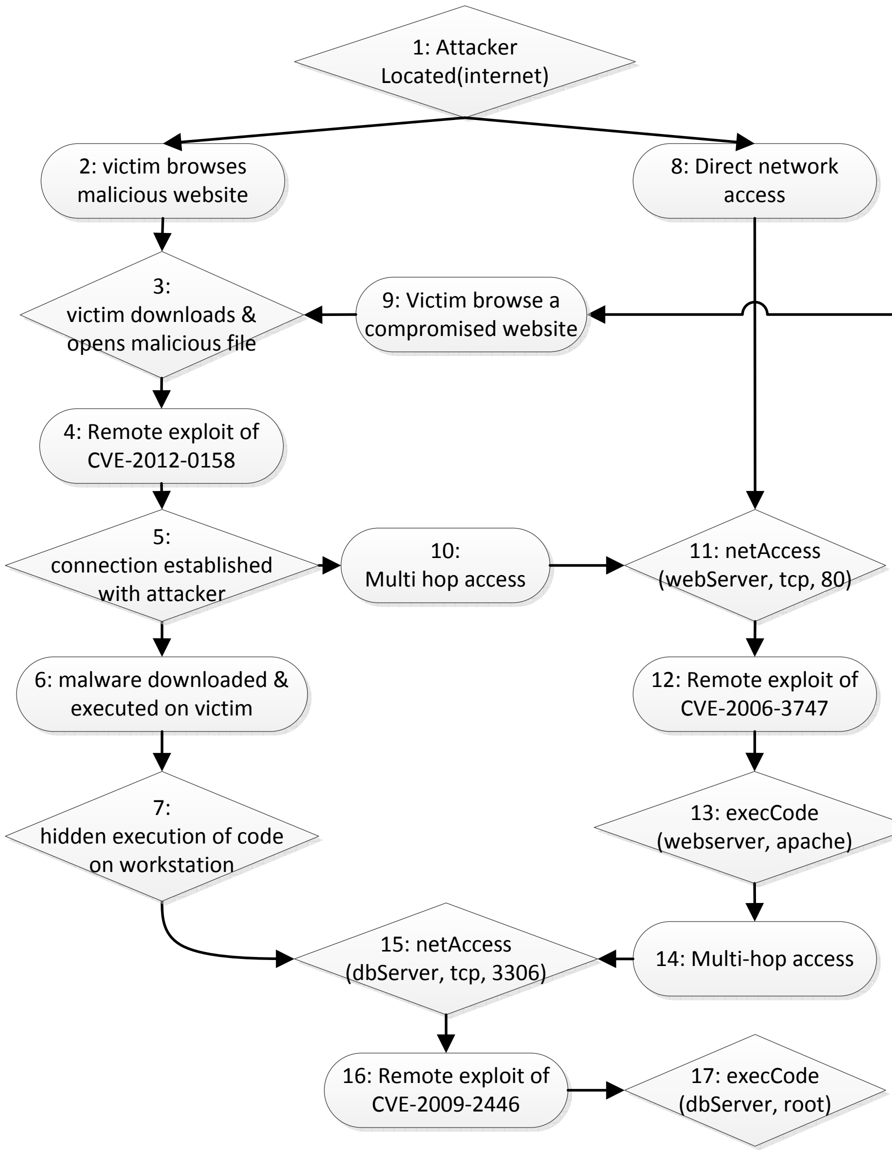
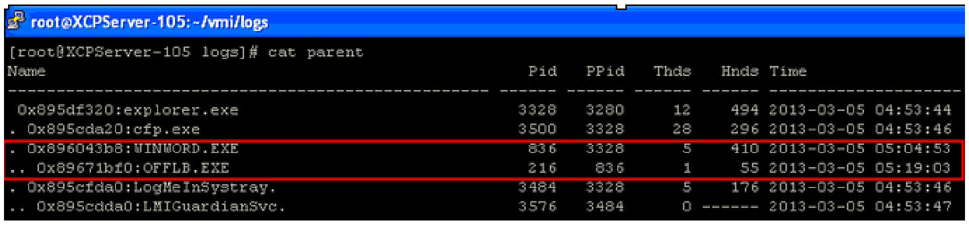

1: Attacker Located(internet)
2: victim browses malicious website
8: Direct network access
3: victim downloads & opens malicious file
9: Victim browse a compromised website
4: Remote exploit of CVE-2012-0158
5: connection established with attacker
10: Multi hop access
11: netAccess (webServer, tcp, 80)
6: malware downloaded & executed on victim
12: Remote exploit of CVE-2006-3747
7: hidden execution of code on workstation
13: execCode (webserver, apache)
15: netAccess (dbServer, tcp, 3306)
14: Multi-hop access
16: Remote exploit of CVE-2009-2446
17: execCode (dbServer, root)

Figure 4.7: Attack Graph for the Test Network.

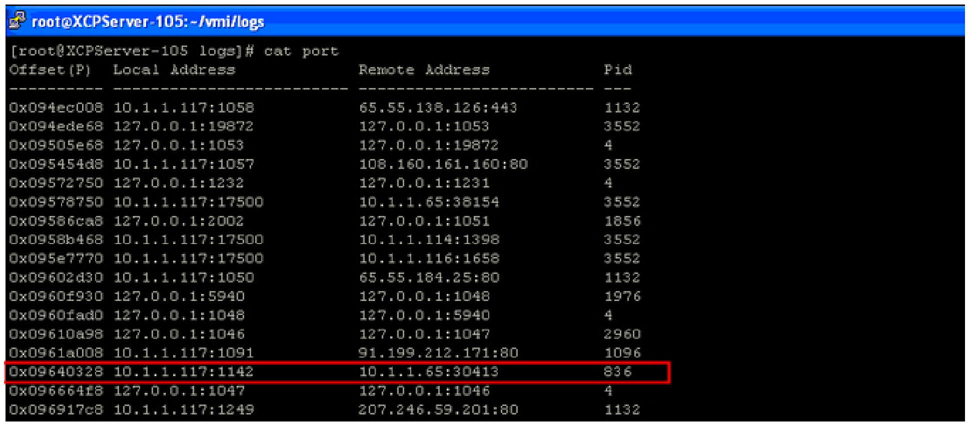
### Exploitation

When a cloud user downloads the malicious DOC file, NICE-A raises an alert to report CVE-2012-0158 vulnerability. At this stage, the receiver VM is not exploited by the remote attacker, until a user opens the file with MS Office 2007. When the file is opened, the embedded shellcode establishes a connection, under the process name WINWORD.EXE, to the remote attacker. Although the connection can be detected by NICE-A, we cannot say it's a malicious behavior. Now, the suspicious process detector (SPD) in the VM Process Monitor is activated to monitor the process using VMI technology. The SPD detects an unknown process created by WINWORD.EXE and tries to make a connection to a remote host. Figure 4.8 shows the process dependency for WINWORD.EXE detected by SPD. Figure 4.9 further details out the network connections established by WINWORD.EXE.



```
root@XCPServer-105:~/vmi/logs
[root@XCPServer-105 logs]# cat parent
Name                                                  Pid   PPid   Thds   Hnds Time
-------------------------------------------------- ------ ------ ------ ------ --------------------
 0x895df320:explorer.exe                              3328   3280     12    494 2013-03-05 04:53:44
. 0x895cda20:cfp.exe                                  3500   3328     28    296 2013-03-05 04:53:46
. 0x896043b8:WINWORD.EXE                               836   3328      5    410 2013-03-05 05:04:53
.. 0x89671bf0:OFFLB.EXE                                216    836      1     55 2013-03-05 05:19:03
. 0x895cfda0:LogMeInSystray.                          3484   3328      5    176 2013-03-05 04:53:46
.. 0x895cdda0:LMIGuardianSvc.                         3576   3484      0 ------ 2013-03-05 04:53:47
```

Figure 4.8: Unknown Process Dependence with WINWORD.EXE.

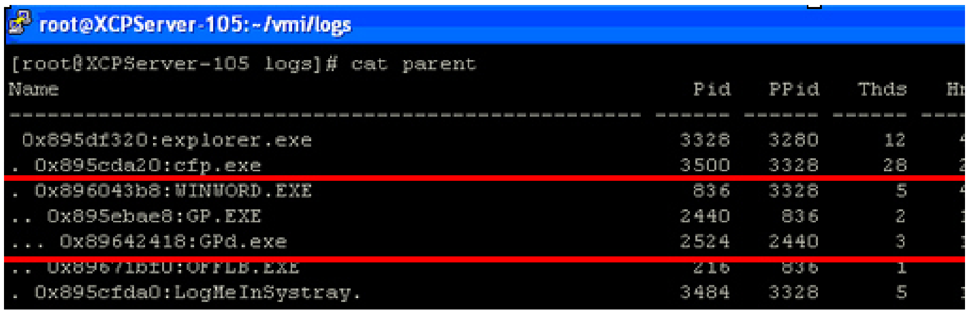


```
root@XCPServer-105:~/vmi/logs
[root@XCPServer-105 logs]# cat port
Offset(P)  Local Address             Remote Address            Pid
---------- ------------------------- ------------------------- ---
0x094ec008 10.1.1.117:1058           65.55.138.126:443         1132
0x094ede68 127.0.0.1:19872           127.0.0.1:1053            3552
0x09505e68 127.0.0.1:1053            127.0.0.1:19872           4
0x095454d8 10.1.1.117:1057           108.160.161.160:80        3552
0x09572750 127.0.0.1:1232            127.0.0.1:1231            4
0x09578750 10.1.1.117:17500          10.1.1.65:38154           3552
0x09586ca8 127.0.0.1:2002            127.0.0.1:1051            1856
0x0958b468 10.1.1.117:17500          10.1.1.114:1398           3552
0x095e7770 10.1.1.117:17500          10.1.1.116:1658           3552
0x09602d30 10.1.1.117:1050           65.55.184.25:80           1132
0x0960f930 127.0.0.1:5940            127.0.0.1:1048            1976
0x0960fad0 127.0.0.1:1048            127.0.0.1:5940            4
0x09610a98 127.0.0.1:1046            127.0.0.1:1047            2960
0x0961a008 10.1.1.117:1091           91.199.212.171:80         1096
0x09640328 10.1.1.117:1142           10.1.1.65:30413           836
0x096664f8 127.0.0.1:1047            127.0.0.1:1046            4
0x096917c8 10.1.1.117:1249           207.246.59.201:80         1132
```

Figure 4.9: Network Connections Created by WINWORD.EXE in Fig. 4.8.

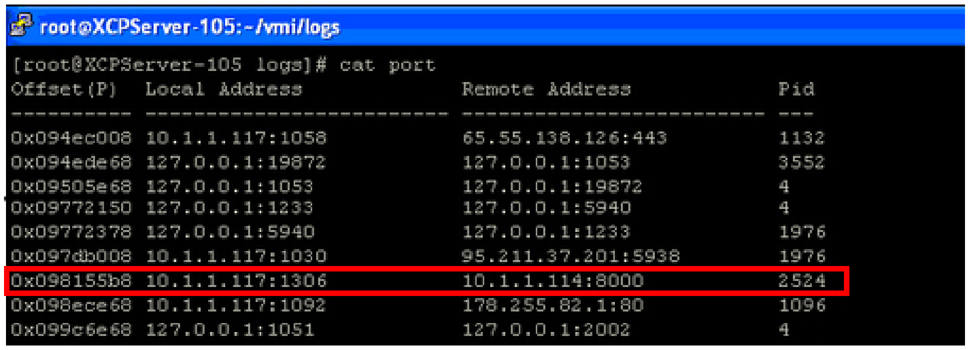
### Persistent Control

To be able to control the compromised VM, attacker takes help of a malware. For instance, malwares like GP.EXE and FU.EXE are used to manage processes remotely and also allow hiding of processes to avoid detection by the user. In our test case, attacker can be anywhere including internal and external network, (s)he also can change the location (attacker's IP



```
root@XCPServer-105:~/vmi/logs
[root@XCPServer-105 logs]# cat parent
Name                                             Pid    PPid   Thds
-------------------------------------------------- ------ ------ ------
 0x895df320:explorer.exe                          3328   3280     12
. 0x895cda20:cfp.exe                              3500   3328     28
. 0x896043b8:WINWORD.EXE                           836   3328      5
.. 0x895ebae8:GP.EXE                              2440    836      2
... 0x89642418:GPd.exe                            2524   2440      3
.. 0x89671bf0:OFFLB.EXE                            216    836      1
. 0x895cfda0:LogMeInSystray.                      3484   3328      5
```

Figure 4.10: Unknown Process Dependency.



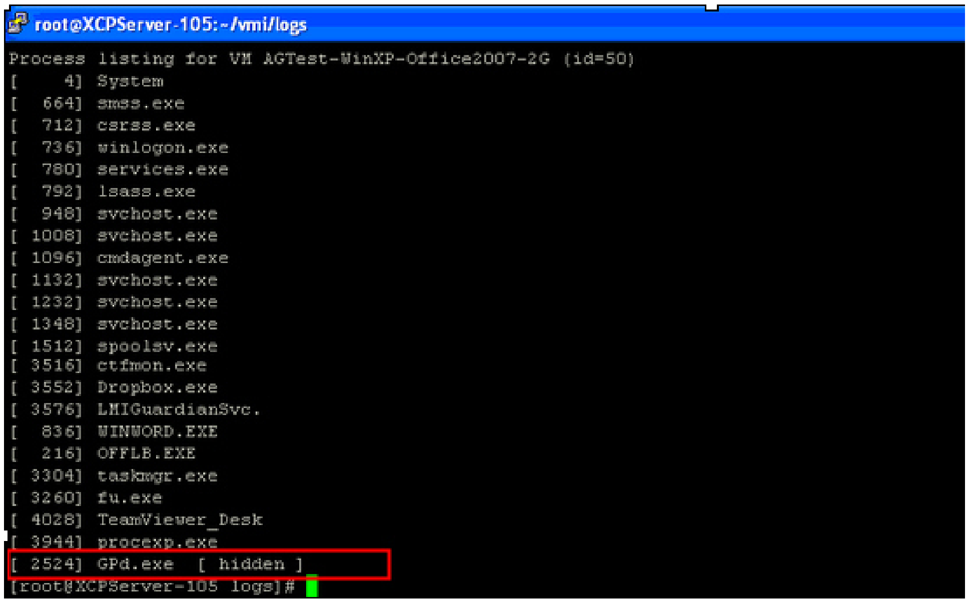
```
root@XCPServer-105:~/vmi/logs
[root@XCPServer-105 logs]# cat port
Offset(P)  Local Address             Remote Address            Pid
---------- ------------------------- ------------------------- ---
0x094ec008 10.1.1.117:1058           65.55.138.126:443         1132
0x094ede68 127.0.0.1:19872           127.0.0.1:1053            3552
0x09505e68 127.0.0.1:1053            127.0.0.1:19872           4
0x09772150 127.0.0.1:1233            127.0.0.1:5940            4
0x09772378 127.0.0.1:5940            127.0.0.1:1233            1976
0x097db008 10.1.1.117:1030           95.211.37.201:5938        1976
0x098155b8 10.1.1.117:1306           10.1.1.114:8000           2524
0x098ece68 10.1.1.117:1092           178.255.82.1:80           1096
0x099c6e68 127.0.0.1:1051            127.0.0.1:2002            4
```

Figure 4.11: Unknown Process Connection.

address) to get hold of command and control anytime. The attacker then transfers these two files stealthy through the reverse connection created by the victim. After GP.EXE is executed on victim VM, it will extract and install a malicious daemon GPd on the victim and sets up a connection to the attacker. The GPd modifies victim's auto run registry to attach itself to autostart script, which guarantees persistent control by the attacker. The attacker now can fully control and manage the processes on the victim VM. Since SPD has been enabled, it can detect an unknown process (GP.EXE) which is created by WINWORD.EXE, as shown in Fig. 4.10. Even though GP.EXE is closed, SPD still be able to detect unknown process GPD.EXE depending on WINWORD.EXE, as shown in Fig. 4.11. We got the experiment result of Fig. 4.8-4.11 with support from volatility project [70].

### Hidden Control

Hiding processes and their dependency is a common strategy for attacker to obfuscate the detection. To make the control hidden from the attacker, FU.EXE is a good tool for attacker to hide the malicious processes. For instance, the GP daemon process (GPd) can be removed from the process chain list to hide the dependence with WINWORD.EXE, so that the malicious daemon can be executed stealthy from user and administration tools, as shown in Fig. 4.10. To prevent attackers to hide themselves using this technology, we develop the SPD with ability to detect the hidden processes and their communication, as shown in Fig. 4.12.



```
Process listing for VM AGTest-WinXP-Office2007-2G (id=50)
[    4] System
[  664] smss.exe
[  712] csrss.exe
[  736] winlogon.exe
[  780] services.exe
[  792] lsass.exe
[  948] svchost.exe
[ 1008] svchost.exe
[ 1096] cmdagent.exe
[ 1132] svchost.exe
[ 1232] svchost.exe
[ 1348] svchost.exe
[ 1512] spoolsv.exe
[ 3516] ctfmon.exe
[ 3552] Dropbox.exe
[ 3576] LMIGuardianSvc.
[  836] WINWORD.EXE
[  216] OFFLB.EXE
[ 3304] taskmgr.exe
[ 3260] fu.exe
[ 4028] TeamViewer_Desk
[ 3944] procexp.exe
[ 2524] GPd.exe   [ hidden ]
[root@XCPServer-105 logs]#
```

Figure 4.12: Detected Hidden Process.

