IHIDS: Introspection-Based Hybrid Intrusion Detection System in Cloud Environment

Amita Kashyap[a,b] and G. Sravan Kumar
[a]Department of Computer Science and Engineering
Sreyas Inst. of Engg. & Technology
Jawaharlal Nehru Technological University, Hyderabad, India
Email:parmar.amita@gmail.com
sravankumar.g@sreyas.ac.in

Sunita Jangir, Emmanuel S. Pilli and Preeti Mishra
[b]Department of Computer Science and Engineering
Malaviya National Institute of Technology, Jaipur, India
Email:{sanijangir, scholar.preeti}@gmail.com
espilli.cse@mnit.ac.in

***Abstract*—Cloud Computing offers on demand services of infrastructure (servers, storage and network), platform (operating systems and databases) and software (applications) over the Internet on pay-as-you-use model. Security is of pre-eminent interest in this new era of computing as many e-commerce and social networking sites move their operations to the Cloud. One of the approaches to protect the Cloud Environment is to create a highly efficient system for Intrusion Detection which can handle both outsider as well as insider attacks. This paper proposes a misuse detection system at hypervisor layer to detect the inter-VM attacks in a virtual environment. It is named as Introspection-based Hybrid approach for Intrusion Detection System for Cloud (IHIDS). Virtual Machine Introspection (VMI) provides the needed system call sequences at the hypervisor layer. The proposed Hybrid model integrates information and evidence from system-wide process monitoring and the virtual network traffic analysis. The main focus of the paper is to detect intrusion at hypervisor layer by simultaneously analyzing network traffic and system calls. Anomalous behavior among tenant VMs caused because of Flooding and Port Scanning attacks is detected. The proposed technique is light-weight as there is no need to hold a large amount of data at any instance of time. It is efficient as well, in terms of complexity and resources it utilizes. It is validated against the dataset created in our lab and results are encouraging.**

***Key words*- Virtual Machine Introspection, Process Monitoring, Intrusion Detection**

I. Introduction

Cloud Computing delivers on-demand service to users with less service provider interaction which seems great, but also increases the threat of cyber attacks. Every year, several organizations report events of attacks. For example, an Annual Security report from Cisco [1] shows that the Boston Marathon bombing made up 40% of the spam messages sent across the world and ENSIA [2] reported attack on Dropbox by Distributed Denial-of-Service (DDoS) which affected the 15 hrs of functionality and Amazon cloud also infected with DDoS botnets and many more attacks every year. Panjwani et al. [3] reported in their report that network scanning activity initialized almost 50% of the attack against cyber systems.

Various common intrusions, in turn, causes confidentiality, availability and integrity issues to cloud services and resources. Some of the common attacks are insider attack or misuse, flooding attack, port scanning attack, backdoor channel attack, a user to root attack, attack on VM or hypervisor and so on. Port scanning attacks search for open ports through which, attacker can gain access to a computer. Attacker sends a message to a port and response will show the status of the port which helps an attacker to identify the attack surface and its vulnerabilities and helps to launch future attacks. Flooding attack is a type of Denial-of-Service (DoS) attack that makes system resources busy and prevent the legitimate users from accessing the service. They occur when attacker overloads a server with more requests than the server can handle. DDoS will occur when an attacker takes control of computer and flood a particular service such as email with messages with enormous blocks of data. Anti-virus software, firewalls, and email filters can help us to get rid of DoS attack and manage unwanted traffic at hosted environment.

There are various detection mechanisms based on virtualization and traditional approach. Traditional approaches include signature matching and dynamic analysis technique, and virtualization-based approach includes Virtual Machine Introspection (VMI). The signature matching approach requires maintenance of the signature database of all known attack patterns and fails when new ones occur. Machine learning approach require a profile of monitoring machine for its normal behavior to detect unseen attacks based on similarity in behavior with learning attack patterns. Dynamic analysis technique analyses the VM programs at the run time, so it is very useful to detect the novel attacks. Dynamic analysis can be bypassed by the intruder if it is deployed at VM. VMI performs memory introspection at hypervisor level for all the programs that are running in VM and handles program change. VMI techniques leverage VMM technology that lacks traditional IDS approach. In a case of a security attack, the VMI helps in protecting the VM monitoring tool from getting compromised. The VM monitoring tool is placed at VMM level and the VM state information gathered at the VMM level is used to infer the behavior of guest VM. Hypervisor uses virtualization platform specific API's like XenCtrl library for Xen hypervisor.

The prime objective of the paper is to propose a robust VMI-based hybrid intrusion detection approach, deployed at the hypervisor layer which performs the process analysis and network traffic analysis to detect the anomalous behavior

978-1-5090-6367-3/17/$31.00 ©2017 IEEE

at Virtualization layer in the cloud environment. The key contributions of this paper are as below:

- To propose a Hybrid model of integrating information and evidence from system call sequences and the traffic analysis on the virtual network bridge.
- A prototype implementation of the proposed security architecture to detect inter-VM attacks such as flooding and port scanning.