### Countermeasures

In different stages of alert, cloud controller can make the decision of whether to take countermeasure or which countermeasure should be taken according to the alert level. If SPD does not detect any clue that the vulnerability related to the first alert is exploited, Cloud IPS can take the alert as false positive.

### *4.6.2 System Performance*

We performed the evaluation on a cloud server with Intel quad-core Xeon 2.4 GHz CPU and 32G memory. For the non-intrusive suspicious process detection and monitoring system, we created eight VMs running Windows XP SP3. In addition to the default processes in Windows XP SP3 (27 processes), we launched multiple instance of NOTEPAD.EXE to increase the number of process to be detected in each VM for different tests. All tests were performed on different number of VMs (from 1VM to 8VMs) but the same configuration. Six different tests (with different number of process) were performed for each test run. Every test was performed 100 times with the same number of process for all VMs in that test run. The average time elapsed for each test is shown in figure 4.13.

As we can see from figure 4.13, the average process time for each test run approaching to a horizontal line which means the detection time is independent from the number of process. However, the average process time is proportional to the number of VM to be monitor simultaneously, as shown in figure 4.14.

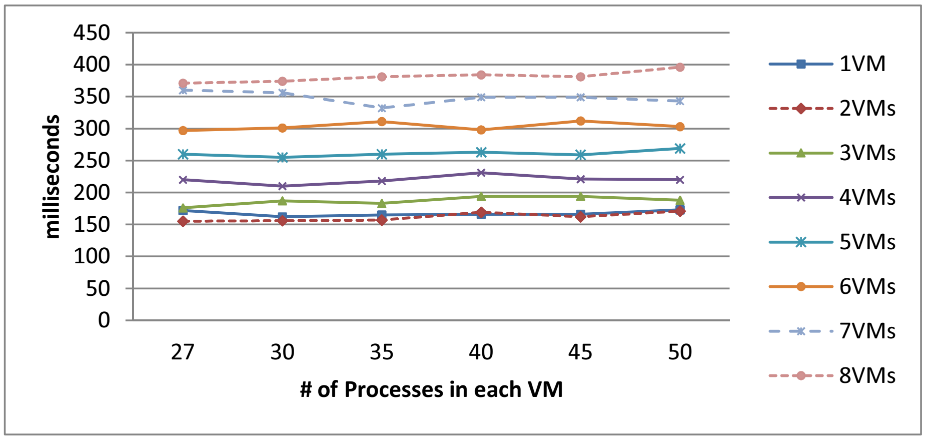


Figure 4.13: VMI Performance Evaluation.

To measure the fine-grained performance impact of our suspicious process detector, we used UnixBench benchmark [71]. The results are shown in figure 4.15. The worst-case

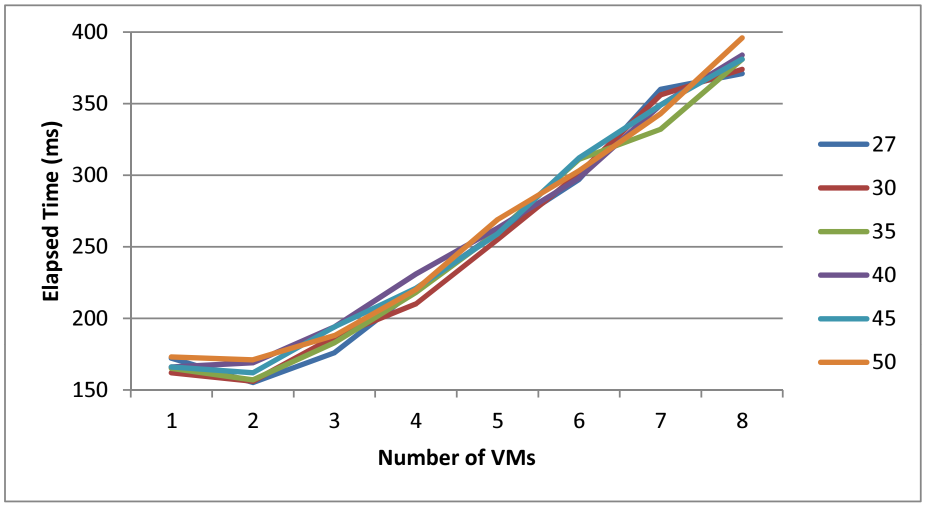


Figure 4.14: VMI Performance Evaluation.

| System Benchmarks Index Values | Unit | w/o SPD | w SPD | overhead |
|---|---|---|---|---|
| Dhrystone 2 using register variables | lps | 956.5 | 910.8 | 4.78% |
| Double-Precision Whetstone | MWIPS | 395.7 | 394.3 | 0.35% |
| Execl Throughput | lps | 298.4 | 281.5 | 5.66% |
| File Copy 1024 bufsize 2000 maxblocks | KBps | 1299.7 | 1236.6 | 4.85% |
| File Copy 256 bufsize 500 maxblocks | KBps | 774.9 | 774.8 | 0.01% |
| File Copy 4096 bufsize 8000 maxblocks | KBps | 2025 | 1837.6 | 9.25% |
| Pipe Throughput | lps | 615.4 | 578.3 | 6.03% |
| Pipe-based Context Switching | lps | 294.4 | 274.3 | 6.83% |
| Process Creation | lps | 204.3 | 193.1 | 5.48% |
| Shell Scripts (1 concurrent) | lpm | 740.5 | 715.5 | 3.38% |
| Shell Scripts (8 concurrent) | lpm | 2217.1 | 2176.3 | 1.84% |
| System Call Overhead | lps | 422.4 | 403 | 4.59% |
| System Benchmarks Index Score |  | 647 | 623.7 | 3.60% |

Figure 4.15: SPD Overhead measurement using Unixbench benchmark.

overhead of our system is 9.25%, while the overheads in most other cases are below 10%. The overall system performance overhead is 3.6% which is a small amount.

## 4.7 Conclusion and Future Work

The system we proposed in this paper integrates a network based intrusion detection system to monitor and detect the traffic in the virtual network and a non-intrusive host based suspicious process monitor and detection system using “out-of-box” VMI technology. Moreover, the host-based intrusion detection is based on VM introspection techniques that do not need the implement special codes in users' VMs.

When hardening network security, hosts cannot be kept apart. Our IDS framework takes care of network security and VM Process Monitor accounts for the security of the host machines. The major benefit for using our framework is to gain the ability to select optimal countermeasures and making the virtual system attack resilient by deploying these countermeasures swiftly. Furthermore, agility of our defense mechanism can be greatly improved with the use of SDN approach and provide an adaptive defense-in-depth approach.

From the case study for the security analysis and system performance we conducted in the evaluation section, it shows that our system is able to capture the malicious traffic and detect the suspicious process related to the alert.

In order to increase the detection accuracy of intrusion and presence of malware in the cloud, we need develop a more sophisticated malware analysis and detection system for our framework in the future to cover different types of VMs, operation systems, vulnerabilities, and attacks. Additionally, our proposed solution suffers from scalability issues since generation of attack graph is complex. We will investigate the decentralized network attack analysis model and/or collaborative approach to address the scalability issues of our framework.

Chapter 5

# DIVINE: DEFENSE IN VIRTUAL INTROSPECTIVE AND NETWORKING ENVIRONMENT

## 5.1 Introduction

Cloud security is one of main concerns for businesses migrating their IT infrastructure to a virtualized and resource-sharing cloud system. Host virtualization and network virtualization are two fundamental techniques to support various cloud-based services. Virtual Machines (VMs) are the basic computing elements in the cloud to establish the Infrastructure-as-a-Service (IaaS) and support both cloud customers and cloud service providers to establish their business functions.

VMs are the main computing resources for cloud users, who may have the root access and install vulnerable applications on them. As a result, VMs can become easy attacking targets. The cloud system administrators usually have the control over the VMs, and thus discovered vulnerabilities can be easily patched. However, patching a discovered vulnerability in a VM may interrupt the users' running services. Moreover, patching a discovered security hole may also cause the dysfunction of the customers' services due to the applications' version dependencies. This situation demands a grace time period before the discovered vulnerabilities being explored by attackers, and thus allows the service providers patching their services accordingly to minimize service interruption to their customers. In this situation, the main research challenge is how to establish an attack-resilient virtual networking environment to safeguard VMs even if they have vulnerabilities that are exploitable by attackers.

To address this challenge, in this paper, we present a dynamic defensive cloud security solution that provides an effective detection and response mechanism for accurately identifying vulnerabilities and malicious processes on users' VMs, rapidly detecting attacks originated from internal and external network, and efficaciously minimizing the impact of security breaches to cloud users.

In IaaS, cloud service providers usually do not restrict cloud users to install or download any software packages, even the software contains malicious code. However, cloud service provider has responsibility to protect cloud users' resources from infection or attacks due to the malicious code and malicious traffic from other malicious users. In such a scenario, traditional detection and protection approaches, such as Network-based Intrusion Detection (NIDS) and anti-malware/anti-virus program, cannot prevent the potential attacks. To effectively control the suspicious traffic in the cloud virtual networking system, the network programming feature of the Software Defined Networking (SDN) [38] is a way to control and manipulate the traffic without affecting the original normal services but also effectively control the suspicious traffic to prevent cloud users' resources from becoming a malicious one.

Using Software Defined Networking (SDN) approaches to mitigate attacks in a virtual cloud networking environment has been gradually adopted by commercial companies such as VMWare [72]. SDN can be used to build the basic firewalls, VPN, and network-based intrusion framework. Since no information provided at the host level, current intrusion detection solutions may experience high false positive or false negative rates. In order to incorporate host-based malware detection, there are existing solutions [73] in the cloud system to detect malware relying primarily on anti-malware agents installed on each VM to detect and prevent malware. Such an "in-the-box" agent-based approach has two main drawbacks: (1) it has shown that anti-malware agents can become attack targets since they reside in VMs and are susceptible to various host-based vulnerabilities [74]; (2) enabling anti-malware agents in each VM can consume a lot of system resource, and thus it is not computational efficient [75] or hardware depended [76][77]. To address these issues, many existing solutions have been presented recently to place the malware detection engine outside of the VM by using VM Inspection (VMI) technology [78][79] to detect and monitor malicious processes in the VM [55]. However, existing solutions cannot address effectively on how to restrain the attacking processes. For example, manipulating the host memory data to restrain the malicious processes can usually cause unpredictable system crash.

To address the above-described issues, in this paper, we present a new defensive mechanism for cloud virtual networks, called "DIVINE" (Defense In Virtual Introspective and Networking Environment). DIVINE addresses the above presented research challenge by establishing a defense-in-depth intrusion detection and prevention framework to safeguard VMs in the cloud. DIVINE provides a systematic approach including four major security components: (1) attack vulnerability modeling and attack prediction, (2) virtual network-based intrusion detection, (3) VMI-based introspection techniques to provide host-based intrusion detection, and (4) SDN-based adaptive attack countermeasures and mitigations.

The contributions of our solutions are in three-fold: (a) *Non-intrusiveness*: the presented virtual programmable network-based defense and VMI based introspect adaptive defensive mechanism: we utilize the programmable caption approaches do not need users' intervene of running and maintaining any security detection and countermeasures be deployed at different-levels of the protocol stare functions. Moreover, the developed SDN-based countermeasures to minimize the impacts to normal traffic. (b) *Dynamic and adaptive defensive mechanism*: we utilize the programmable capabilities of virtualization and software networking to deploy countermeasures according to the severity of detected vulnerabilities and attacks. (c) *Computation and communication efficiency*: our performance evaluations demonstrate that the proposed VM Security Index (VSI) is a good metric to represent VMs' security strength, and the Return of Investment (RoI) metric provides a reasonable countermeasure selection approach to minimize the attack impacts and the cost of deploying an optimal countermeasure. The system performance evaluation results presented the system resource consumption of the deployed traffic detection and filtering solutions.

## 5.2 System and Models

DIVINE is designed to work in a cloud virtual networking environment consists of a cluster of cloud servers and their interconnections. We assume that the latest virtualization solutions are deployed on cloud servers. The virtual environment can be classified as Privilege Domains (PDs), e.g., the dom0 of XEN servers [80][81] and the host domain of KVM [67], and Unprivilege Domains (UPDs), e.g., VMs. Cloud servers are interconnected through programmable networking switches, such as physical OpenFlow Switches (OFS) [21] and Open vSwitches (OVS) [68] deployed in the PDs. Attacks on PDs and the underlying hypervisors are possible [82], however this work does not focus on their security; and it is reasonable to assume that they are free of attacks due to the difficulty and continuous improvements of virtualization security. In this work, we refer OFSs and OVSs and their controllers as to the SDN techniques. The deployed security mechanism focuses on providing a non-intrusive approach to prevent attackers from exploring vulnerable VMs and use them as a steppingstone for further attacks. We assume that attacks occur at the cloud data plane and the control plane formed by dedicated management servers through secure channels is not accessible by attackers.

In the following subsections, we present the DIVINE detection & protection model and attack model that are used in the following sections.

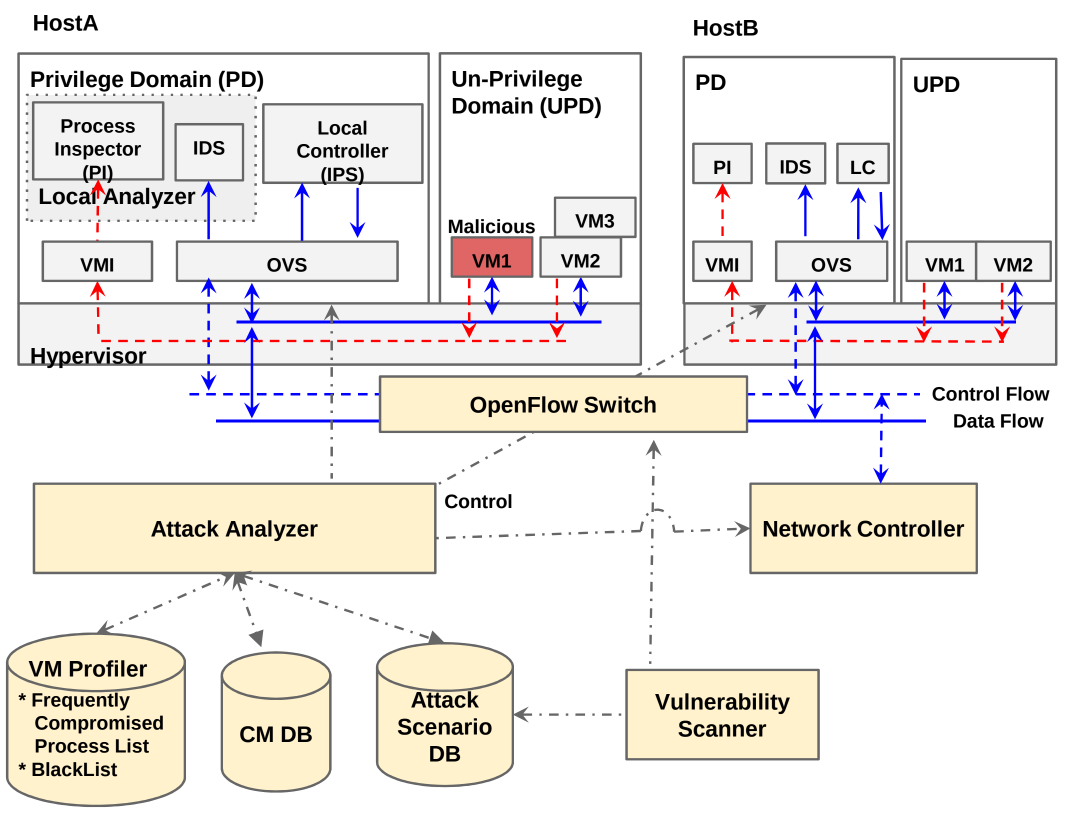


Figure 5.1: Detection and Protection Model.

### *5.2.1 Detection and Protection Model*

As shown in Figure 5.1, DIVINE security mechanism consists components designed for two levels activities: (1) *local*: on each physical cloud server; and (2) *system-level*: on a cluster of cloud servers. Each cloud server has a Local Analyzer (LA) composed by network Intrusion Detection/Prevention System (IDS/IPS) agents and a Process Inspector (PI) installed in its PD.

IDS inspects the traffic on the OVS of each VMs and submit an alert to the system Attack Analyzer (AA) when a malicious packet is detected in the virtual networks. PI can monitor processes in each VM and inspect dependency among them, and their network connections. The historical networking and processes information of each VM is maintained in a centralized VM Profiler.

The Network Controller (NC) is responsible to regulate the network traffic of the entire cloud cluster. It has a local agent on each cloud server, called Local Controller (LC) located in the PD. The LC provides IPS capability to inspect and block malicious traffic. The security policy and filtering rules are deployed in the LC instead of in the NC to improve the packet processing efficiency. Moreover, the IPS is initiated on-demand, i.e., only when vulnerable VMs are identified by the system vulnerability scanner, and then further security actions performed by the IPS, e.g., Deep Packet Inspection (DPI), traffic blocking, etc., are required. The initiation of IPS is determined by the centralized Attack Analyzer (AA) in the cloud server cluster that also maintains a countermeasure database (CM DB) and an Attack Scenario database (AS DB) that can be used for attack prediction and countermeasure selection.

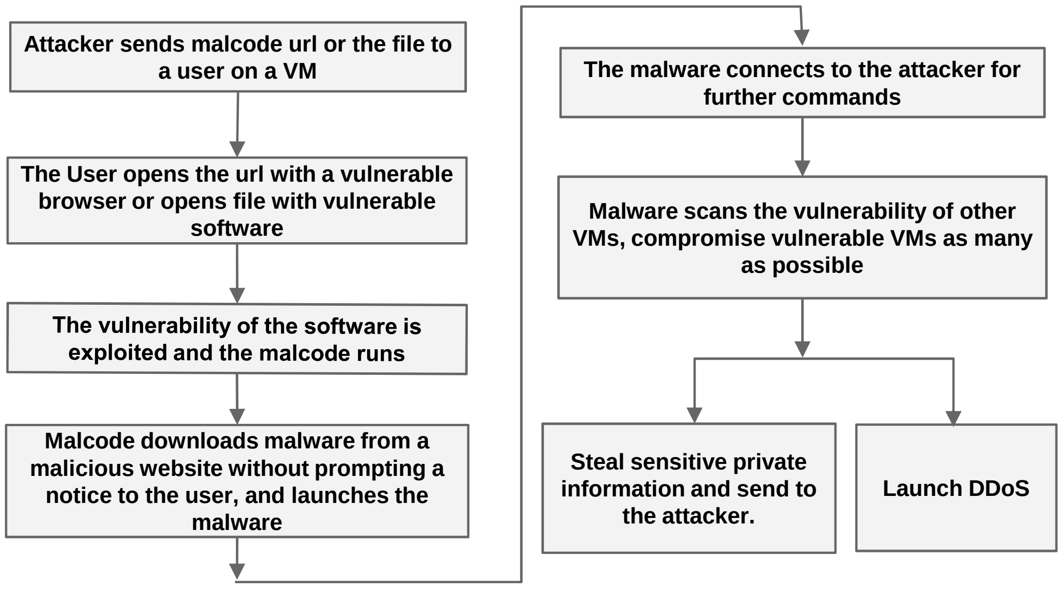
### *5.2.2 Attack Model*

Attackers can reside on either external systems or internal VMs to deploy malicious attacks. Internal attackers can have the privilege to install malware on the controlled VMs. Their goal is to explore vulnerabilities on other VMs in UPDs, and then deploy attacks to get the root privilege of them as “zoombies” or “bots”. External attackers can send malware to cloud users to persuade them to run the malware (e.g., through phishing attacks [83], cross-site scripting attacks [84], drive-by download attacks [85], etc.) or explore the vulnerabilities of the user’s VM.

After compromising a VM, the attacker can use the victim VM as a steppingstone to deploy further attacks such as exploring the topology of the local network, scanning the vulnerability of other VMs, deploying attacks based on detected vulnabilities, DDoS attacks, and so on. As a consequence, the property of cloud users are in dangerous, and the reputation of the cloud provider will be ruined.

Due to a large number of attacks are possible targeting at the vulnerabilities of VMs, we generalize the attacks through the attack graph analysis model (described in the following subsection) and use a special attack scenario to illustrate our approach in this work.

**Attack Scenario**



Attacker sends malcode url or the file to a user on a VM
The User opens the url with a vulnerable browser or opens file with vulnerable software
The vulnerability of the software is exploited and the malcode runs
Malcode downloads malware from a malicious website without prompting a notice to the user, and launches the malware
The malware connects to the attacker for further commands
Malware scans the vulnerability of other VMs, compromise vulnerable VMs as many as possible
Steal sensitive private information and send to the attacker.
Launch DDoS

Figure 5.2: Workflow of the Attack Process.

As shown in Figure 5.2, the attacker can compromises a vulnerable VM by a drive-by-download attack or a malicious email attachments at beginning. After gaining the access on this victim VM, the attacker can launch further attacks from the compromised VMs. Our research is to identify the compromised VMs and prevent them from further deploying attacks to other VMs in the cloud. The attack scenario DB maintains the information on what attack can do for the next step after compromising a VM. It describes the reachability information of the VM to other vulnerable VMs in the cloud system.

## 5.3 DIVINE Design and Implementation

In this section, we first present the system architecture of DIVINE. Then the VM and virtual network intrusion detection and prevention approaches are described. Finally, the optimal countermeasure selection strategy is presented.

### *5.3.1 System Architecture*

The proposed DIVINE framework is illustrated in Figure 5.1. It shows the framework within one cloud server cluster. The system components include *Attack Analyzer* (AA) and *Network Controller* (NC). The local components include Proccess Inspector (PI), VMI, IDS, and Local Controller (LC). In follows, we illustrate how to use these components.

**Attack Analyzer:** AA is a centralized information process center to process the security-related information and has the whole picture of the security status of the monitoring cloud server cluster. The major tasks of this component include collecting and processing information about the identified alerts, suspicious traffic and suspected processes from each PI, selecting the best countermeasure based on the knowledge of current attacks and system status, and sending the commands to the NC or an LC for countering or mitigating the attack. These functionalities are realized by four subcomponents: Attack scenario analysis model, Countermeasure Pool, countermeasure selection algorithm, and VM profiler.

Figure 5.3 shows the workflow of AA from a raised alert to a selected countermeasure. After received an alert from the IDS agent, AA matches the vulnerability in the attack scenario DB. If the alert already exists and it shows a known attack (i.e., matching the attack signature), AA will send the pre-selected countermeasure control messages to NC or an LC takes actions immediately to counter or mitigate the attack. If the alert exists in the attack scenario DB and is a known vulnerability, AA calculates the severity of the alert based on its risk probability using the base score of the exploited vulnerability from CVSS [42]. The

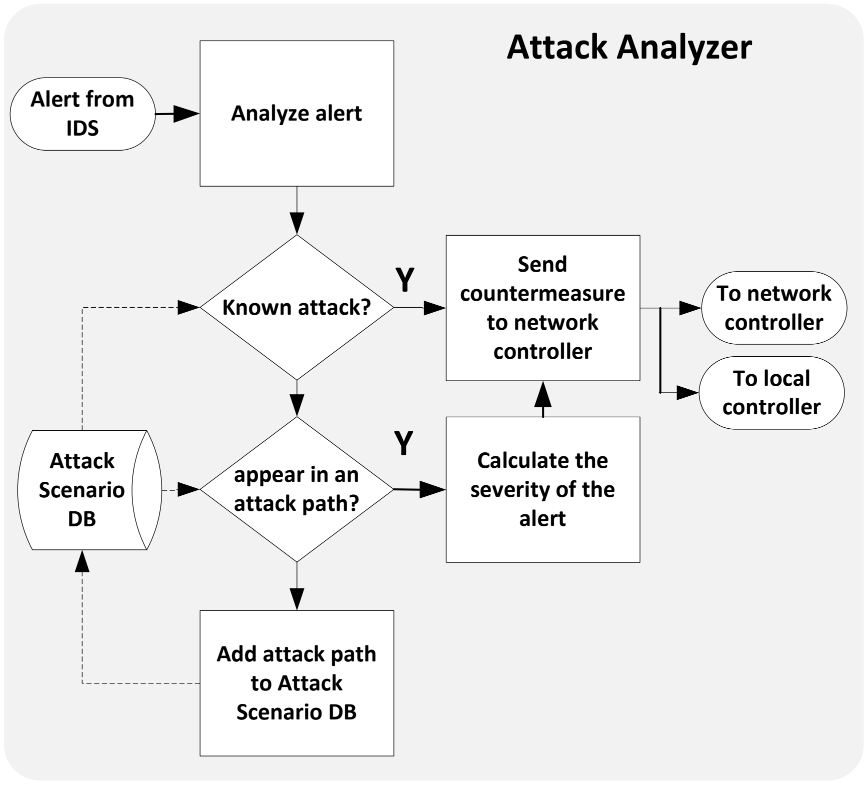

Attack Analyzer
Alert from IDS
Analyze alert
Known attack?
Y
Send countermeasure to network controller
To network controller
To local controller
Attack Scenario DB
appear in an attack path?
Y
Calculate the severity of the alert
Add attack path to Attack Scenario DB

Figure 5.3: Workflow of the Attack Analyzer.

result of the severity assessment is then sent to the countermeasure selection algorithm to select the optimal countermeasure, which will be described in details in Section 5.3.3. The selected countermeasure is then sent to the NC or an LC to reprogram corresponding OFS and/or OVS. The countermeasure strategies include reconfiguring the network topology or redirecting the suspicious flow to an LC for further traffic inspection. More strategies will be discussed in section 5.3.3.

**Attack Scenarios**: To keep track of all possible attack in the cloud, AA maintains an attack scenario analysis model to analyze vulnerabilities and their relationship from all monitored VMs. The related tasks to the attack scenario include constructing and updating the attack scenario for each VM when a vulnerability is patched or the network topology is changed, correlating alerts with attack scenarios, predicting attacks, managing the VM Profiler, and selecting the optimal countermeasure. The attack scenario DB is pre-constructed based on the following information:

- *Cloud system information* is collected from each PD. The information includes the number of VMs in each cloud server, the running services on each VM, and VM's Virtual Interfaces (VIFs) information.
- *Virtual network topology and configuration information* is collected from the NC, which includes virtual network topology, host connectivity, VM connectivity, every VM's IP address, MAC address, port information, and traffic flow information.
- *Vulnerability information* is generated by both on-demand vulnerability scanning (i.e., initiated by the NC and PI) and regular penetration testing using the well-known vulnerability databases, such as Open Source Vulnerability Database (OSVDB) [43], Common Vulnerabilities and Exposures List (CVE) [41], NIST National Vulnerability Database (NVD) [44], etc.

Many alert correlation techniques has been proposed [33][86][87][88] to reduce the false detection rate. In DIVINE, alert correlations and analysis are also handled by AA in the control center. This component has two major functions: (1) correlate alerts and integrate them into the attack scenario analysis model, (2) provide threat information or countermeasure to NC and LCs for virtual network reconfiguration or further inspection.
**VM Profiler**: VM Profiler keeps tracking the security-related status of each VM. These profiles are necessary for the AA to identity suspicious events. We use three lists to record the security status of the processes for VMs in the cloud.

- *Frequently Compromised Process List (FCP)*: FCP is a list of processes related to well-known vulnerabilities in CVE, NVD, and OSVDB because these vulnerabilities are easy to be compromised by zero-day attacks, for example, IExplorer.exe, Acrobat.exe, WinRAR.exe, WINWORD.exe and so on. FCP is a public list for all VMs.

- *Blacklist (BL)*: BL is a list of malicious processes that have been identified by a PI from a VM. The process in BL is not allowed to establish a communication channel to other VMs in the cloud.
- *Whitelist (WL)*: WL is a list of processes that have not been identified as suspicious.

**Network Controller**: NC is the main component to conduct the VM oriented (high level) countermeasure on the suspicious and malicious traffic based on the decision from AA. NC is the key component to support programmable network using OpenFlow protocol [39]. In DIVINE, each cloud server has a software switch, i.e., implemented by using Open vSwitch (OVS) [68] as the edge switch to handle all of traffic to and from VMs. The communication between cloud servers (i.e., physical servers) is handled by OFS. Both OVS and OFS are controlled by the NC, allowing the controller to set security/filtering rules on both OVS and OFS. NC is also responsible for collecting network information of current OpenFlow network, and provides inputs to AA to construct attack graphs.

### *5.3.2 Intrusion Detection and Countermeasures*

In this section, we present the intrusion detection and countermeasure strategies at the both network-level and VM-level in a non-intrusive fashion.

#### Network Level - Local Controller

LC is a traffic filter installed in the PD of each cloud server. Based on the vulnerability information (detected by the vulnerability scanner) in the attack scenario DB, and suspicious process detected by VMI, AA will inform NC about the suspicious traffic, NC then issues the commands to OVS to conduct the flow redirection to LC for further inspection. If the

LC detects any malicious patterns in the packet based on its local filtering rules, the packet will be dropped and an alert will be sent to the AA. In other words, the LC serves as the in-line mode of IPS that can be added and configured in an on-demand fashion into the existing virtual network infrastructure to guard against malicious activities. The deployment of LC is enabled by applying SDN-based approaches to provide an adaptive on-demand defense-in-depth solution.

To demonstrate how to use the LC, we developed the SDN support of the Snort inline mode [89], where the Snort IPS engine has 3 virtual interfaces (VIFs): (a) *eth0* is the VIF linked to the management network for communicating with the AA; (b) *eth1* is the VIF linked to the virtual data network for receiving suspicious traffic; and (c) *eth2* is the VIF linked to the data network for forwarding the data traffic passed the examination. To reduce system resource consumption, we devise a fine-grain access control module at the OVS level to regulate the traffic among all VIFs. The in-line mode IPS agent is configured as an IP forwarding middle box, and both *eth1* and *eth2* do not assign any IP addresses. The Ethernet packets that are redirected, inspected, and forwarded without being modified to make the IPS is unperceivable to the packet sender and receiver. This feature guarantees that the mechanism of LC is non-intrusive.

### Host level - Local Analyzer

The LA including the security components that installed on the PD of each cloud server including the VMI, PI, and IDS agent. The LA is designed to monitor and analyze both network traffic and internal processes of all VMs on the cloud server.

The major tasks of LA include 2 types of inspections: (a) traffic inspection, and (b) process inspection. The traffic inspection keeps eyes on the traffic within the host, while the process inspection monitors the running processes in each VM. Each packet in a flow is monitored and detected by the IDS agent. If any potential malicious activity is detected, an alert is then raised by the IDS agent and sent to AA for further process.

Process inspection is able to identity the source process of each TCP/UDP connection by hooking up the network traffic with the socket status of the corresponding processes. This process information is sent to AA with the related traffic information for further analysis. This approach allows the AA to link the network traffic to the internal processes in a VM to identify the consequence of a detected malicious packet.

PI is the key component for process inspection. It identifies running processes in each VM by traversing the data structures of all of running threads through the VMI module. The PI gets the RAW data from memory of each VM through the hypervisor. We solve the *semantic gap* problem of these RAW data by reconstructing the related data structures of VM OS (this is very challenging when using Windows guest OS due to the closed-source OS, compared to the work on Linux guest OS[90][62]) to reveal the useful information about the processes and network connections in each VM. This information is essential for AA to know which process initiates the network traffic and then to make the right decision about the traffic. The source process initiates a TCP/UDP connection can also be identified by enumerating the network connections and sockets maintained by a VM. PI also inspects the dependency of processes in each VM and discovers the family tree of discovered processes to track the malicious activities. Detecting the hidden malicious process is a very important task for securing the system.

For the malicious process detection, traditional methods [73] primarily rely on the detection and monitoring agent installed in the protected VM, in which special modifications of the guest OS are required. However, such solutions have drawbacks in that the system integrity is not preserved, detection and monitor agent is easy to be attacked in the guest OS. To address these issues, we place the detection and monitor agent in the PD, which is outside of a VM.

In the presented solution, we developed five submodules in the VM process inspection module: security console, daemon, VMI, semantic reconstruction, extractor and executor. The security console is an interactive console for administrators to setup the VM to be monitored and configure the monitor strategies and rules. These configuration messages will be sent to the daemon module with a command and displayed on the console to notify the administrators or serve as a log. VMI module is responsible for mapping the addresses between the application process' virtual address space in the VM and the actual physical address space through the two-layer mappings.

The first layer translates the Guest Virtual Address (GVA) to the Guest Physical Address (GPA). The second layer translates the GPA to the Host Physical Address (HPA). By reading the GVA from the outside of VM, VMI is able to read the content of the memory information in the VM through the translation from VM's GVA to VM's HPA.

In the virtualization technology, such as the virtual machine monitor (VMM) or hypervisor can only read VM's internal virtual hardware information, which is a large volume of unreadable raw data. It lacks of meaningful semantic view of internal structures in the guest operating system and also hard to directly map to the current running status in the system, in which this situation is often referred as "sematic gap" problem. The semantic translation module is the one for addressing the sematic gap issue. It is responsible for reconstructing the kernel-level process of a VM and recovering the VM's virtual hardware information from the information captured by the VMI.