This paper is arranged into six sections. Section 2 provides the extensive summary of related work done on Intrusion Detection in the cloud environment. Section 3 describes the background detail. Section 4 describes proposed security architecture, algorithm, and relevant description. Discussion for experiment set-up, the proposed technique analysis, and results are in section 5. Section 6 ends with the conclusion.

II. Related Work

Here we discuss some major related research work on intrusion detection in virtualization environment.

Roschke et al. [4] implemented the architecture of VM integrated IDS. IDS Management unit (IMU) comprises of several components and these parts are Event Database, Analysis Component, Event gatherer, and IDS Remote controller. The IMU is joint remotely to various IDS sensors installed in the VMs. IMU manages the IDS sensors from a remote location. These IDS sensors use a Signature matching approach for intrusion detection. This Signature matching approach requires regular maintenance of database so that it can detect the novel attacks.

Warrender et al. [5] proposed a method of simple enumeration of system call sequence delayed time embedding. Forrest [6] methods are used to generate the database of normal behavior. Later the test system calls are compared with the dataset of normal system calls. The observed mismatch found is called as an anomaly for every locality. They call this as a Locality Frame Count (LFC). If the LFC is higher than the threshold, was considered as a part of an anomaly.

Gupta et al. [7] proposed Immediate System Call Sequence (ISCS) for anomalous behavior detection by analyzing the system calls. It uses a key - value pair concept, where a key expresses the system call and the value indicates the immediate sequence of system calls. They developed the technique which leverages VM which makes it unsafe towards attacks on security tool. Though ISCS uses runtime behavior analysis, it is based on sequence matching technique, where the semantic behavior of the programs are not considered and are prone to false alarms even for casual system faults.

Mishra et al. [8] proposed NvCloudIDS, a cloud security architecture for detecting intrusions at virtualization and network layers in the cloud environment. The model provides two levels of detection mechanisms, at CNS and at VMM of the cloud infrastructure. Detection engine employs the VMI with machine learning technique. It utilizes system calls for malware detection and network features for network attack detection. Tupakula et al. [9] described Intrusion Detection Engine (IDE) used by a VMM based IDS, along with various elements such as Operating System Library Repository (OSLR), shared packet buffer, packet differentiator, and analyzer that makes the IDE well suited in a virtual environment. IDE compare each of the packets for earlier known signature patterns. Machine learning techniques on OSLR to update the information about VM are applied by Anomaly module.

Modi et al. [10] integrated signature based approach with anomaly detection in a cloud environment to improve the reliability and efficiency of security tools. However, the integration with the anomaly module does not minimize the false positives that Snort generated. Lin et al. [11] implemented Network IDS that deployed at the Dom0 of the Virtual Machine Manager. Their architecture configured NIDS rules for every VM based on services and the OS running in the VM. It requires regular maintenance as it imposes overhead at VMM.

Srinivasan et al. [12] implemented IDS which they deployed at VM to detect the anomalies. Their IDS is known as eCloudIDS that has two components named as uX-Engine and sX-Engine. The Cloud Instance monitor observes user specific actions and H-log-H subcomponent collect the logs. An audit log preprocessor is used to find the information from the log file and preprocess the information so that uX-Engine can use this information for the classification purpose. The accuracy and FPR achieved by the eCloudIDS are 89% and 9% respectively. IDS may be under control of exploited VM. Hofmeyr et al. [13] proposed N-gram approach which is based on the pattern matching method, maintains the short sequences of the monitored system calls of firm size. By invoking the unwanted function, any unexpected error, overflowing the buffer, a rare pattern can be generated that may not be sure about the suspicious behavior of a trace. IDS that do VM-level analysis for network traffic and processes are less robust and inefficient. An attacker can easily exploit the VM-level approach. Detection at higher privileged layer is more trustworthy. So, we need a more efficient VMM based hybrid approach for process analysis and network traffic analysis.

III. Background

Let us consider the basic background concepts in a brief. Xen supports fully virtualized HVM guests. That means the VM running on top of the hypervisor is not aware of sharing processing time with other clients on the same hardware. The host should have the ability to imitate underlying hardware for each of its guest machines. This type of virtualization enables to run an OS directly on a VM without any alteration leaving an image as if it was run on the bare hardware. The influence of this is that HVMs uses hardware extension that provides significantly fast access to underlying hardware on host system. Xend tool-stack configures the host network stack. Figure 1 shows the host configuration with Bridge networking virbr0 and has shared a physical device. The guest machine can attach to shared device and have full LAN access. When a driver becomes available, it allows the guest to transit smoothly to the PV device, from the emulated device. The default configuration of Xen employs bridging at the back-end

domain, Dom0 which makes all domains appear as individual hosts on the network. A software bridge created in the back-end domain. To establish connectivity of the host, the back-end virtual network devices (vifDOMID.DEVID) are added to the bridge along with a physical Ethernet device.