For example, in order to retrieve the socket and network connection information of processes in a MS Windows VM, the PI locates the _ADDRESS_OBJECT and _TCPT_OBJECT data structures in VM's memory, which is pointed by _AddrObjTable and _TCBTable in PE (Portable and Executable) header of the kernel module tcpip.sys. The PI traverses the linked list of _ADDRESS_OBJECT and _TCPT_OBJECT data structures to identify all the in-use sockets and active network connections, and the processes they belong to.

### 5.3.3 Countermeasure Selection

In DIVINE, countermeasure selection is based on the attributes of the countermeasures and the security status for the victim VM. We define *VM Security Index (VSI)* in section 5.3.3 to represent the security and health status for each VM and define the improvement (i.e., reduction) of the VSI value after applying a countermeasure as the *benefit* of the countermeasure. For the countermeasure, we consider four attributes for each countermeasure: (1) *Cost* is the unit that describes the expenses required to apply the countermeasure in terms of resources and operational complexity, and it is defined in a range from 1 to 5, where 5 means the highest cost. (2) *Intrusiveness* is the negative effect that a countermeasure brings to the Service Level Agreement (SLA) and its value ranges from 1 (the least intrusive) to 5 (the most intrusive), also with value 0 means that the countermeasure has no impacts on the SLA. (3) *Condition* is the requirement for the corresponding countermeasure. (4) *Effectiveness* is the percentage of probability changes of the node, for which this countermeasure is applied.

#### Countermeasures DB

We consider the countermeasure strategies at both network level and host level. The network level countermeasures (represented as a set *NLC*) primarily rely on network reconfiguration strategies through SDN. *NLC* can contain countermeasures such as traffic isolation, deep packet inspection (DPI), MAC address rewrite, IP address rewrite, traffic drop, block the source port, network topology change, and so on.

The network reconfiguration strategies mainly involve two levels of action: layer-2 and layer-3. At layer-2, virtual bridges are main components in cloud's virtual networking system to connect two VMs directly. A virtual bridge attaches multiple VIFs. VMs on different bridges are isolated at layer 2. Based on this layer-2 isolation, we can deploy layer-2 network reconfiguration to isolate suspicious VMs. For example, vulnerabilities due to ARPspoofing [48] attacks are not possible when the suspicious VM is isolated to a different bridge. Layer-3 reconfiguration is another way to isolate/mitigate an attack. Through the NC, the flow table on each OVS or OFS can be modified to change the network topology.

As for the host-level countermeasure (represented as $HLC$) strategies, our system mainly involves two levels of actions: VM-level network reconfiguration and process-level network reconfiguration. The VM-level reconfiguration strategy can either disable the suspicious VM's VIF to isolate it from other VMs or force the suspicious traffic to the LC for further deep-packet inspection.

Process-level network reconfiguration provides a fine-grained and scrutinized control over the network connections. With the processes and sockets information of a VM reported from an LA, the NC is able to make OVS to isolate the traffic issued by the inspected processes or to redirect the traffic to an LC for further inspection before delivering to the destination. The advantage of the process-level network reconfiguration is that all other network services remain untouched while the activity of the suspicious process is monitored or isolated.

In summary, the host level countermeasure ($HLC$) strategies of DIVICE solutions include: (1) *VMIsp*: VM-level Inspection put a suspected VM under the inspection of a in-line mode intrusion prevention system (IPS). (2) *VMIso*: VM-level Isolation disables all network traffic to and from a suspected VM. (3) *PrIsp*: Process-level Inspection redirects the traffic of a suspicious process in the VM to a in-line mode IPS. (4) *PrIso*: Process-level Isolation prevents the suspicious process in a suspected VM from communicating with other VMs while the network services from other processes stay unaffected.

**VM Security Index**

In our framework, we define a VM Security Index (VSI) for each VM to represent the security level and health status of the VM in the current virtual network environment. We refer to the VEA-bility metric in [51] and define our VSI with three different parameters which include Vulnerability ($V$), Exploitability ($E$), and Health status of processes ($H$) for a VM. The VSI for a VM $k$ as shown in (5.1). The value is ranged from 0 to 10, where lower value means better security.

$$VSI_k = \alpha \times V_k + \beta \times E_k + \gamma \times H_k, \tag{5.1}$$

where $\alpha$, $\beta$ and $\gamma$ are weights for each parameter respectively with $\alpha + \beta + \gamma = 1$ and $\alpha, \beta, \gamma \geq 0$.

$V_k$ is the vulnerability score for VM $k$. The score is the exponential average of the base score from each vulnerability of the VM with the maximum value of 10, as shown in (5.2). Basically, vulnerability score considers the base scores of all the vulnerabilities on a VM. A base score is assigned by CVSS [42] and it depicts how easy it is for an attacker to exploit the vulnerability and how much damage it may incur.

$$V_k = \min\{10, \ln \sum e^{BaseScore(v)}\}. \tag{5.2}$$

$E_k$ is a exploitability score for VM $k$. It takes the exponential average of exploitability score with the maximum value of 10 as shown in (5.3). $S_k$ represents the number of services provided by VM $k$. $NS_k$ represents the number of network services the VM $k$ can connect to.

$$E_k = \min\{10, \ln \sum e^{ES(v) \times \frac{S_k}{S_k + NS_k} + ES(v') \times \frac{NS_k}{S_k + NS_k}}\}, \tag{5.3}$$

where $ES(v)$ represents the exploitability score of vulnerability $v$ in VM $k$, $ES(v')$ represents the exploitability score of vulnerability $v'$ in the VMs which the VM $k$ is able to connect to.

The exponential addition of base scores allows the vulnerability score to incline towards higher base score values and increases in logarithm-scale based on the number of vulnerabilities. The exploitability score on the other hand shows the accessibility of a target VM, and depends on the ratio of the number of services to the number of network services as defined in [51]. Higher exploitability score means that there are many possible paths for the attacker to reach the target.

In the following presentation, we use Windows VM as the illustration example to present how the presented process inspection works. $H_k$ is the health status of processes in VM $k$. Higher value of $H_k$ means more hidden process in action and more read operation in VM $k$. This value is determined by the difference between two process lists, where the list from process view ($pl_p$) and the list from the thread view ($pl_t$). If $pl_t(k) = pl_p(k)$, which means no hidden process exists, we set $H_k = 0$. In the case of $pl_t(k) \neq pl_p(k)$, which means some hidden processes exist, we set $H_k$ to equation (5.4).

$$H_k = (1 - \prod_{p \in pl_t(k) - pl_p(k)} (1 - \frac{cpu(k,p,t)}{t}) \times e^{-read(k,p)}) \times 10, \tag{5.4}$$

where

- $pl_p(k)$ is a process linked list retrieved from the EPROCESS chain (i.e., a process chain in Windows) in the system of VM $k$. Each item in the list represents a process with its process ID (PID) and process name (pname). The attacker can hide the malicious process from this list.

- $pl_t(k)$ is a process linked list retrieved from the process information in each thread in the system of VM $k$. Each item in the list also represents a process with its process ID (PID) and process name (pname). Every process can be collected by checking the information in the threads; even the hidden process.

- $read(k,p)$ is a value representing the count of reading operations performed by the process $p$ in VM $k$.
- $cpu(k,p,t)$ is the elapsed time of process $p$ in action in the system of VM $k$ during the measurement time period of $t$ seconds.

VSI can be used to measure the security level of each VM in the virtual network in the cloud system. It can be also used as an indicator to demonstrate the security status of a VM, i.e., a VM with higher value of VSI means is easier to be attacked. In order to prevent attackers from exploiting other vulnerable VMs, the VMs with higher VSI values need to be monitored closely by the system (e.g., using DPI) and mitigation strategies may be needed to reduce the VSI value when necessary.

**Countermeasure Selection Algorithm**

In DIVINE, the countermeasure selection algorithm is based on the cost-benefit model. Every countermeasure strategy in the countermeasure pool is assigned by an estimated cost beforehand. For the *benefit* metric, the algorithm needs to predict the reduction of VSI after applying the countermeasure. We define that the countermeasure with the highest value of the Return of Investment (ROI) is the best countermeasure to counter the attack event. The ROI of a countermeasure *cm* applying on VM *k* is defined in (5.5).

$$ROI_{cm}^{k} = \frac{benefit(k,cm)}{cm.cost + cm.intrusiveness}, \tag{5.5}$$

where $benefit(k,cm)$ is the VSI reduction for VM $k$ after applying the *cm*, *cm.cost* is the cost to apply the *cm*, and $cm.intrusiveness$ means the intrusive level of the *cm*. The optimal countermeasure selection for VM $k$ can be derived by simply running the following linear optimization function:

$$\max_{\forall cm} ROI_{cm}^{k}, \tag{5.6}$$

where $cm \in NLC \cup HLC$.

For the suspicious traffic from a suspected process or VM, Process Inspection and VM Inspection (i.e., *PrIsp* and *VMIsp*, respectively) are the recommended countermeasures because all of known attack packets will be detected by the inline-IPS while the normal traffic remains unaffected. As for the traffic from an identified malicious process or VM, Process Isolation and VM Isolation (i.e., *PrIso* and *VMIso*, respectively) are recommended countermeasures because it can eliminate the possibility of further attacks exploring the same vulnerability. This strategy also saves the resource for detecting the traffic in the future.

For a single network attack that may impact the behavior of a suspected vulnerable process, Process Inspection and Isolation (i.e., *PrIsp* and *PrIso*, respectively) are sufficient to handle this situation while all other services in the VM are unaffected. For a comprehensive attack that may turn the VM into an attack agent and involve multiple processes on the compromised VM, VM Inspection and Isolation (i.e., *VMIsp* and *VMIso*, respectively) are needed because all the processes are vulnerable to this attack.

## 5.4 Performance Evaluation

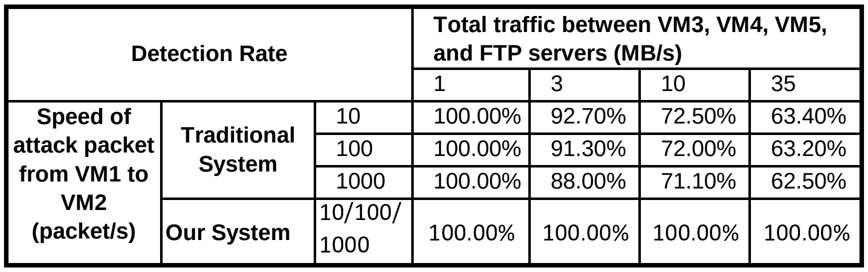
Our evaluation includes two aspects: security assessment and system performance evaluation. We consider the system performance from system computing resource consumption and networking overhead due to the operation of major components in our framework. For the security performance, we design a case study to evaluate the detection rate and security operation overhead.

### *5.4.1 Security Analysis*

We established a demonstrative cloud environment to study the security performance of DIVINE. The exploit website is played by Metasploit with CVE-2012-0158 exploit script. A customized MS08-067 attack tool is ready for download in the malware website. We assume that the malicious websites are prepared by the attacker. In the emulated environment, we setup two ftp servers with some large files to be downloaded by the VMs inside the cloud system. The max download speed limitation can be set on each ftp servers.

There are 14 VMs in the cloud system. All of them have the vulnerabilities mentioned above; say they are vulnerable to the attacks in this test. VM1 is the first VM that is compromised by the drive-by-download attack [85]. And after being compromised, it tries to attack VM2. Our proposed system is to detect the attack issued from VM1 and block it.

Here we test the security performance in 2 scenarios: 1) the traditional system: there is an IDS in the cloud system to detect the attack from VM1; 2) DIVINE approach: based on our design, the traffic of the malware process is inspected by the IPS while other traffic is inspected by the IDS. Then, we record the detection rate reported in both scenarios under the various ftp traffic speeds and attack speeds.



| Detection Rate | | | Total traffic between VM3, VM4, VM5, and FTP servers (MB/s) | | | |
|---|---|---|---|---|---|---|
| | | | 1 | 3 | 10 | 35 |
| Speed of attack packet from VM1 to VM2 (packet/s) | Traditional System | 10 | 100.00% | 92.70% | 72.50% | 63.40% |
| | | 100 | 100.00% | 91.30% | 72.00% | 63.20% |
| | | 1000 | 100.00% | 88.00% | 71.10% | 62.50% |
| | Our System | 10/100/1000 | 100.00% | 100.00% | 100.00% | 100.00% |

Figure 5.4: Comparison of Detection Rate.

Upon the testing environment we setup, we use the detection rate as the metric to compare our system with normal security system. From the result shown in Figure 5.4, in normal detection system, the detection engines can handle all the traffic generated by attacker at different rate when network has 1MB/s application traffic. When the traffic on network increases to 3MB/s, the performance is compatible with the attacking rate. The detection rate drops to about 72% and about 63% when the total traffic load raise to 10MB/s and 35MB/s, respectively. In contrast, the detection rate keeps 100% all the time in our established environment because our system identifies the traffic sent out from the malware process as suspicious and redirects them to IPS for further inspection. Thus, our proposed solution has better security performance in terms of the successful detection rate.

Figure 5.5 shows the plotting of *VSI* for these virtual machines before countermeasure selection and application. The IP addresses in the figure are VMs' private IP addresses, and letter "L" represent Linux VM and letter "W" represents Windows VM.

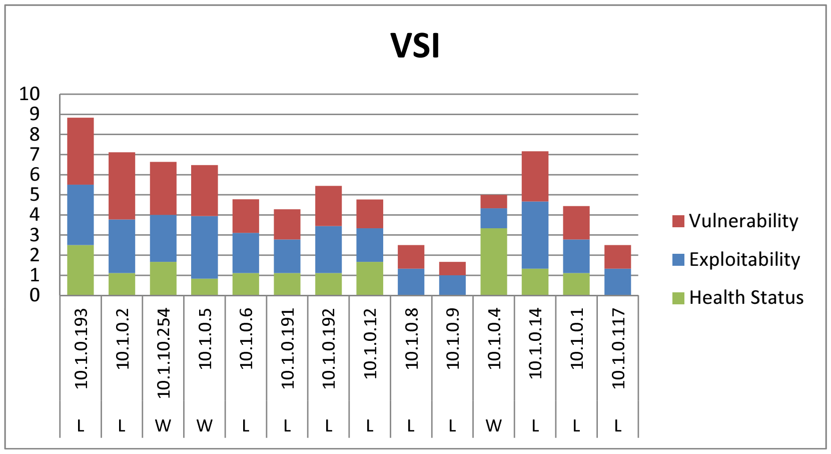


Figure 5.5: VM Security Index.

### *5.4.2 System Performance*

The system performance evaluations in this section are all performed on the Dell PowerEdage R510 servers with Intel quad-core Xeon 2.4 GHz CPU and 32G memory. Each server is installed Citrix XenServer 6.0 as a cloud server.

In DIVINE, there are three major components in each physical cloud server: LC, LA and VMI module. They are the main resource consumption sources of the system. Thus, the system performance study only evaluates the performance on one physical server.

In Figure 5.6, we use packet generator to mimic real traffic in the cloud system. The traffic load, in form of packet sending speed, increases from 1 to 5000 packets per second. To guarantee the fairness of this evaluation, we set the number of virtual CPU to 1 for LC and LA. LA consumes less CPU while LA takes more CPU resource. Figure shows that the packet rate reaches the 3000 packet/s, the CPU utilization of LC reach to around 52%, while LA only occupies about 40%. The LA includes an IDS agent and a PI. The results demonstrate that IPS is relatively heavier than IDS and the process inspection operation is lightweight.

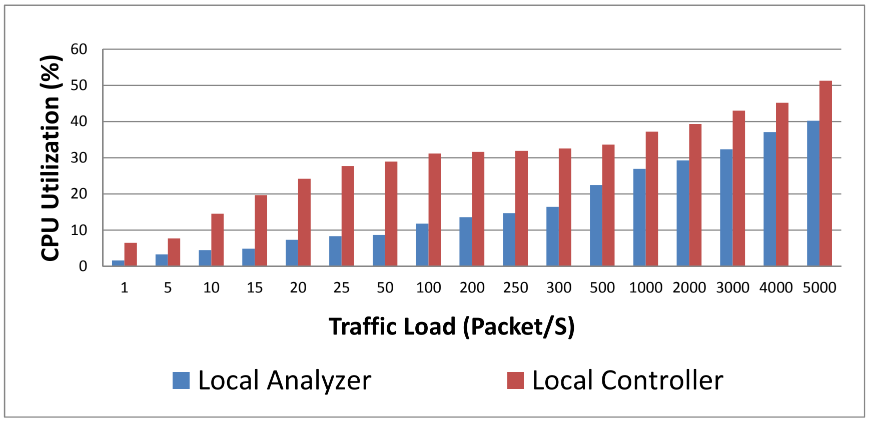


Figure 5.6: CPU Utilization of LA and LC.

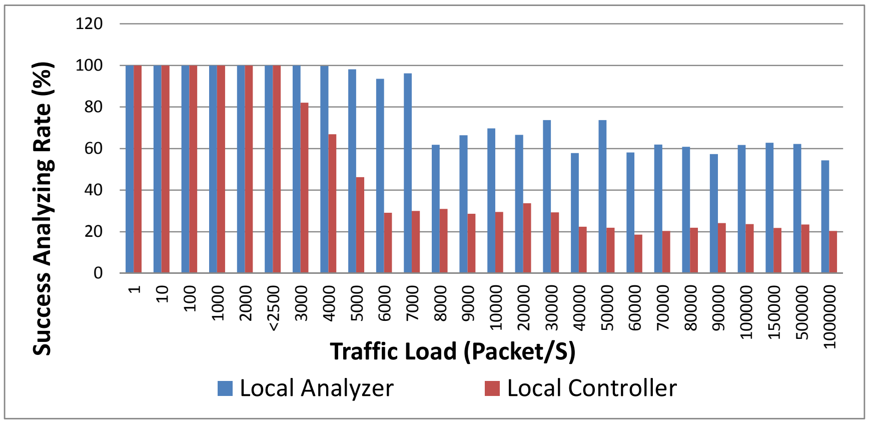


Figure 5.7: Successful Analyzing Rates of LA and LC.

Figure 5.7 shows the performance of LA in terms of percentage of successfully analyzed packets, which is defined as success analyzing rate, i.e., the value of the number of the analyzed packets divided by the number of all received packets. The higher value means more packets can be handled by LA. In this Figure, we start the traffic load from 2500 packets per second since when the packet sending rate is lower than 2500 packets per second, both LA and LC achieves the 100% successful analyzing rate. From the result, the LC demonstrates 100% performance because every packet captured by LC is cached in the detection queue. However, 100% success analyzing rate of LC is at the cost of the analyzing delay. As for LA, the detection agent does not queue the captured packets and thus no delay is introduced. However they are all experiencing packet drop when traffic load is large. This results demonstrate that IDS is preferred and IPS is used according to the system's security situation is a good strategy to conserve the system resource.

## 5.5 Related Work

In [72], IDS solutions are envisioned to utilize SDN [38] approach to detect and mitigate network-based intrusions, where networking functions can be programmed through software switch and OpenFlow protocol [21]. FRESCO [91] presented an secure OpenFlow application platform. Flow-based switches, such as OVS [68] and OpenFlow Switch (OFS) [21], support fine-grained and flow-level control for packet switching [39]. With the help of the central controller, all OpenFlow-based switches can be monitored and configured.

Many existing solutions have been presented recently to place the malware detection engine outside of the VM by using VM Inspection (VMI) technology [79][78][92]. For the VMM based malware detection, Livewire [54] proposed first concept of placing an"out-of-VM" monitor and applying VMI technology to reconstruct the semantic view of the internal structure of the VM. VM-watcher [55] is an "out-of-box" approach that overcomes the semantic gap created by the missing information about detailed internal view of the system by "in-the-box" approaches. To close the semantic gap, it applies a new technique called 'guest view casting' to reconstruct internal semantic views (e.g., files, processes, and kernel modules) of a VM non-intrusively from the outside. Antfarm [59] tracks processes in a VM by tracking address space in the VMM. Lycosid [60] is a VMM-based hidden process detection and identification service and uses cross-view validation principle to detect maliciously hidden OS processes. Maitland [61] introduced a light-weight introspection technique in absence of hypervisor, that provides the same levels of within-VM observability. To address isolation challenges in out-of-VM approaches, Out-Grafting [62] was proposed. It allows fine-grained process-level execution monitoring.

## 5.6 Conclusion and Future Work

In this paper, we present DIVINE that is a multi-level intrusion detection approach incorporating both network and host-based intrusion detection techniques. In order to minimize the intrusion to cloud services, DIVINE provides a countermeasure selection approach by using a comprehensive evaluation strategy by evaluating a set of metrics considering cost, efficiency, and security improvements. Using the SDN approach, we can greatly improve the agility of the defensive mechanism and provide an adaptive defense-in-depth approach. Moreover, the host-based intrusion detection is based on VM introspection techniques that do not need the implement special codes in users' VMs. Our evaluation results shows that the presented solution can greatly improve the security of the virtual cloud networking system and minimize the intrusiveness to cloud users.

The presented solution only considers one cloud server cluster with one centralized attack analyzer and one network controller. In the next step, we will investigate in multi-cluster environment involves multiple attack analyzers and network controllers and study the scalability issues in a large cloud networking environment.

# Chapter 6

# SCALABLE SDN-BASED NETWORK RECONFIGURATION MODEL CONSIDERING SECURITY AND QOS CONSTRAINTS

## 6.1 Introduction

Cloud computing has become a new computing paradigm for hosting and delivering services over the Internet in the past few years. More and more small and medium business companies have started the cloud to obtain fast access to the best business applications and better boost the infrastructure resources. However, recent studies have shown that cloud security concern hinder customers' migration to the cloud. The reason for the customer's hesitation or rejection due to the virtualization, which a key technology of cloud computing. Virtualization introduces attack surfaces for the cloud, such as hypervisor vulnerabilities [93], VM escape [94], VM hijacking [95], VM Sprawl [96], and breaking the isolation of VM and virtual network [97][98].

In the traditional data centers, where system administrators have full control over the host machines, vulnerabilities can be detected and patched by the system administrator in a centralized manner. However, patching known security holes in cloud data centers, where cloud users usually have the privilege to control software installed on their managed VMs, may not work effectively and can violate the Service Level Agreement (SLA). Furthermore, cloud users can install vulnerable software or even the malicious and suspect code on their VMs, which essentially contributes to loopholes in cloud security. The challenge is to establish an effective detection and response system for accurately identifying vulnerabilities and attacks and minimizing the impact of security breaches to cloud users.

Fortunately, with the recent advent of Software-Defined Networking (SDN), cloud provider is able to efficiently route suspicious and malicious flows from cloud users in their virtual networks [99] and conveniently build security applications in their virtual networks [91] to detect and response the malicious traffic. However, the dynamic routing mechanism performed by the cloud provider shouldn't impact the SLA for cloud users. This paper discusses how SDN can be applied in the context of cloud computing to dynamically change the underlying networking infrastructure in order to maintain the security needs while balancing the required QoS for cloud users.

In this article, we propose a scalable framework to track the vulnerabilities in each VM and the vulnerability dependency among VMs in the virtual network using the attack graph analytical model. An attack graph is a modeling tool to illustrate all possible multi-stage, multi-host attack paths that are crucial to understand threats and then to decide appropriate countermeasures [40]. However, the well-known state explosion problem of attack graph [100] makes it unable to model attack scenarios in a large scale network, such as the network in a large data center. The framework we proposed is able to confine the size of attack graph in a manageable scale using SDN dynamic network reconfiguration features while maintaining the minimum correlation among attack graphs.

In this framework, we leverage the programmable network feature from SDN architecture to isolate, quarantine, and inspect the traffic sending from vulnerable services. We attempt to achieve this goal by using moving target defense techniques that force attackers to continuously chase their target, deterring, and eliminating attacks without interrupting regular network traffic. This will eliminate the attacker's time and space advantage and create agility against advanced persistent threats [101].

The contributions of our framework are presented as follows:

- We create a real-time network reconfiguration model for a SDN-based virtual network to respond any major abnormal events in the network.

- The model we proposed is not only passively response the network events, but also proactively change the network configuration using moving target defense techniques.
- The reconfiguration strategies in our model make each user's virtual network environment more secure while keeping their network functionality same as their SLA requirement.
- Our framework is able to partition the network into several logically isolated zones with different security properties by classify vulnerabilities in the network and minimize the vulnerability dependency between zones.
- Our framework applies novel distributed attack graph analytical model to track the vulnerabilities in each zone by confining the connectivity of vulnerabilities between zones.

## 6.2 Problem Statement

Attack graph and attack tree face different scalability issues in different phases: generation, representation, evaluation, and modification [102]. The network size is keeping increasing (e.g., growing number of mobile devices), but we still lack of solutions to deal with the scalability of security evaluation. Current attack graph and attack tree representations still suffer from scalability problem. Attack graph can be generated in polynomial time complexity, but evolution has a state space explosion problem to cover all set of attack paths. Attack tree can be evaluated in a scalable manner only if it can be generated efficiently, but it still lacks of an efficient attack tree generation method. Therefore, we still need a more robust method of graph-based evaluation and tree-based generation methods.

Using attack graph to model attackers' possible behaviors and vulnerabilities in a network have been well-studied, e.g., [29][63][32][65]. However, there is no previous work that uses attack graphs in a highly dynamic virtualized and reconfigurable networking environment, and considers applying the optimal countermeasure in a dynamic fashion. Moreover, the real-time and scalability issues become significant in such a dynamic environment. In a cloud data center environment, a large number of VMs can be hosted in the cloud. Since an attack graph maintains the correlations among vulnerabilities, with the number of identified vulnerable VMs, the complexity of an attack group increases exponentially. To address this problem, we propose a network reconfiguration model to reduce the scope of the attack graph by partitioning the monitored network into secured zones. Our clustering approach focus on one or a combination of the following clustering approaches directions:

- Grouping VMs with similar vulnerabilities (see figure 6.1 (a))
- Discover groups of connected VMs with the least network and application connections to other groups (see figure 6.1 (b)).

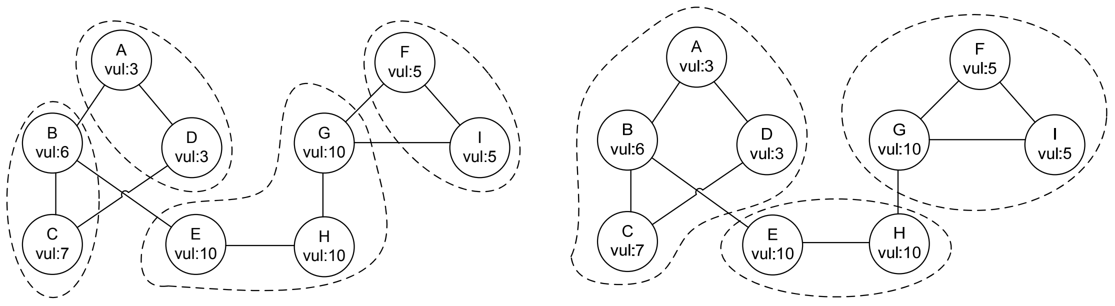


Figure 6.1: Grouping VMs with (a) Vulnerabilities or (b) Connectivity of Each Node.

We propose a comprehensive solution to address the network and the attack graph clustering problem. In order to confine the size of attack graph and reduce the vulnerability dependency between VMs, the VM grouping approach is applied. VM grouping is based on the vulnerability type and severity in each VM. VMs with similar vulnerability or the same level of severity are considered as a security group. After grouping based on the vulnerabilities, we can have a clearer view of the system's security situation and know which vulnerable applications are affecting the system's security. After grouping, an aggregated vulnerability metric, can be assigned to each vulnerability group. Such vulnerability metrics, which the group VM security index based on the CVSS score, are important in quantitative assessment of a system's security. It is also an important indicator for applying the reconfiguration to mitigate the potential risk and reduce the potential impact to the network and other VMs. We calculate the vulnerability score with a base score from the CVSS system [42] for each VM using exponential average to measure all vulnerabilities for each VM (VM Security Index), so that the score will be at least equal to the most serious vulnerability in the systems. For the same security group, we also take the exponential average on the vulnerability score from all of the VMs in the same group (group VM Security index) and assign it an aggregated metric to measure the security level of the group.

Besides the VM grouping based on the security, we also partition the network according to the connectivity and reachability of VMs. The size of an attack graph is not only affected by the vulnerabilities in the system, but also the network connectivity of each VM. If we can group VMs with the strongly connected graph and partition the network according to the least connection of the cluster, we can then confine the size of the attack graph and also minimize the vulnerability dependency among attack graphs. The problem of graph partitioning consists of dividing the vertices in a number of groups of predefined size, such that the number of edges lying between the groups is minimal. The number of edges running between clusters is called cut size [103]. Most variants of the graph partitioning problem are NP-hard. Network clustering is a fundamental task in many fields of science and engineering. These methods tend to cluster networks such that there are a dense set of edges within every cluster and few edges between clusters [104]. Many algorithms have been proposed from practitioners in different disciplines including computer science and physics. Successful examples are Min-Max Cut [105] and Normalized Cut [106], as well as Modularity-based algorithms [107][108][109].

In our framework, we take multi-objective optimization approach to obtain the optimal attack graph clusters by considering both security factors of VM groups and the reachability among groups. The similar vulnerability of a group is able to reduce the redundancy nodes and paths in the attack graph so that the size of attack graph can be reduced by removing the redundancy. The least connection between groups is able to reduce the dependency in both reachability and vulnerability relation so that the attack graph cluster is easier to be managed and adjusted based on different security concerns.

## 6.3 Threat, Networking and Attack Graph Models

To better describe the problem and the proposed solution, we define several models in our proposed framework in this section.

### *6.3.1 Threat Model*

In this work, we assume the hypervisor is trusted and secured, which means the hypervisor properly isolates the resources and environment for the running VMs. The hypervisor is protected against any exploits launched by the attacker and the detection engine installed in the hypervisor is invisible to the attacker. We also assume users are able to install vulnerable software and execute any malware or malicious code in their VM. System administrator cannot patch the software or remove the malicious code without user agreement. However, CSP allows to block the traffic issued by such processes.

### *6.3.2 Networking Model*

In order to partition the network to obtain a manageable size of the network to be monitored, we create a networking model for a virtual network in a cloud system. The networking model is not only able to confine the size of attack graphs but also able to provide an assessment model for the reconfiguration strategies.

We model the network as a directed graph $G = (V, E)$, where $V$ is the set of hosts, $E$ is the set of links. Suppose there is a flow with source $S$ and destination $D$ $(S, D \in V)$, and the duration of the flow can be divided into multiple intervals. The goal of the network reconfiguration is to find a route between $S$ and $D$ that satisfies the following constraints for every interval of the flow duration:

- *Capacity constraint*: the new route should not include those nodes or links that are already overloaded or those nodes or links that do not have the bandwidth requirement for the flow.
- *QoS constraint*: the mutated routes should maintain the required quality, such as bounded delays or number of hops.
- *Security constraint*: the security index of the new selected route should be the optimal or a sub-optimal path.
- *Cost constraint*: the reconfiguration cost of the new route should be the least if there are multiple routes exist.

### 6.3.3 Attack Graph Model

An attack graph is a modeling paradigm to illustrate all possible multi-stage, multi-host attack paths that are crucial to understand threats and then to decide appropriate countermeasures [40]. In an attack graph, each node represents either precondition or consequence of an exploit. Attack graph is helpful in identifying potential threats, possible attacks and known vulnerabilities in a cloud system. Since the attack graph provides details of all known vulnerabilities in the system and the connectivity information, we get a whole picture of the current security situation of the system where we can predict the possible threats and attacks by correlating detected events or activities.

An attack graph consists of vertices representing the conditions and actions. Conditions are the system configuration or access privileges that should be true in order to exploit any vulnerability. Actions are the steps that attacker performs in order to compromise VMs. The actions of attacker depend upon existence of one or more conditions. The edges in the attack graph connect nodes, and a set of edges which constitute of a path from the initial node to the final node denotes an attack path.

We define an attack graph a directed graph $AG = (AV, AE)$ where $AV = N_C \cup N_D \cup N_R$ and $AE = E_{pre} \cup E_{post}$. $AV$ is the set of vertices that include three types of nodes: conjunction node $N_C$ to represent exploit, disjunction node $N_D$ to denote result of exploit, and root node $N_R$ for showing initial step of an attack scenario. $AE$ denotes the set of directed edges. An edge $e \in E_{pre} \subseteq N_D \times N_C$ represents that $N_D$ must be satisfied to achieve $N_C$. An edge $e \in E_{post} \subseteq N_C \times N_D$ means that the consequence shown by $N_D$ can be obtained if $N_C$ is satisfied.

We apply the risk probability measurement on each attack graph based on the Bayesian network probability model to calculate the conditional risk probability for each vulnerability node and the cumulative risk probability for the target node in an attack graph. The risk probability is based on the CVSS score of the vulnerability found in the monitored network and VMs. We assign each attack graph an aggregate potential risk probability by averaging the cumulative risk probability of all target nodes in a monitored network. The target nodes can be randomly selected from the network or selected from VMs which including important assets.

## 6.4 System Design

In this section, we first present the system architecture of our design and then detailed descriptions of its components.

### 6.4.1 Overall Architecture

The proposed framework is illustrated in figure 6.2. There are four stages in the framework: attack graph construction, attack graph clustering, security evaluation, and reconfiguration.