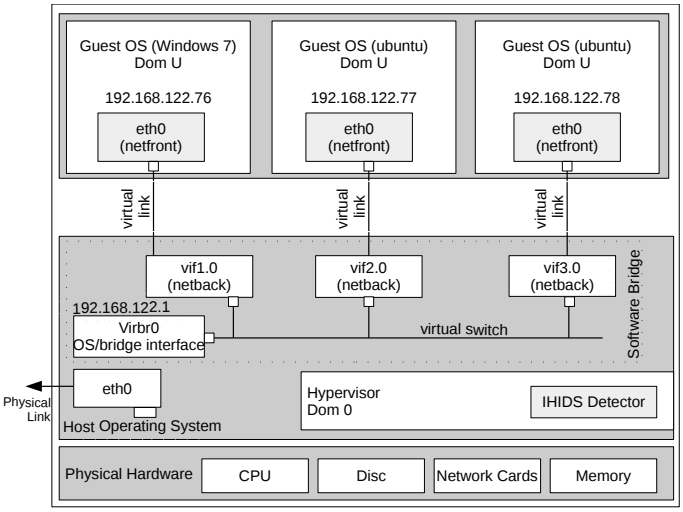

Fig. 1. Placements of proposed Model of IHIDS in Cloud

The physical device remains eth0 while the bridge is named virbr0. A virtual device is attached to an appropriate bridge at the time of configuring the VM. XL tool-stack is used to set up the VM. Data modification, identity spoofing, password based attack, port scanning and denial-of-service are common network attacks. Port Scanning is among the most popular techniques of reconnaissance that attackers practice to identify services that they can exploit. A potential target VM runs several services that listen to renowned ports. Through scanning the ports available on the potential target, the malicious tenant user finds potential weaknesses that can be exploited. FIN scan, XMAS scan, SYN scan, TCP-ACK scan attacks are considered. Another category of attack discussed here is Flooding attack. Attack definition and description can be found in Pilli et al. [14]. Flooding is a variety of Denial-of-Service attack which can bring down a service or network by means of flooding it with enormous amounts of traffic. When a service or network is overburden with extensive amount of packets initiated connection requests that are incomplete, and it cannot process any further legitimate requests for connection. We considered SYN Flood and UDP Flood in this paper.

IV. IHID Security Design

This section outlines the proposed **I**ntrospection-based **H**ybrid approach for **I**ntrusion **D**etection **S**ystem in cloud Environment (IHIDS). We first provide an overview of proposed security architecture with deployment scenario in the cloud. This security approach is mainly designed to detect inter-VM attack where one VM of a cloud server, attack another tenant VM on the same cloud server. The proposed detection mechanism detects attacks on the virtual network by performing two different methods. First is virtual network traffic monitoring. Second is the virtual memory introspection of the target VM. The subsequent section explains how we construct new dataset comprises of information and evidence from network behavior capture and system-wide process monitoring of target VM. The detection mechanism is based on two parallel operations. A correlation of attack with the network features for detecting various flood and port scan attacks. Second, validating the system calls frequency with the significant amount of baseline behavior captured earlier for normal operations of the target VM. The baseline dataset helps in determining the anomaly in the process as well as system faults. Figure 2 depicts the detection components of IHIDS approach. It comprises of four main elements: virtual Network Monitoring, Process Monitoring, Alert and Log Generation. The virtual network monitoring consists of three subcomponents: 1) Network Packet Capturing, 2) Packet Pre-processing, 3) Network Packet Detection Engine (nDetect). The Process Monitoring consists of three subcomponents: 1) SystemCall Tracing, 2) Trace Pre-processing, 3) System Call Behavior Detection Engine (sDetect). Each subcomponent is deployed as described below:

A. Network Packet Capturing

This sub-component captures live malicious network traffic at the higher privileged domain Dom0 of the hypervisor. It captures the network packets coming and going through the live VM. Packet sniffer wireshark[1] is deployed and configured at hypervisor layer to capture packets at virtual bridge virbr0. This bridge connects all the VMs to one another. Virtual interface vif1.0 is an interface for back-end traffic in the form of vifDOMAINID: DEVID of the VM. Filters are set to capture all the traffic coming and going through this bridge for target VM. Packets are stored in .pcapng files. Separate file is created for recording the traffic profile during all the attack scenarios.

B. Packet Pre-processing

Network traffic capture files (.pcapng files), holds the network statistics in form of packets. These data are further processed to extract the packet header information. Network packet header information is useful to understand the type of attack. tshark is used to extract vital information such as IP.src, IP.dst, tcp.port, IP.proto, tcp.flags, eth.src, eth.dst etc. Relevant information are extracted from .pcapng files and stored in the form of database table.