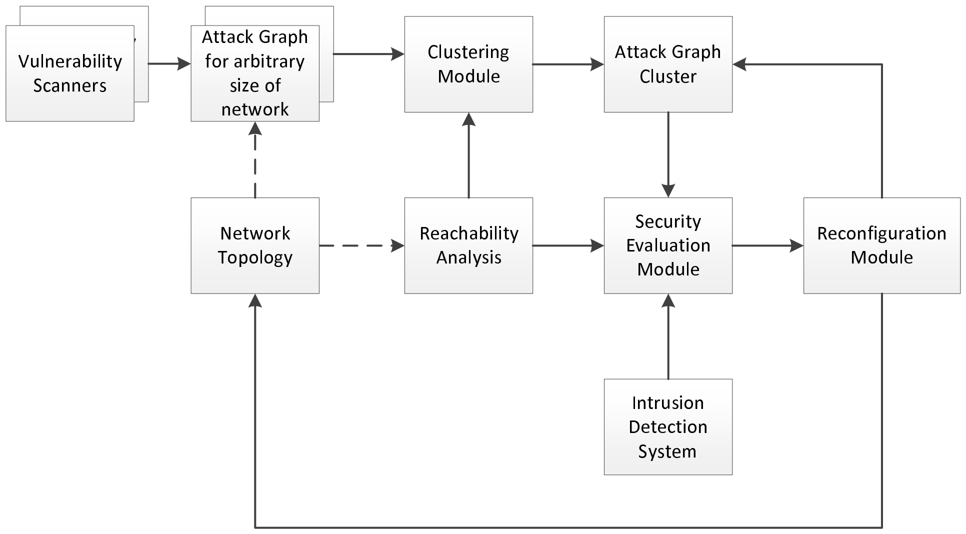


Figure 6.2: Reconfiguration Model.

Attack graph construction stage is responsible for creating attack grapsh for an arbitrary size of the monitored network. Attack graph clustering stage is responsible for adjusting each attack graph to be a manageable size by applying SDN-based dynamic reconfiguration approaches while maintaining the original network services in each host. Our clustering approach is able to reduce the attack graph size and the vulnerability dependency among attack graphs by reconfiguring the network service in a vulnerable host, reroute the network path to a more secure traffic path, or reconfiguring the network topology through a SDN architecture. The attack graph clustering requires network topology and network reachability information to adjust the size of the monitored network. The security related index from the security evaluation module is also important triggers to activate the adjustment of attack graphs and monitored network.

The security evaluation module provides the security related metrics for the attack graph clustering module to perform the adjustment of the attack graph and the reconfiguration of the network topology. The evaluation module maintains the following metrics for each cluster in the monitored network:

- $N_i$: The size (number of the node) of the cluster $i$.
- $AG_i$: the size of the attack graph for the cluster $i$.
- $GSI_i$: the global security index of the cluster $i$.
- $Risk_i$: the risk probability of the attack graph for the cluster $i$.
- $AvgBase_i$: the exponential average of the base score of vulnerabilities in the cluster $i$.

The security evaluation module also receives the alert from the intrusion detection system (IDS) once an abnormal event or traffic is detected by the IDS. This alert notification will trigger the dynamic reconfiguration strategy in the reconfiguration module to take the action.

### 6.4.2 Reconfiguration Strategies

The reconfiguration strategy using a programmable network feature of the software defined networking to protect the system. We provide an active protection and prevention system based on the vulnerabilities and alerts information in the attack graph, and trying to construct an algorithm to make the attack path more difficult to attacker through the virtual network reconfiguration. The reconfiguration can be done by the network controller through the following reconfiguration primitive operation: flow forwarding, flow redirection, flow reflection, MAC address rewriting, IP address change, packet drop, packet rate-limit, firewall or filter, and port blocking.

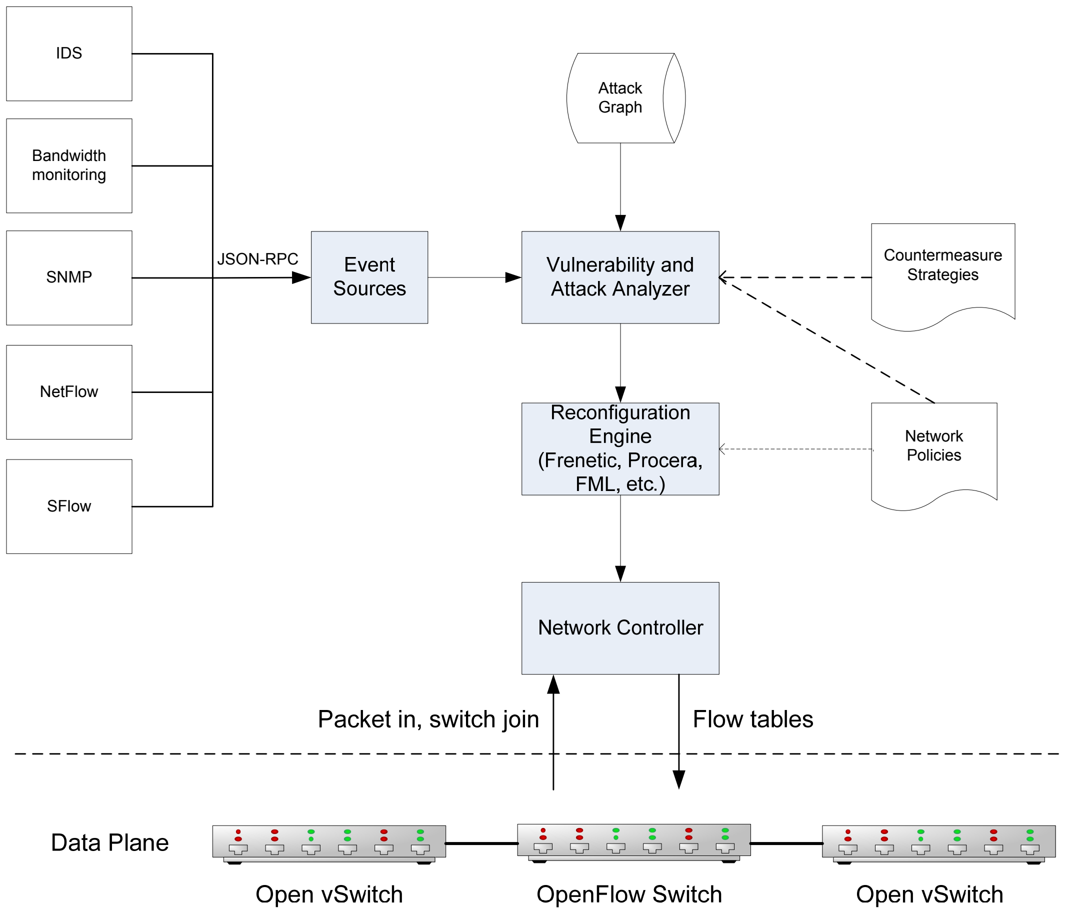


Figure 6.3: Network Reconfiguration Framework.

The reconfiguration strategy relies on the programmable network feature of the software defined networking to secure the network traffic by changing the network topology and network resource usage. The strategy could be based on the vulnerability information in the attack graph and alerts information from NIDS. The goal of the strategy is to make the attack path in the attack graph more difficult for an attacker to launch the attack. The Vulnerability and Attack Analyzer in the figure 3 will create the reconfiguration strategy based on the network events from several management and monitor servers (e.g. NIDS, bandwidth monitoring, SNMP, NetFlow, and sFlow) and information from the attack graph database, countermeasure strategy pool, and network security policy. Reconfiguration Engine is a parser for translating the policy and strategy in a high-level language definition to a low-level instructions. The reconfiguration job is done by the network controller follow the instruction from the reconfiguration engine. The reconfiguration primitive operations include: flow forwarding, flow redirection, flow reflection, MAC address rewriting, IP address change, packet drop, packet rate-limit, firewall or filter, and port blocking. The reconfiguration strategy can be divided into two different categories: static reconfiguration and dynamic reconfiguration.

*Static reconfiguration* is based on current vulnerabilities in the system to reconfigure the traffic or the virtual network. Each exploit node in the AG contains the vulnerability information. The static reconfiguration will search the critical path in the AG, which is the easiest path for the attacker and the most insecure path in the virtual network, and reconfigure the network connectivity of each VM in the path. The reconfiguration approach is selected based on the attack category and the attack type of the vulnerability. This information can be easily retrieved from the NVD database. After applying the reconfiguration strategy, the cumulative risk probability of the target node in AG will be reduced and the security index of current network also will be reduced. The security index is calculated from the number of different paths and the total length of the paths in the AG.

*Dynamic reconfiguration* is based on alerts raised by the NIDS of the system. We assume that alerts raised by NIDS are major alerts after aggregated and correlated. The raised alert means a major vulnerability has been exploited. In order to protect the system, The similar approach to the static reconfiguration is applied to prevent the system from being compromised. Instead of searching critical path in the AG, we need to match the alert with the node in AG and apply the reconfiguration using the same approach in the static reconfiguration.

The evaluation of the reconfiguration strategy is another focus in this research. The reduced risk probability of the target node in AG, the reduced value of security index of current network, which includes the number of attack path and total weight of all paths, the number of normal services affected, and the response delay to the normal traffic or service request are all considered. Scalability issue in the attack model or attack graph across multiple cloud servers and the scalability issues in the reconfiguring programmable virtual network across multiple servers on the same site or across different geographical sites are all considered in this research.

### *6.4.3 SDN-based Countermeasures Deployment and Reconfiguration*

A countermeasure is an action or a series of actions that thwarts attacks, wherein it can change the network configurations and traffic policies. Here we consider deploying a security appliance (e.g., IPS) that introduces a sequence of security actions in cloud virtual networking environments, and thus, we do not differentiate countermeasures and the security appliance in the remaining discussion. When an attack event or a software vulnerability has been identified, one (or a set) of effective countermeasures needs to be chosen among many for deployment. The attributes such as cost, time to deploy, and the potential to reduce the performance or availability of the system resources need to be considered. Using an attack graph to model the attackers' behaviors and selecting countermeasures have been well-studied, [9]. However, there is no previous work that uses attack graphs in a highly dynamic virtualized and reconfigurable networking environment and considers applying countermeasures in a dynamic fashion. In general, there are many countermeasures that can be applied to the cloud virtual networking system, depending on available security appliances that can be applied.

Several common virtual-networking-based countermeasures are listed in Table 6.1. The network reconfiguration strategies involve several levels of actions from layer-2 to the upper layers. At layer-2, virtual bridges (including tunnels that can be established between two bridges) and VLANs are the main components in the clouds virtual networking system to connect two VMs directly. A virtual bridge is an entity that attaches Virtual Interfaces (VIFs). Virtual machines on different bridges are isolated at layer 2. VIFs on the same virtual bridge but with different VLAN tags cannot communicate to each other directly. Based on this layer-2 isolation, the reconfiguration module can deploy a layer-2 network reconfiguration to isolate suspicious VMs. For example, vulnerabilities due to ARP-spoofing [48] attacks are not possible when the suspicious VM is isolated to a different bridge. As a result, this countermeasure disconnects an attack path in the attack graph causing the attacker to explore an alternate attack path. The layer-3 reconfiguration is another way to disconnect an attack path. Through the network controller, the flow table on each OpenFlow switch (e.g., both software and physical switches) can be modified to change the network topology. Similarly, upper-level counter measures such as port changing/blocking, application protocol filtering, DPI, etc., can be deployed.

Table 6.1: Possible Countermeasure Types

| Layers | Countermeasure |
|---|---|
| Layer-2 | MAC address change |
| Layer-2 | Switch reconfiguration |
| Layer-2 or 3 | Traffic redirection |
| Layer-2 or 3 | Traffic isolation |
| Layer-3 | IP address change |
| Layer-3 | Network topology change |
| Layer-4 | Change/Block port |
| App | Deep Packet Inspection (DPI) |
| App | Software patch |
| App | Quarantine |
| App/System | Software diversification |
| App/System | Software rejuvenation |
| System | Introducing VMs |
| System | Creating filtering rules |
| ... | ... |

We must note that using the virtual network reconfiguration approach at a lower layer has the advantage in that upper layer applications will experience a minimal impact. This approach is only possible when using the software-switching approach to automate the reconfiguration in a highly dynamic networking environment. Countermeasures (such as traffic isolation) can be implemented by utilizing the traffic engineering capabilities of the OpenFlow switches to restrict the capacity and reconfigure the virtual network for a suspicious flow. When a suspicious activity, such as network and port scanning is detected in the cloud system, it is important to determine whether the detected activity is indeed malicious or not. For example, attackers can intentionally hide their scanning behavior to prevent the network IDS from identifying their actions. In this situation, changing the network configuration will force the attacker to perform more explorations and in turn will make their attacking behavior stand out.

## 6.5 Distributed Attack Graph Construction

Although many attack graph generation and representation algorithms can be done in a polynomial complexity, but the evaluation phase still suffers from a scalability problem due to the state space explosion when exploring all set of possible paths with a given graph. Most attack graph adopted heuristic methods, and still there is no algorithm that can evaluate all possible states in a polynomial complexity. Also, there is a very limited number of research done on the attack graph dynamic updating.

In this work, a distributed attack graph generation and evaluation approach is developed. The high-level flow chart of the method is illustrated in figure 6.4 and the detailed steps are described as follows:

### *6.5.1 Information Collection Phase*

The first step is done by the Attack Analyzer of the system. It collects network connectivity information from the SDN network controller, collects applications & services installation status in each device from scanners (e.g. nmap or vulnerability scanner), and gathers the firewall rules policies from IPSes or firewall devices.

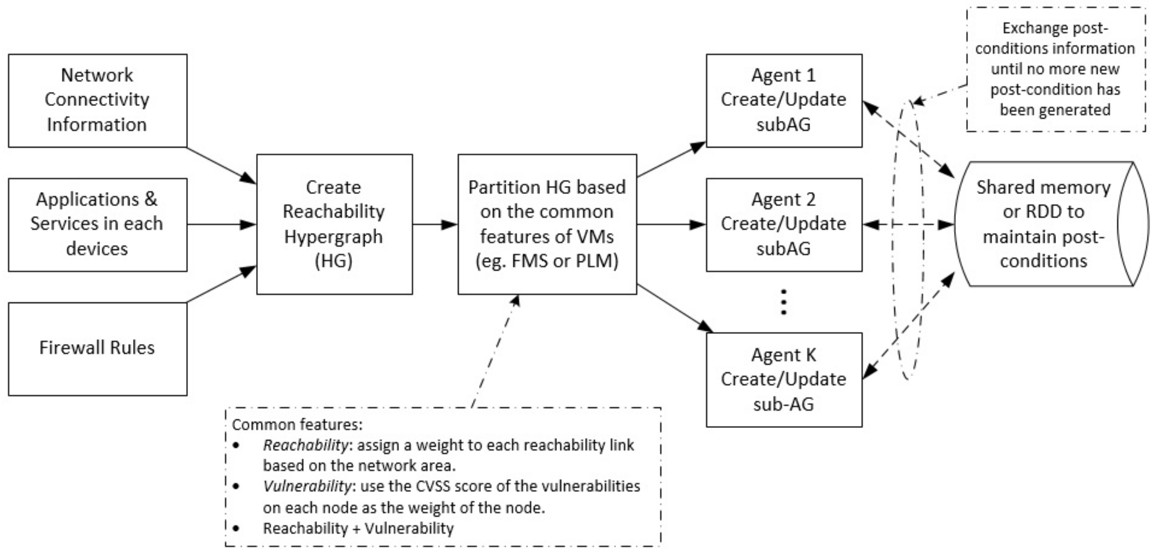

Network Connectivity Information
Applications & Services in each devices
Firewall Rules
Create Reachability Hypergraph (HG)
Partition HG based on the common features of VMs (eg. FMS or PLM)
Agent 1 Create/Update subAG
Agent 2 Create/Update subAG
Agent K Create/Update sub-AG
Exchange post-conditions information until no more new post-condition has been generated
Shared memory or RDD to maintain post-conditions
Common features:
• Reachability: assign a weight to each reachability link based on the network area.
• Vulnerability: use the CVSS score of the vulnerabilities on each node as the weight of the node.
• Reachability + Vulnerability

Figure 6.4: Distributed Attack Graph Construction Flow.

### *6.5.2 Hypergraph Creation Phase*

Attack Analyzer then creates a Reachability Hypergraph (HG) based on the collected information from the previous step. The hyper-edge in the HG represents the service reachability status between two or more VMs or devices. One hyper-edge could be connected to the same port on several different VMs, or different ports on the same VM or different VMs.

### *6.5.3 Hypergraph Partition Phase*

The HG partition module in the Attack Analyzer applies the selected HG partition algorithm to partition the hypergraph into several smaller clusters. The partition algorithm will find the mini-cuts in the HG and separate the HG into a number of *k* smaller HG with the minimum cuts between clusters. The hypergraph partition is a well-known NP-hard problem. We use the existing ad hoc solution to solve this problem, such as FMS and PLM.

When the attack graph applies the partition algorithm, it can assign weight to each node and edge in the hypergraph. The weight assignment is related to how the algorithm partitions the hypergraph. Three options here for the algorithm to assign the weight: (a) *Reachability*: assign a weight to each reachability link based on the network area or proximity, (b) *Vulnerability*: use the CVSS score of the vulnerabilities on each node as the weight of the node, and (c) consider both reachability and vulnerability.

### *6.5.4 Distributed Attack Graph Phase*

Based on the number of clusters after performing the HG partition algorithm, Attack Analyzer deploys the same number of attack graph construction agents in the virtual network and distributes the sub-hybergraph information from the corresponding cluster to each agent for the sub-attack graph construction. Each agent then executes the algorithm in figure 6.5 to create a sub-attack graph.