C. Network Detection Engine (nDetector)

This sub-module deploys stateful protocol analysis to detect the attack by investigating the virtual network packets information stored in database table. Attack detection queries are executed to detect the attack. The detection query is based on the attack correlation with the network packet protocol as mentioned in Table I. These queries are applied to the database to identify the type of attack. After identifying the type of attack, a relevant alert is generated to Cloud Admin and relevant log entry is made. Algorithm 1 describes the necessary steps for virtual network analysis.

[1]https://www.wireshark.org/.

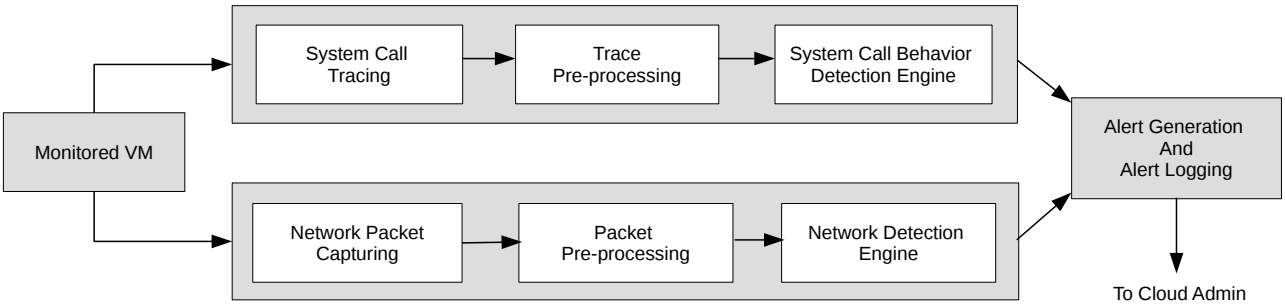

Fig. 2. IHID Component Diagram

D. System Call Tracing

Drakvuf [16], an advanced VMI tool has been installed on the VMM layer to collect system-wide behavior log generated by the processes running on the tenant VM. It uses break-point injection mechanism which traps the live VM execution state. Break-points are the op-codes inserted to every kernel function that will be trapped. A modified version of hypervisor is installed which incorporates built-in XSM policy. It enables to define permissible interactions between the hypervisor, domains and the related resources like memory and devices. XSM enables Xen to implement mandatory access control. Figure 3 and Figure 4 shows the sample behavior of system in form of ordered sequence of system calls on target VMs during the Flooding attack and Port Scanning attack respectively. In Figure 3, the victim machine is Windows VM. However, in Figure 4, the victim machine is Ubuntu VM. As OS running in both the VMs is different, the system behavior is represented by sequence of OS specific system calls. Hence, data is collected, examined and analyzed separately for each VM, discussed in Section V.

E. Trace Pre-processing

VMI tool generates a long list of system calls. This list is reduced by converting it into sequences of numbers; each number represents a system call. This trail of the numbers is stored in an appropriate .csv file for further processing by the detection module explained hereafter.

TABLE I
PROTOCOL FEATURE CORRELATION WITH ATTACK

Attack Type	Protocol Field(s) to Examine
Flood Attack	
TCP-SYN Flood	SYN Flag set in Source and ACK flag set in destination addresses
UDP Flood	Large number of UDP packets with same no of ICMP messages
Port Scan Attack	
TCP-SYN Scan	Packets with SYN flag and no corresponding ACK flag
FIN Scan	FIN flag and sequence number
XMAS Scan	URG,PUSH and FIN flags set
TCP-Connect Scan	Huge no. of failed connects and port requests in sequential manner
ACK Scan	ACK flags and RST as replies

F. SystemCall Behaviour Detection Engine (sDetector)

This sub-module deploys statistical anomaly-based detection method to detect the attack by finding the mismatch

```
[SYSCALL] vCPU:0 CR3:0x89166000,svchost.exe SessionID:0 ntoskrnl.exe!NtSetEvent Arguments: 2
[SYSCALL] vCPU:0 CR3:0x89166000,svchost.exe SessionID:0 ntoskrnl.exe!NtAlpcSendWaitReceivePort
Arguments: 8
[SYSCALL] vCPU:0 CR3:0x89e58000,svchost.exe SessionID:0 ntoskrnl.exe!NtAlpcSendWaitReceivePort
Arguments: 8
SYSCALL] vCPU:0 CR3:0x89e58000,svchost.exe SessionID:0 ntoskrnl.exe!NtSetInformationWorkerFactory
Arguments: 4
[SYSCALL] vCPU:0 CR3:0x89e58000,svchost.exe SessionID:0 ntoskrnl.exe!NtAlpcQueryInformationMessage
Arguments: 6
[SYSCALL] vCPU:0 CR3:0x89e58000,svchost.exe SessionID:0 ntoskrnl.exe!NtSetTimer Arguments: 7
[SYSCALL] vCPU:0 CR3:0x89e58000,svchost.exe SessionID:0 ntoskrnl.exe!NtSetEvent Arguments: 2
SYSCALL] vCPU:0 CR3:0x89e58000,svchost.exe SessionID:0 ntoskrnl.exe!NtSetEvent Arguments: 2
[SYSCALL] vCPU:0 CR3:0x89e58000,svchost.exe SessionID:0 ntoskrnl.exe!NtClose Arguments: 1
```