Each agent begins with finding the necessary information in the sub-hypergraph, e.g., the edge nodes in the sub-hypergraph, external nodes connecting to these edge nodes, and each internal node's vulnerabilities and reachability information. With these necessary information, each agent starts to create the sub-attack graph. While creating the attack graph, each agent will generate new post-conditions of the inference rules in the attack graph construction engine. The new post-condition represents the information or privilege gain for the attacker. The attacker can use this gain to exploit other possible vulnerabilities in the system. Therefore, we need to keep track all the new post-conditions generated from each agent, and allow each agent to get the new generated post-conditions from other agents.

We use the shared memory and resilient distributed dataset (RDD) to maintain these new generated post-conditions. When there is a new post-condition generated in an agent, the agent will write the information in the shared memory. When an agent finishes the

```
Require: Sub-Hyper Graph ($SHG$)
 1: $EN$ = Find_Edge_Nodes($SHG$);
 2: for each $e \in EN$ do
 3:     $Attackers$ = Find_External_Nodes_with_priv($e$)
 4: end for
 5: $Vulns$ = Find_Vulns_Info($SHG$)
 6: $Reaches$ = Find_Reach_Info($SHG$)
 7: $SAG$ = Create_SubAG($Attackers, Vulns, Reaches$)
 8: while $true$ do
 9:     $PostConds$ = Find_New_PostConds($SAG$)
10:     for each $p \in PostConds$ do
11:         Write_To_Share_Memory(Agent, $p$)
12:     end for
13:     $PreConds$ = Read_Update_Share_Memory(Agent)
14:     if $PreConds.size() == 0$ then
15:         $break$
16:     else
17:         for each $pc \in PreConds$ do
18:             for each $vuln \in SHG$ do
19:                 if is_precondition($pc, vuln$) then
20:                     Update_AG($SAG, pc$)
21:                 end if
22:             end for
23:         end for
24:     end if
25: end while
```

Figure 6.5: Sub-Attack Graph Creation Algorithm - SubAG_Creation

attack graph generation, it will read new post-conditions from the shared memory and call the attack graph update function based on these new post-conditions. The update function will try to match the new post-condition with each pre-condition in the current sub-AG. If there is a match and is able to exploit a certain vulnerability in a node, the agent will update the sub-AG accordingly. If the content of the shared memory is empty and no any new post-condition generated from all agents, each agent will put themselves in the passive mode and stop the creation algorithm.

## 6.6 Experiments

We created a web-based tool to run the distributed attack graph construction experiment and to evaluate the performance of our solution. Table 6.2 shows the running time of 8 different tests with different network size (number of nodes) and different number of agents.

Table 6.2: Running Time of the Distributed Attack Graph Construction

| Nodes | Edges | AG-nodes | AG-edges | 1 Agent (s) | 2 Agents (s) | 3 Agents (s) |
|---|---|---|---|---|---|---|
| 11 | 52 | 39 | 53 | 2.3 | 4.2 | 6.4 |
| 32 | 152 | 55 | 82 | 4.2 | 6.3 | 8.3 |
| 74 | 369 | 102 | 224 | 7.2 | 10.2 | 13.5 |
| 124 | 1280 | 234 | 522 | 13.3 | 19.2 | 20.1 |
| 165 | 2321 | 398 | 790 | 28.4 | 26.3 | 25.9 |
| 199 | 3420 | 792 | 1323 | 38.2 | 27.3 | 26.1 |
| 230 | 5892 | 1342 | 2319 | 63.2 | 48.5 | 40.3 |
| 360 | 6203 | 2021 | 3080 | 83.9 | 63.2 | 55.4 |

When the number of nodes in the hypergraph is small, the performance of a single agent is better than the performance of multiple agents. This is because the overhead introduced by the post-conditions information exchange between agents and the attack graph updating. When the network size increases to 165, the performance of attack graph generation using multiple agents is better than the performance of the same process in a single agent. When the number of nodes further increased, more agents will get better performance.

Figure 6.6 shows the performance comparison of the different distributed attack graph construction deployment. We can conclude from the experiment results that while the number of nodes in the hypergraph increase, the performance with more agents is better than the performance with less number of agents. The benefit of the distributed algorithm becomes obvious in a large size of network or a network with the more complicated reachability situation.

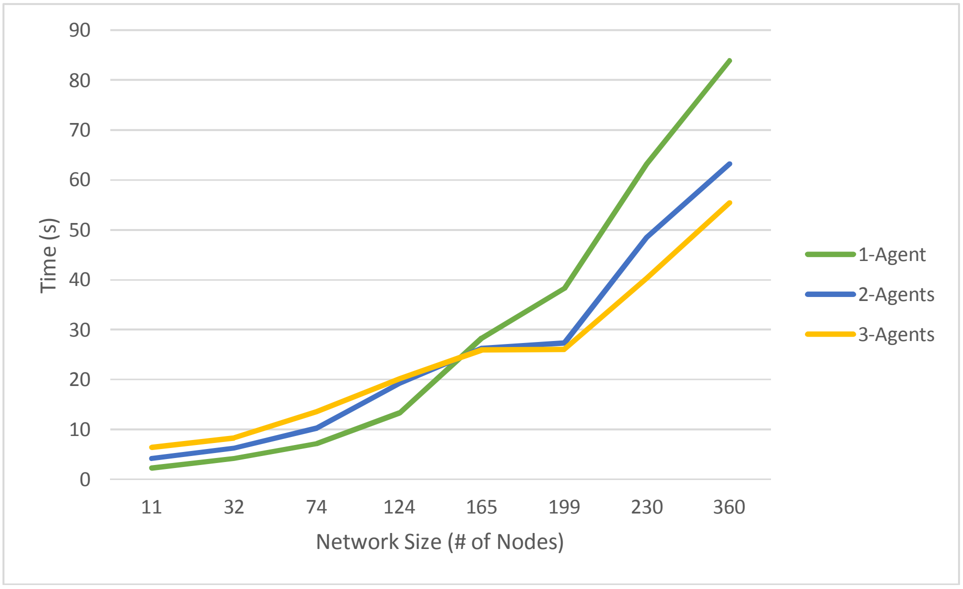


Figure 6.6: Performance Comparison of Different Distributed Attack Graph Construction Deployment.

## 6.7 Conclusion and Future Work

In this work, we investigate into the scalability issues when using attack graph-based security analysis model in a programmable and virtualized networking environment, e.g., a datacenter networking environment with Software Defined Networking (SDN) support. We present an attack graph-based security/vulnerability analysis framework to track the vulnerabilities in each virtual machine (VM) and the vulnerability dependency among VMs. To address the scalability issue using attack graphs in a large datacenter networking environment, an attack graph should be confined by a reasonable size. To this end, we create an SDN-based reconfiguration model by leveraging the programmable network features using the SDN framework to isolate, quarantine, and inspect the traffic sending from vulnerable VMs. The presented model considers both security and QoS constrains for end users. It improves virtual network environment's security with the minimal level of intrusiveness to user's normal traffic. We achieve this goal by using moving target defense techniques that force attackers to continuously chase their targets, deterring, and eliminating attacks without interrupting normal network traffic. This will eliminate the attacker's time and space advantage and create agility against advanced persistent threats.

# Chapter 7

# FUTURE WORK

In the future, I will focus on constructing analytical models to describe attacker's behavior and predict the actions from attackers. A microcosm of cloud system is also an interesting and important research direction from the security and scalability perspective of cloud-based applications. Anomaly detection to tackle the unknown and zero-day attacks and SDN-based proactive countermeasure deployment strategies are also important tasks to secure the cloud system and mitigate the impact of attacks.

## 7.1 Anomaly Detection Model to Tackle Unknown and Zero-day Attacks

To address the zero-day attack issue, we need an ability to identify zero-day vulnerabilities because they are believed to be used primarily for carrying out targeted attacks. The research challenges here are how to accurately identify the zero-day vulnerability from the internal behavior of VM and how to precisely detect suspicious flows which contain zero-day unknown attack from the anomaly-based intrusion detection system. Zero-day vulnerabilities does not appear in SAG since they are unknown to the scanner and they even do not exist in the vulnerability database. One approach to reduce the false negative rate of the intrusion detection is to improve the scanner's ability to uncover unknown vulnerabilities. The intelligent vulnerability scanner is not just a network-level scanner, it also can become a host-level vulnerability scanner in a virtualization environment. Using VM Introspection technology [54], the scanner is able to non-intrusively monitor and detect the internal behavior in all VMs from the hypervisor [110].

I will investigate in the zero-day vulnerability identification through a comprehensive approach, which includes network traffic inspection and host-based process behavior introspection. Network Controller in SDN and network-based intrusion detection system conduct the anomaly-based intrusion detection to filter out and isolate suspicious flows. The VM Process Monitor in our preliminary work [110] continuously monitor the sending and receiving processes of the flow in corresponding VMs, etc. Moreover, we will explore unsupervised machine learning approaches, such as SOM [111], hierarchical clustering [112], subspace clustering [113] and statistical inference [114][115], to detect anomalous traffic. It is interesting to investigate into how to use SDN to improve the detection accuracy based on unsupervised detection models and how to incorporate the detection into attack graph-based analysis models.

Chapter 8

# CONCLUSION

The most common provision service or delivery models in the could computing are IaaS, PaaS and SaaS. These three service models provide infrastructure resources, application platform and software to customers. In the past few years, many efforts from academy and industry have been devoted to these areas. In these dynamic provisioning schemes for virtual resources, many challenges still need to be addressed. For example, most of virtual network resources provisioning still based on traditional virtual network technologies, such as VLAN and VPN, to provide secured and isolated communication channel. Today's cloud network model has largely focused on providing basic reachability using dynamic or static IP address assigned to customer VMs, with basic firewall capabilities available on each virtual server. They are lack of fine-grained security, privacy, audit compliance, unpredictable performance, and poor reliability.

This dissertation presents our previous work NICE, which a framework to detect and mitigate collaborative attacks in the cloud virtual networking environment. NICE utilizes the attack graph model to conduct attacks detection and prediction. The proposed solution investigates how to use the programmability of software switches based solutions to improve the detection accuracy and defeat victim exploitation phases of collaborative attacks. The system performance evaluation demonstrates the feasibility of NICE and shows that the proposed solution can significantly reduce the risk of the cloud system from being exploited and abused by internal and external attackers. Another existing work integrated a network based intrusion detection system to monitor and detect the traffic in the virtual network and a non-intrusive host based suspicious process monitor and detection system using "out-of-box" VMI technology. Moreover, the host-based intrusion detection is based on VM introspection techniques that do not need the implement special codes in users' VMs.

In order to increase the detection accuracy of intrusion and presence of malware in the cloud, we need to develop a more sophisticated malware analysis and detection system for our framework in the future to cover different types of VMs, operation systems, vulnerabilities, and attacks.

# REFERENCES

[1] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, May 2010.

[2] W. Dawoud, I. Takouna, and C. Meinel, "Infrastructure as a service security: Challenges and solutions," in *2010 The 7th International Conference on Informatics and Systems (INFOS)*, 2010, pp. 1–8.

[3] H. Takabi, J. B. Joshi, and G. Ahn, "Security and privacy challenges in cloud computing environments," *IEEE Security & Privacy*, vol. 8, no. 6, pp. 24–31, Dec. 2010.

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

[5] D. Huang, X. Zhang, M. Kang, and J. Luo, "MobiCloud: building secure cloud framework for mobile computing and communication," in *2010 Fifth IEEE International Symposium on Service Oriented System Engineering (SOSE)*, Jun. 2010, pp. 27 –34.

[6] D. Huang, "Mobile cloud computing," *IEEE COMSOC Multimedia Communications Technical Committee (MMTC) E-Letter*, vol. 6, no. 10, pp. 27–31, 2011.

[7] B. Joshi, A. Vijayan, and B. Joshi, "Securing cloud computing environment against DDoS attacks," in *Computer Communication and Informatics (ICCCI), 2012 International Conference on*, Jan. 2012.

[8] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[9] C.-J. Chung, P. Khatkar, T. Xing, J. Lee, and D. Huang, "NICE: network intrusion detection and countermeasure selection in virtual network systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 10, no. 4, pp. 198–211, Jul. 2013.

[10] C.-J. Chung, J. Cui, P. Khatkar, and D. Huang, "Non-intrusive process-based monitoring system to mitigate and prevent VM vulnerability explorations," in *2013 9th International Conference Conference on Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom)*, Oct. 2013, pp. 21–30.

[11] C.-J. Chung, T. Xing, D. Huang, D. Medhi, and K. Trivedi, "SeReNe: on establishing secure and resilient networking services for an SDN-based multi-tenant datacenter environment," in *Workshop on Dependability Issues on SDN and NFV (DISN 15)*, 2015.

[12] W. Hagen, *Professional Xen Virtualization*. Wiley Publishing, Inc., 2008.

[13] C. Takemura and L. Crawford, *The Book of XEN*. No Starch Press, 2009.

[14] The Linux Foundation, “Linux bridge,” 2012. [Online]. Available: http://www.linuxfoundation.org/collaborate/workgroups/networking/bridge

[15] IEEE Standards Association, “IEEE 802.1d standard,” 2012. [Online]. Available: https://standards.ieee.org/about/get/802/802.1.html

[16] A. Saha and M. Molle, “Thinking outside the box: extending 802.1x authentication to remote ”splitter” ports by combining physical and data link layer techniques,” in *28th Annual IEEE International Conference on Local Computer Networks, 2003. LCN ’03. Proceedings*, 2003, pp. 324–333.

[17] J. Pettit, J. Gross, B. Pfaff, M. Casado, and S. Crosby, “Virtual switching in an era of advanced edges,” in *2nd Workshop on Data Center – Converged and Virtual Ethernet Switching (DC-CAVES), ITC*, vol. 22, 2010.

[18] “Open vSwitch project,” http://openvswitch.org, May 2012.

[19] “Citrix XenServer.” [Online]. Available: http://www.citrix.com/xenserver

[20] “OpenStack.” [Online]. Available: http://www.openstack.org/

[21] “Openflow.” [Online]. Available: http://www.openflow.org/wp/learnmore/

[22] H. Wu, Y. Ding, C. Winer, and L. Yao, “Network security for virtual machine in cloud computing,” in *Computer Sciences and Convergence Information Technology (ICCIT), 2010 5th International Conference on*, Dec. 2010, pp. 18 –21.

[23] S. Kumar, “Service cloaking and authentication at data link layer,” in *2nd International Symposium on Advanced Networks and Telecommunication Systems, 2008. ANTS ’08*, 2008, pp. 1–3.

[24] I. Cisco Systems, “Virtual lan security best practices,” 2002. [Online]. Available: http://www.cisco.com/warp/public/cc/pd/si/casi/ca6000/prodlit/vlnwpwp.pdf

[25] Coud Sercurity Alliance, “Top threats to cloud computing v1.0,” https://cloudsecurityalliance.org/topthreats/csathreats.v1.0.pdf, March 2010.

[26] Z. Duan, P. Chen, F. Sanchez, Y. Dong, M. Stephenson, and J. Barker, “Detecting spam zombies by monitoring outgoing messages,” *Dependable and Secure Computing, IEEE Transactions on*, vol. 9, no. 2, pp. 198 –210, Apr. 2012.

[27] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, “BotHunter: detecting malware infection through IDS-driven dialog correlation,” in *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, ser. SS’07, 2007, pp. 1–16.

[28] G. Gu, J. Zhang, and W. Lee, “BotSniffer: detecting botnet command and control channels in network traffic,” in *Proceedings of 15th Ann. Network and Distributed Sytem Security Symposium*, ser. NDSS’08, 2008.

[29] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, “Automated generation and analysis of attack graphs,” in *2002 IEEE Symposium on Security and Privacy, 2002. Proceedings*. IEEE, 2002, pp. 273– 284.

[30] “NuSMV: A new symbolic model checker,” http://afrodite.itc.it:1024/~nusmv.

[31] S. H. Ahmadinejad, S. Jalili, and M. Abadi, “A hybrid model for correlating alerts of known and unknown attack scenarios and updating attack graphs,” *Computer Networks*, vol. 55, no. 9, pp. 2221–2240, Jun. 2011.

[32] X. Ou, S. Govindavajhala, and A. W. Appel, “MulVAL: a logic-based network security analyzer,” in *Proceedings of the 14th conference on USENIX Security Symposium - Volume 14*. Berkeley, CA, USA: USENIX Association, 2005, pp. 8–8.

[33] R. Sadoddin and A. Ghorbani, “Alert correlation survey: framework and techniques,” in *Proceedings of the 2006 International Conference on Privacy, Security and Trust: Bridge the Gap Between PST Technologies and Business Services*, ser. PST ’06. New York, NY, USA: ACM, 2006, pp. 37:1–37:10.

[34] L. Wang, A. Liu, and S. Jajodia, “Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts,” *Computer Communications*, vol. 29, no. 15, pp. 2917–2933, Sep. 2006.