Fig. 3. System calls observed during Flooding Attack on Windows VM

```
[SYSCALL] vCPU:0 CR3:0x787af000,unity-panel-ser UID:1000 linux!sys_recvmsg
[SYSCALL] vCPU:0 CR3:0x75dc3000,gmain UID:1000 linux!sys_clock_gettime
[SYSCALL] vCPU:0 CR3:0x75dc3000,gmain UID:1000 linux!sys_poll
[SYSCALL] vCPU:0 CR3:0x776cb000,compiz UID:1000 linux!sys_clock_gettime
[SYSCALL] vCPU:0 CR3:0x75200000,dbus-daemon UID:1000 linux!sys_recvmsg
[SYSCALL] vCPU:0 CR3:0x776cb000,compiz UID:1000 linux!sys_read
[SYSCALL] vCPU:0 CR3:0x776cb000,compiz UID:1000 linux!sys_clock_gettime
[SYSCALL] vCPU:0 CR3:0x76ea7000,rs:main Q:Reg UID:101 linux!sys_gettimeofday
```

Fig. 4. System calls observed during the Scanning Attack on ubuntu VM

in system call occurences examined during the incidence of attack. In normal execution situation, if any mismatch is observed with respect to the baseline behavior and this mismatch crosses the predetermined threshold value, the system call is pronounced as suspicious. Threshold value is proportional to average frequency of system call derived from significant amount of trace collected during baseline dataset. This sub-module detects the incidence of mismatch in process behavior along with the name of system call, abnormal frequency observed, and threshold for that system call. Abnormal system calls detected by sDetector during port scanning attack on Linux VM are shown in Figure 6. Algorithm 2 describes the necessary steps for VM process analysis.

G. Alert Generation and Alert Logging

During the scenario of normal execution of the tenant VM, when any mismatch observed touching the threshold set for system calls in the dataset, the VM activity is marked suspicious. All incidences of mismatch are recorded to build the proof of attack. An alert is generated to Cloud Admin with the information of suspicious system calls detected by sDetector with the preceding output logs of nDetector. Cloud Admin may take appropriate action to drop such connection by isolating the malicious tenant VM or terminate the suspicious processes depend on the severity of collected alert logs. Notations used for algorithms are as below:

- nCapture: Capture the network traffic from virtual bridge using wireshark
- sCapture: Capture the system calls generated by monitored VM using VMI.
- nModule : Store the network packets in database after processing
- sModule : Pre-process and stores the system call information in file.
- threshold: threshold value to detect abnormal occurrence of system calls in determined time window

Algorithm 1 : Network Packet Analysis

```
1: procedure VIRTUAL NETWORK MONITORING
2: Cloud Admin→ Executables to generate network packet database
3:     for Every VM to be monitored at VMM do
4:         nCapture→Create (vmPacket.pcapng) temp file to path/..
5:         Extract Packet header info→ in tbl-VMPacket in database
6:         Execute Query on tbl-VMPacket using Table II info→Result
7:         if (Result>Threshold) then
8:           Alarm(Raise(AttackType,DomainID,AttackerDomainID)→CA
9:           Result→ Logfile
10:        end if
11:        Remove temp file from path/..
12:        Remove data from tbl-VMPacket
13:    end for
14: end procedure
```

Algorithm 2 : System Call Analysis

```
1: procedure VM PROCESS MONITORING
2:     for Every VM to be monitored at VMM do
3:         CA→ Executable generate System Call Trace for VM
4:         sCapture→Create(VMSysCall.txt) temporary file to path/...
5:         CALL Proc-SysCallMapping(VMSysCall.txt, OSType)
6:         sDetect:Frequency of each syscall in VMSysCall.csv→Result
7:         if (Result>Threshold) then
8:           Alarm(Raise(SysCall,Frequency,Process Name)→CA
9:           Result→ Logfile
10:        end if
11:        Remove temp file from path/..
12:        Remove data from tbl-VMPacket
13:    end for
14: end procedure
1: procedure PROC-SYSCALLMAPPING(VMSysCall txt, OSType)
2:      input:List of System call and Native system
3:      service routine (NTSysCallList.csv, LinuxSysCallList.csv)
4:    if (OSType = Windows) then
5:        MappingList = NTSysCallList.csv
6:    end if
7:    if (OSType = Linux) then
8:        MappingList= LinuxSysCallList.csv
9:    end if
10:   for Each SysCall in file VMSysCall.txt do
11:         sModule:Convert VMSysCall.txt→ Sequence of System Calls
12:       store to(VMSysCall.csv)
13:   end for
14: end procedure
```