[35] S. Roschke, F. Cheng, and C. Meinel, “A new alert correlation algorithm based on attack graph,” in *Computational Intelligence in Security for Information Systems*, ser. Lecture Notes in Computer Science. Springer, 2011, vol. 6694, pp. 58–67.

[36] A. Roy, D. S. Kim, and K. Trivedi, “Scalable optimal countermeasure selection using implicit enumeration on attack countermeasure trees,” in *Dependable Systems Networks (DSN), 2012 IEEE/IFIP 42st International Conference on*, 2012.

[37] N. Poolsappasit, R. Dewri, and I. Ray, “Dynamic security risk management using bayesian attack graphs,” *Dependable and Secure Computing, IEEE Transactions on*, vol. 9, no. 1, pp. 61 –74, Feb. 2012.

[38] Open Networking Fundation, “Software-defined networking: The new norm for networks,” *ONF White Paper*, April 2012.

[39] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: enabling innovation in campus networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[40] X. Ou, W. F. Boyer, and M. A. McQueen, “A scalable approach to attack graph generation,” in *Proceedings of the 13th ACM conference on Computer and communications security*, ser. CCS ’06. New York, NY, USA: ACM, 2006, pp. 336–345.

[41] Mitre Corporation, “Common vulnerabilities and exposures, CVE,” http://cve.mitre.org/.

[42] P. Mell, K. Scarfone, and S. Romanosky, “Common vulnerability scoring system (CVSS),” http://www.first.org/cvss/cvss-guide.html, May 2010.

[43] O. Database, “Open source vulnerability database (OVSDB),” http://osvdb.org/.

[44] NIST, “National vulnerability database, NVD,” http://nvd.nist.gov.

[45] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, “NOX: towards an operating system for networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, Jul. 2008.

[46] X. Ou and A. Singhal, *Quantitative Security Risk Assessment of Enterprise Networks*. Springer, Nov. 2011.

[47] M. Frigault and L. Wang, “Measuring network security using bayesian Network-Based attack graphs,” in *Computer Software and Applications, 2008. COMPSAC ’08. 32nd Annual IEEE International*, Aug. 2008, pp. 698 –703.

[48] K. Kwon, S. Ahn, and J. Chung, “Network security management using ARP spoofing,” in *Computational Science and Its Applications – ICCSA 2004*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2004, vol. 3043, pp. 142–149.

[49] “Metasploit,” http://www.metasploit.com.

[50] “Armitage,” http://www.fastandeasyhacking.com.

[51] M. Tupper and A. Zincir-Heywood, “VEA-bility security metric: A network security analysis tool,” in *Third International Conference on Availability, Reliability and Security, 2008. ARES 08*, Mar. 2008, pp. 950 –957.

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

[53] H. Qian and D. Medhi, “Server operational cost optimization for cloud computing service providers over a time horizon,” in *Proceedings of the 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services*, ser. HotICE’11, 2011.

[54] T. Garfinkel and M. Rosenblum, “A virtual machine introspection based architecture for intrusion detection,” in *Proc. of the 10th Annual Network and Distributed Systems Security Symposium*, 2003.

[55] X. Jiang, X. Wang, and D. Xu, “Stealthy malware detection and monitoring through VMM-based ”out-of-the-box” semantic view reconstruction,” *ACM Transaction on Information and System Securirty*, vol. 13, no. 2, pp. 12:1–12:28, Mar. 2010.

[56] X. Jiang and X. Wang, “Out-of-the-Box monitoring of VM-Based high-interaction honeypots,” in *Recent Advances in Intrusion Detection*, ser. Lecture Notes in Computer Science, C. Kruegel, R. Lippmann, and A. Clark, Eds. Springer Berlin Heidelberg, Jan. 2007, no. 4637, pp. 198–218.

[57] "VMware NSX Network Virualization." [Online]. Available: http://www.vmware.com/products/nsx/

[58] P. Salvador, A. Nogueira, U. Franca, and R. Valadas, "Framework for zombie detection using neural networks," in *Fourth International Conference on Internet Monitoring and Protection, 2009. ICIMP '09*, 2009, pp. 14–20.

[59] S. T. Jones, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Antfarm: Tracking processes in a virtual machine environment," in *Proceedings of the USENIX Annual Technical Conference*, 2006, pp. 1–4.

[60] ——, "VMM-based hidden process detection and identification using lycosid," in *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, ser. VEE '08. New York, NY, USA: ACM, 2008, pp. 91–100.

[61] C. Benninger, S. Neville, Y. Yazir, C. Matthews, and Y. Coady, "Maitland: Lighter-weight VM introspection to support cyber-security in the cloud," in *2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*, Jun. 2012, pp. 471 –478.

[62] D. Srinivasan, Z. Wang, X. Jiang, and D. Xu, "Process out-grafting: an efficient "out-of-VM" approach for fine-grained process execution monitoring," in *Proceedings of the 18th ACM conference on Computer and communications security*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 363–374.

[63] O. M. Sheyner, "Scenario graphs and attack graphs," Ph.D. dissertation, Carnegie Mellon University, 2004.

[64] P. Ammann, D. Wijesekera, and S. Kaushik, "Scalable, graph-based network vulnerability analysis," in *Proceedings of the 9th ACM conference on Computer and communications security*, ser. CCS '02. New York, NY, USA: ACM, 2002, pp. 217–224.

[65] S. Jajodia, "Topological analysis of network attack vulnerability," in *Proceedings of the 2nd ACM symposium on Information, computer and communications security*, ser. ASIACCS '07. New York, NY, USA: ACM, 2007, pp. 2–2.

[66] H. Qian, D. Medhi, and T. Trivedi, "A hierarchical model to evaluate quality of experience of online services hosted by cloud computing," in *2011 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2011, pp. 105–112.

[67] "Kernel based Virtual Machine (KVM)." [Online]. Available: http://www.linux-kvm.org/

[68] "Open vSwitch project," http://openvswitch.org/. [Online]. Available: \url{http://openvswitch.org/}

[69] S. Roschke, F. Cheng, and C. Meinel, "A flexible and efficient alert correlation platform for distributed IDS," in *2010 4th International Conference on Network and System Security (NSS)*. IEEE, Sep. 2010, pp. 24–31.

[70] "Volatility." [Online]. Available: https://code.google.com/volatility/

[71] "Unixbench – a Unix benchmark suite." [Online]. Available: http://www.tux.org/pub/tux/niemi/unixbench/

[72] VMware INC., "Vmware vcloud networking and security overview, available at http://www.vmware.com/files/pdf/products/vcns/vCloud-Networking-and-Security-Overview-Whitepaper.pdf," 2012.

[73] J. Oberheide, E. Cooke, and F. Jahanian, "CloudAV: n-version antivirus in the network cloud," in *Proceedings of the 17th USENIX Security symposium*, 2008, pp. 91–106. [Online]. Available: http://static.usenix.org/events/sec08/tech/full_papers/oberheide/oberheide_html/

[74] F. Boldewin, "Peacomm. c cracking the nut-shell," *Anti Rootkit*, 2007. [Online]. Available: https://www.os3.nl/_media/2007-2008/students/matthew_steggink/rp1/peacomm.c.pdf

[75] W. Yan and N. Ansari, "Why anti-virus products slow down your machine?" in *Proceedings of 18th Internatonal Conference on Computer Communications and Networks, 2009. ICCCN 2009*, 2009, pp. 1–6.

[76] M. I. Sharif, W. Lee, W. Cui, and A. Lanzi, "Secure in-vm monitoring using hardware virtualization," in *Proceedings of the 16th ACM conference on Computer and communications security*, 2009, pp. 477–487. [Online]. Available: http://dl.acm.org/citation.cfm?id=1653720

[77] C. Willems, R. Hund, and T. Holz, "CXPInspector: hypervisor-based, hardware-assisted system monitoring," Horst Gortz Institute for IT Security, Tech. Rep. TR-HGI-2012-002, Nov. 2012. [Online]. Available: https://www.syssec.rub.de/media/emma/veroeffentlichungen/2012/11/26/TR-HGI-2012-002.pdf

[78] B. Dolan-Gavitt, B. Payne, and W. Lee, "Leveraging forensic tools for virtual machine introspection," 2011. [Online]. Available: https://smartech.gatech.edu/handle/1853/38424

[79] K. Nance, M. Bishop, and B. Hay, "Virtual machine introspection: Observation or interference?" *Security & Privacy, IEEE*, vol. 6, no. 5, pp. 32–37, 2008. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4639020

[80] "Citrix XenServer," http://www.citrix.com/xenserver. [Online]. Available: http://www.citrix.com/xenserver

[81] "Xen Cloud Platform (XCP)," http://www.xen.org/products/cloudxen.html. [Online]. Available: http://www.xen.org/products/cloudxen.html

[82] R. Wojtczuk, "Subverting the xen hypervisor," *Black Hat USA*, vol. 2008, 2008. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.167.5640&rep=rep1&type=pdf

[83] A. van der Merwe, M. Loock, and M. Dabrowski, “Characteristics and responsibilities involved in a phishing attack,” in *Proceedings of the 4th international symposium on Information and communication technologies*, ser. WISICT ’05. Trinity College Dublin, 2005, pp. 249–254. [Online]. Available: http://dl.acm.org/citation.cfm?id=1071752.1071800

[84] S. Fogie, J. Grossman, R. Hansen, A. Rager, and P. D. Petkov, *XSS Attacks: Cross Site Scripting Exploits and Defense*. Syngress Publishing, 2007.

[85] K. Rieck, T. Krueger, and A. Dewald, “Cujo: efficient detection and prevention of drive-by-download attacks,” in *Proceedings of the 26th Annual Computer Security Applications Conference*, ser. ACSAC ’10. New York, NY, USA: ACM, 2010, pp. 31–39. [Online]. Available: http://doi.acm.org/10.1145/1920261.1920267

[86] L. Wang, A. Liu, and S. Jajodia, “Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts,” *Computer Communications*, vol. 29, no. 15, pp. 2917–2933, Sep. 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S014036640600137X

[87] H. T. Elshoush and I. M. Osman, “Alert correlation in collaborative intelligent intrusion detection systems–A survey,” *Applied Soft Computing*, vol. 11, no. 7, pp. 4349–4365, Oct. 2011.

[88] D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, S. Roschke, F. Cheng, and C. Meinel, “A new alert correlation algorithm based on attack graph,” in *Computational Intelligence in Security for Information Systems*, Á. Herrero and E. Corchado, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, vol. 6694, pp. 58–67. [Online]. Available: http://www.springerlink.com.ezproxy1.lib.asu.edu/content/3261404r85752460/

[89] “Snort,” http://www.snort.org.

[90] A. Srivastava and J. Giffin, “Tamper-resistant, application-aware blocking of malicious network connections,” in *Recent Advances in Intrusion Detection*, ser. Lecture Notes in Computer Science, R. Lippmann, E. Kirda, and A. Trachtenberg, Eds. Springer Berlin Heidelberg, Jan. 2008, no. 5230, pp. 39–58. [Online]. Available: http://link.springer.com.ezproxy1.lib.asu.edu/chapter/10.1007/978-3-540-87403-4_3

[91] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson, “FRESCO: modular composable security services for software-defined networks,” in *Network & Distributed System Security Symposium (NDSS’13)*, ser. NDSS’13, 2013. [Online]. Available: http://www.csl.sri.com/users/vinod/papers/fresco.pdf

[92] Z. Gu, Z. Deng, D. Xu, and X. Jiang, “Process implanting: A new active introspection framework for virtualization,” in *2011 30th IEEE Symposium on Reliable Distributed Systems (SRDS)*, Oct. 2011, pp. 147 –156.

[93] D. Perez-Botero, J. Szefer, and R. B. Lee, "Characterizing hypervisor vulnerabilities in cloud computing servers," in *Proceedings of the 2013 International Workshop on Security in Cloud Computing*, ser. Cloud Computing '13. New York, NY, USA: ACM, 2013, pp. 3–10. [Online]. Available: http://doi.acm.org/10.1145/2484402.2484406

[94] K. Owens, "Securing virtual compute infrastructure in the cloud," Savvis Communications Corp, White paper, 2009. [Online]. Available: http://viewer.media.bitpipe.com/1018468865_999/1296679360_880/Securing-Virtual-Compute-Infrastructure-in-the-Cloud.pdf

[95] A. Koto, H. Yamada, K. Ohmura, and K. Kono, "Towards unobtrusive VM live migration for cloud computing platforms," in *Proceedings of the Asia-Pacific Workshop on Systems*, ser. APSYS '12. New York, NY, USA: ACM, 2012, pp. 7:1–7:6. [Online]. Available: http://doi.acm.org/10.1145/2349896.2349903

[96] E. Kotsovinos, "Virtualization: Blessing or curse?" *Commun. ACM*, vol. 54, no. 1, pp. 61–65, Jan. 2011. [Online]. Available: http://doi.acm.org/10.1145/1866739.1866754

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

[98] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM conference on Computer and communications security*, ser. CCS '09. New York, NY, USA: ACM, 2009, pp. 199–212. [Online]. Available: http://doi.acm.org.ezproxy1.lib.asu.edu/10.1145/1653662.1653687

[99] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks." in *NSDI*, vol. 10, 2010, pp. 19–19. [Online]. Available: https://www.usenix.org/legacy/event/nsdi10/tech/full_papers/al-fares.pdf

[100] J. B. Hong and D. S. Kim, "Scalable security model generation and analysis using k-importance measures," in *Security and Privacy in Communication Networks*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, T. Zia, A. Zomaya, V. Varadharajan, and M. Mao, Eds. Springer International Publishing, Jan. 2013, no. 127, pp. 270–287. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-319-04283-1_17

[101] C. Tankard, "Advanced persistent threats and how to monitor and deter them," *Network Security*, vol. 2011, no. 8, pp. 16–19, Aug. 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1353485811700861

[102] J. Hong and D.-S. Kim, "HARMs: hierarchical attack representation models for network security analysis," *Australian Information Security Management Conference*, Dec. 2012. [Online]. Available: http://ro.ecu.edu.au/ism/146

[103] S. Fortunato, “Community detection in graphs,” *Physics Reports*, vol. 486, no. 3–5, pp. 75–174, Feb. 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0370157309002841

[104] X. Xu, N. Yuruk, Z. Feng, and T. A. J. Schweiger, “SCAN: a structural clustering algorithm for networks,” in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD ’07. New York, NY, USA: ACM, 2007, pp. 824–833. [Online]. Available: http://doi.acm.org/10.1145/1281192.1281280

[105] C. Ding, X. He, H. Zha, M. Gu, and H. Simon, “A min-max cut algorithm for graph partitioning and data clustering,” in *ICDM 2001, Proceedings IEEE International Conference on Data Mining, 2001*, 2001, pp. 107–114.

[106] J. Shi and J. Malik, “Normalized cuts and image segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905, Aug. 2000.

[107] R. Guimerà and L. A. Nunes Amaral, “Functional cartography of complex metabolic networks,” *Nature*, vol. 433, no. 7028, pp. 895–900, Feb. 2005. [Online]. Available: http://www.nature.com/nature/journal/v433/n7028/abs/nature03288.html

[108] M. E. J. Newman and M. Girvan, “Finding and evaluating community structure in networks,” *Phys. Rev. E*, vol. 69, p. 026113, Feb 2004. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevE.69.026113

[109] A. Clauset, M. E. J. Newman, and C. Moore, “Finding community structure in very large networks,” *Phys. Rev. E*, vol. 70, p. 066111, Dec 2004. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevE.70.066111

[110] C.-J. Chung, J. Cui, P. Khatkar, and D. Huang, “Non-intrusive process-based monitoring system to mitigate and prevent vm vulnerability explorations,” in *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom ’13), 2012 IEEE 9th International Conference on*, 2013.

[111] T. Kohonen, *Self-Organizing Maps*. Berlin: Springer-Verlag, 2001, vol. 3rd.

[112] L. Khan, M. Awad, and B. Thuraisingham, “A new intrusion detection system using support vector machines and hierarchical clustering,” *The VLDB Journal*, vol. 16, no. 4, pp. 507–521, Oct. 2007, 00117. [Online]. Available: http://link.springer.com.ezproxy1.lib.asu.edu/article/10.1007/s00778-006-0002-5

[113] L. Parsons, E. Haque, and H. Liu, “Subspace clustering for high dimensional data: a review,” *SIGKDD Explor. Newsl.*, vol. 6, no. 1, pp. 90–105, Jun. 2004, 00609.

[114] A. Scherrer, N. Larrieu, P. Owezarski, P. Borgnat, and P. Abry, “Non-gaussian and long memory statistical characterizations for internet traffic with anomalies,” *IEEE Transactions on Dependable and Secure Computing*, vol. 4, no. 1, pp. 56–70, 2007.

[115] F. Simmross-Wattenberg, J. Asensio-Perez, P. Casaseca-de-la Higuera, M. Martin-Fernandez, I. Dimitriadis, and C. Alberola-Lopez, “Anomaly detection in network traffic based on statistical inference and alpha-stable modeling,” *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 4, pp. 494–509, 2011.