Among the techniques proposed till now for behavior analysis of process quantifies mismatches or anomalous position that appear in the traces related to the normal database. It calculates total no. of mismatches in given length of the trace, then computes the maximum no. of mismatches the traces could have generated and shown the actual mismatches as a proportion of that maximum. If traces would be of arbitrary length, the total mismatch percentages may not worth while in general. So a local measure is needed for online detection, which can reveal how many mismatches are present in a particular part of the trace, and limits the amount of information we have to retain at any instance. For this purpose, we have taken traces of fixed length of time at different intervals to generate the normal behavior. As a preparatory to develop an appropriate measure, we calculated the mismatches found in fixed-length of trace, we refer it as attack window. This attack window is identical to the windows we used for virtual network packet capturing. Once the mismatches are counted they are validated against the predetermined threshold values to generate the alert to Cloud Admin. Figure 5 shows the alerts generated for flood attack on Windows VM while Figure 6 shows the alerts generated through sDetector sub module for port scanning attack on Linux VM. Cloud Admin can put limit on the rate of packet or drop packet or detach the VM from the cloud.

V. EXPERIMENT AND RESULTS

An archetype of present approach has been implemented on a system having ubuntu 15.10 as host OS with 16 GB RAM, 1 TB HDD and hypervisor Xen 4.8. Hypervisor is hosting three tenant VMs: Windows 7 with 3GB RAM and 150 GB disk, ubuntu 14.04 with 2.5 GB RAM and 200 GB disk, ubuntu 14.04 with 2 GB RAM and 150 GB disk. The open source Xen 4.8 [15] is considered for Dom0 implementation which manages all unprivileged domain DomU of the hypervisor. This architecture provides security detection checks for network attacks like Flooding attack: TCP-SYN flooding, UDP flooding and Port Scanning attack: TCP-ACK scan, SYN scan, FIN scan and XMAS scan. The main objective is as below:

Network monitoring and VM process monitoring was performed for various span of time during one week to observe the normal behaviour of the tenant VM. A well known port scanner, nmap[2] and for flooding attack hping3[3] utility has been installed and used by the attacker VM. We collected network traffic for various programs running on the live VM for fix duration of time. We also collected the system calls generated by VMs for various lengths of time to consider the normal behavior of system calls.

Drakvuf [16], an advanced VMI tool is used to collect system calls generated by the processes running on the tenant VM. A little modified version of hypervisor has been installed which incorporates built-in XSM policy. It enables to define permissible interactions between the hypervisor, domains and the related resources like memory and devices. XSM enables Xen to implement mandatory access control.

To evaluate the attack detection functionality, we sent large number of flood packets to target VM on open port with source spoofed with fake IP address. Port scan helped to determine the available open ports on the target VM. We collected network packet traces. These packets are pre-processed and vital header information are extracted and stored in the form of database table. As explained in the previous section, using Table III information, queries are formed and executed to detect the attack packets.

[2]https://www.nmap.org/.

[3]https://www.hping3.org/.

System calls generated during various attacks have been stored in separate files. These files are pre-processed as mentioned earlier. It results in a trail of numbers where each number represents a system call. For each of these system calls the frequency of occurrence is stored as baseline behavior.

A. DataSet Generation

Significant amount of system traces are collected at different times from the guest VM. Around 6020000 system calls traces from linux VM and around 173000 traces from windows VM to determine the normal behaviour of the VMs. No major changes are made in guest VM during this period of trace collection. A new dataset has been created which consist of all the system calls generated system-wide during the period of two weeks. Target VM has various processes running on open ports. The ports have been filtered for certain traffic but all. Figure 5 shows the sample trace collected during flooding attack on VM running the windows 7 host operating system. Since the Windows Native System Services routine related to the system calls can behave differently in the way that they handle and interpret input parameters, these parameters are not considered for analysis. The system call name is substituted with a unique number [18], This reduces the trace into sequence of numbers. Using this sequence, a dataset of baseline behaviour is built for tenant VM which represents its normal or expected behaviour. For each of the system call in the trace, the dataset stores normal occurrences of the system call. Time duration for process monitoring is other key parameters for this method as the detection mechanism depends on this time frame and the threshold value. The threshold is set as a range of (20% to 50%) of normal frequency of a system call. The baseline dataset is used it to monitor the system calls from processes and identify anomalies if system calls crosses the threshold. Code module has been developed for pre-processing and detection of both the network anomaly and system call anomaly which is explained in previous section along with Algorithms. Discussing one of the system call from Figure5.

TABLE II
ATTACK DATASET

Attack Type	No of Packets Generated	No of System Calls Generated
UDP Flood	2351440	3923
TCP-SYN Flood	1144985	9754
SYN Scan	8029	584
FIN Scan	10029	5526
XMAS Scan	10026	10691
TCP-ACK Scan	16070	16199

`NtWaitForSingleObject` is windows native system service routine, stored in ntdll.dll of the windows machine is invoked when a system call is triggered. As the figure shows, `NtWaitForSingleObject` outnumbers the baseline behaviour of the VM. When `NtWaitForSingleObject` is called from memory allocation/deallocation routines, it waits until the specified object attains a state of Signalled. If the operational time out is specified, actions are performed. Else, the current thread is put in a waiting state and selects a new thread for execution on the current processor. As this signal is raised with abnormal frequency which exceeds the defined threshold and these events are notified in form of logs to CA. Some of abnormal system calls found for various flooding and port scanning attack are shown in Figure 5 and Figure 6 respectively. Figure 6 shows system call `sys_socket` routine which is invoked to create an endpoint for communication, returns a file descriptor that refers to that point of communication. The arguments to the function call, indicates a communication domain and chooses the protocol family, to be used for communication. They are defined in sys/socket.h in the system.

Alert!NtDeviceIoControlFile	246 times instead of (25-32)
Alert!NtWaitForSingleObject	143 times instead of (40-68)
Alert!NtAlpcSendWaitReceivePort	196 times instead of (59-74)
Alert!NtReleaseWorkerFactoryWorker	52 times instead of (13-17)
Alert!NtRequestPort	6 times instead of (2-3)

Fig. 5. Abnormal Sys Calls Detected during Flood Attack on WindowVM

Alert!sys_socket	56 times instead of (30-38)
A lert!sys_connect	150 times instead of (55-85)
Alert!sys_getpeername	174 times instead of (51-83)
Alert!sys_sendmsg	93 times instead of (67-84)
Alert!sys_sendto	72 times instead of (41-51)
Alert!sys_socketpair	64 times instead of (25-51)

Fig. 6. Abnormal System Calls Detected during Port Scanning on Linux VM

B. Results And Discussion

Proposed architecture is deployed at virtualization layer which monitors VM processes and virtual network traffic between tenant VMs. Process monitoring was performed for various span of time during one week to observe the normal behaviour of the tenant VM. To demonstrate inter-VM attack, Flooding and port scanning were performed by the attacker VM. Network traffic is collected for a live VM for fix duration of time. Security module nDetector is able to efficiently detect the attacks with highest 98.08% of accuracy. We also collected the system calls traces generated by VMs for same lengths of time. Abnormal behavior detected by sDetector is recorded in the log file as evidence of attack. Since the proposed architecture is deployed at Dom0 of VMM and pre-processing as well as detection mechanism works at Dom0, only the VMI activity (System Call Tracing) for target VM can be considered for any processing delay which a target VM could face. Other components such as Network Packet Capturing, Packet pre-processing, Network Packet Detection Engine, Trace Pre-processing and SystemCall Behavior Detection Engine cause negligible delay in VMM execution. Once the attack is identified, this architecture prevents the attack to build up and save the target VM from significant damages.$ Both the modules collectively generates alert to CA for further action. Detection accuracy of network detection engine for various attacks are shown in Table III along with FPR and no. of system calls detected as abnormal for each type of attack. As the observations indicate, the proposed approach

TABLE III
OBSERVATIONS

Attack Type	Detection Accuracy (%)	Fall-Out	No Of System Call Detected as Abnormal	Alert Raised
UDP Flood	98.08	5.22	39	Yes
TCP-SYN Flood	89.16	11.052	20	Yes
SYN Scan	91.04	9.1113	42	Yes
FIN Scan	93.1	8.051	15	Yes
XMAS Scan	95.87	2.051	35	Yes
TCP-ACK Scan	98.2	5.221	32	Yes

can accurately detect UDP flood with 98.08% accuracy, FIN Scan with 93.1% accuracy with FPR of 8.05%. TCP-ACK Scan and UDP Flood were detected with FPR of 5.2% while XMAS scan was detected with 95.87% and lowest FPR of 2.05% which indicates that the proposed security architecture is suitable for known attacks. Accuracy and FPR are calculated as following.

- **Accuracy**: Performance of the algorithm can be measured by Accuracy. It indicates the rate at which an algorithm can predict the positive or negative instance correctly. Formula to calculate accuracy is as following:

$$Accuracy = \frac{TN + TP}{TN + TP + FN + FP} \quad (1)$$

- **Fall-out**: Fall-out or False Positive Rate (FPR) indicates the probability, an IDS shows an alarm when there is no incidence of attack took place. It is calculated using below formula:

$$Fall - out = \frac{FP}{TN + FP} \quad (2)$$

VI. CONCLUSION AND FUTURE WORK

The paper presents a very efficient security architecture deployed at the virtualization layer, to detect inter-VM attack where the network traffic never passes through a real networking device. The detection mechanism proposed is perplexed by common network attacks such as flooding and port scanning. Virtual network traffic analysis at hypervisor layer and VM process behavior analysis using VMI at hypervisor level are applied to detect the abnormal behaviour of the VM during the incidence of the attack.

This framework is validated with newly generated dataset which was collected from instance of VM deployed in the private cloud set-up at our institute. This technique has low computation cost as it is deployed at the hypervisor level, and the processing of the VM is not hindered. This definition is compact on space and complexity as it handles all key information in either form of data table or small text files for each VM. It has high detection rate as mentioned in Table III for the known attack types.

Detection mechanism depends on the duration of trace collection and the threshold value which is proportional to average frequency of system call derived from significant amount of trace collected during baseline dataset. This technique gives good detection rate for known kind of attacks and we would like modify this technique to make it suitable for various classes of unknown attack. VM process monitoring depends on modified version of hypervisor and we would like to end the dependence on various versions. We are interested and are already working on a solution to deploy this detection system for other unmodified version of hypervisors.

REFERENCES

[1] CISCO, "Cisco annual security report ", [Online]. Avail-able:http://www.cisco.com/web/offer/gist_ty2_asset/Cisco_2014_ASR.pdf, 2014.

[2] M. Dekker, D. Liveri, and M. Lakka, "Cloud security incident reporting, framework for reporting about major cloud security incidents(ENISA)", December, 2013.

[3] S. Panjwani, S. Tan, K.M. Jarrin and M. Cukier, "An experimental evaluation to determine if port scans are precursors to an attack", in:*International Conference on Dependable Systems and Networks (DSN05)*, IEEE, Yokohama, Japan, 2005, pp. 602-611.

[4] S. Roschke, F. Cheng, and C. Meinel, "Intrusion detection in the cloud", in:*8th International Conference on Dependable, Autonomic and Secure*, Chengdu, China, 2009, pp. 729-734.

[5] C. Warrender, S. Forrest, and B. Pearlmutter, "Detecting intrusions using system calls: alternative data models", in:*IEEE SYMPOSIUM ON SECURITY AND PRIVACY*, Oakland, CA, USA, 1999, 133-145.

[6] S. Forrest, S.A. Hofmeyr and A. SoMayaji,"A sense of self for UNIX processes", *IEEE Symposium*, Oakland, CA, USA 1996.

[7] S. Gupta and P. Kumar, "An immediate system call sequence based approach for detecting malicious program executions in cloud", *Wireless Personal Communications*, vol. 81, no. 1, pp. 405–425, 2015.

[8] P. Mishra, E.S. Pilli, V. Varadharajan and U. Tupakula, "NvCloudIDS: A security architecture to detect intrusions at network and virtualization layer in cloud", in:*International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, IEEE, Jaipur, India, 2016, pp. 56-62.

[9] U. Tupakula, V. Varadharajan and N. Akku, "Intrusion detection techniques for infrastructure as a service cloud", in:*IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing IEEE*, Sydney, NSW, Australia, 2011, pp. 744-751.

[10] C. Modi, D. Patel, B. Borisanya, A. Patel, and M. Rajarajan, "A novel framework for intrusion detection in cloud", in:*5th International Conference on Security of Information and Networks*, ACM, Jaipur, India, 2012, pp. 67-74.

[11] Lin, Chih-Hung, Chin-Wei Tien, and Hsing-Kuo Pao, "Efficient and effective NIDS for cloud virtualization environment ", in: *IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom)*, Taipei, Taiwan, 2012, pp. 249-254.

[12] M. K. Srinivasan, K. Sarukesi, A. Keshava, and P. Revathy, "ecloudids tier-1 ux-engine subsystem design and implementation using self-organizing map (som) for secure cloud computing environment ", *International Conference on Security in Computer Networks and Distributed Systems*, Springer, Berlin Heidelberg, 2012, pp. 432-443.

[13] Hofmeyr, Steven A., Stephanie Forrest, and Anil Somayaji,"Intrusion detection using sequences of system calls", *Journal of computer security*, vol. 6, no. 3, 1998, pp. 151-180.

[14] Pilli Emmanuel S., Ramesh C. Joshi, and Rajdeep Niyogi. "Data reduction by identification and correlation of TCP/IP attack attributes for network forensics", in:*International Conference & Workshop on Emerging Trends in Technology*, ACM, Mumbai, Maharashtra, India, 2011, pp. 276-283.

[15] Xenproject.org, (2006), "Xen Hypervisor ", [Online].Available:https://www.xenproject.org/.

[16] Lengyel, Tamas K. and Maresca, Steve and Payne, Bryan D. and Webster, George D. and Vogl, Sebastian and Kiayias, Aggelos, "Scalability, fidelity and stealth in the DRAKVUF dynamic malware analysis system", in:*Proc. of the 30th Annual Computer Security Applications Conference*, New Orleans, Louisiana, USA, 2014, ACM, pp. 386-395.

[17] Payne, Bryan D, "Simplifying virtual machine introspection using libvmi", *Sandia report*,(2012), pp. 43-44.

[18] Microsoft, "NtXxx Routines"[Online]. Available:https://msdn.microsoft.com/en-us/library/windows/hardware/ff557720(v=vs.85).aspx/