A HIERARCHICAL INTRUSION DETECTION SYSTEM FOR CLOUDS: DESIGN AND EVALUATION

Hisham A. Kholidy [1,2,3], Fabrizio Baiardi [2] and Salim Hariri [1]

hkholidy@ece.arizona.edu, hisham_dev@yahoo.com baiardi@di.unipi.it
hariri@ece.arizona.edu

Esraa M. Elhariri [1,3], Ahmed M. Yousof [1] and Sahar A. Shehata [1]

esraaelhariri@email.arizona.edu, amyousof@email.arizona.edu
saharabdelfattah@email.arizona.edu

1- NSF Cloud and Autonomic Computing Center, College of Electrical and Computer Engineering, University of Arizona, USA.
2- Dipartimento di Informatica, Università di Pisa, Pisa, Italy
3- Faculty of Computers and Information, Fayoum University, Fayoum, Egypt

ABSTRACT

Security and availability are critical for cloud environments because their massive amount of resources simplifies several attacks to cloud services. This paper introduces a distributed deployment and a centralized one for our Cloud intrusion detection framework, CIDS-VERT. After describing the architectures and the components of the two deployments it describes the experimental results that confirm that the deployments overcome some limitation of current IDSs to detect host, network and DDoS attacks. Lastly, we discuss the integration and the correlation of the host and network IDSs alerts to build a summarized attack report.

KEY WORDS

Cloud Computing, Security, Intrusion Detection, Attacks, DDoS.

1. INTRODUCTION

Cloud computing is a large-scale distributed computing paradigm where a pool of virtualized, dynamically-scalable, managed computing services are delivered on demand over the Internet. The adoption of this paradigm is delayed because of concerns on host and network attacks that can violate the integrity and confidentiality of data in the cloud. Furthermore, attackers can exploit the large amount of resources in a cloud for their advantage. As an example, they can implement a DDoS to block the access of legal users to clouds.

This paper introduces two deployment models, the Distributed and the Centralized one, for our cloud intrusion detection framework (CIDS-VERT) [1] that deals with host, networks and DDoS attacks. We also discuss the integration and correlation of alerts of host and network IDSs of CIDS-VERT to produce a single attack report. The remainder of this paper is organized as follows. Section 2 briefly highlights some preliminaries such as host, network, and DDOS attacks and the software libraries to simplify these attacks, the protocol to integrate host and network IDS alerts. Section 3 describes the hierarchical architecture of the proposed cloud IDS. Section 4 introduces the Distributed and Centralized deployment models and outlines their relative advantages. Section 5 discusses the approaches to integrate, correlate, and summarize distinct alerts from host and network IDSs. Section 6 shows some experimental results and evaluates the accuracy of the proposed deployments. Finally, Section 7 draws some conclusion remarks and outlines future work.

2. PRELIMINARIES

2.1 Host, Network, and DDoS Attacks

Attacks utilize network media and manipulate computing and/or network resources to severely degrade the performance of the services and eventually shutdown the entire network. We can classify attacks according to the type of penetration (inside, outside), type of interactions (passive, active) and the mechanism to launch the attack. [2, 3]

Penetration Type: Penetration can be carried out as an outsider or as an insider. Insiders are legitimate system users that are conducting malicious activities through their accounts or illegally using other user accounts. Instead, an outsider launches attacks from outside the network perimeter or implements probing or scanning attacks to acquire information on the target network to launch the real attacks. Potential outsiders range from amateur to organized crime, cyber terrorists, and hostile governments.

Interaction Type: Attack classification should also consider the interaction between the attackers and the network environment. Based on this criterion, network attacks can be either classified as active attacks or passive attacks. In a passive attack (e.g., wiretapping, port scanner, idle scan), the attacker listens to the streams of traffic to gather valuable information. Thus the anomalous behaviors caused by attacks of this type are hard to observe since they leave the minimum footprint. Active attacks aims to change the configuration of system resources or affect their operation (e.g., Denial of Service Attacks, Spoofing, Man-in-middle attack, ARP positioning). They trigger an anomalous behaviors that can be observed and quantified provided that the appropriate metrics are used.

Mechanism Type: the mechanisms and techniques to launch an attack partition attack into five classes: Denial of Service (DoS), User to Root (U2R), Remote to Local, probing, and virus/worm attacks.

Denial of Service (DoS) attack: This attack prevents services for the users by limiting or denying their access to network resources such as bandwidth, memory, buffers, and/or processing power. To this purpose, these attacks can target software vulnerabilities, change configuration, or exhaust the network resource to its limit. Possible examples include ICMP Nukes, Teardrop, Land Attack, the ping of death, and playing with the configuration of a compromised router. While these attacks can be easily fixed by installing software patches, reloading correct configuration, and limit the access to resources they impose a critical load on network administrators that increases with the number of these attacks. We describe a popular attack in this class, the Distributed Denial of Service (DDoS), in Section 2.1.2.

User to Root (U2R) attack: Attackers with login access can bypass authentication to gain the higher privileges of another user in controlling and accessing the system.

Remote to Local (R2L) attack: Attackers can bypass normal authentication and execute commands and programs on the target with local machine privileges.

Probe/Scanning attacks: These attacks blueprint the network and its resources to discover vulnerability or entry points that the attacker can use to penetrate or attack network resources.

Worm/virus: This attack is run by a malicious piece of code that spread across a network and target hosts or network resources to cause dysfunction, data loss, or data theft.

In this paper we will mainly focus on the detection of the DDoS attacks and both the host and network ones.

2.1.1 Host and Network Attacks

Attacks can also be classified into network or host ones according to the type of vulnerabilities they exploit. Network attacks exploit vulnerabilities in the communication protocols or in the underlying interconnection structure to attack network communications. As an example, if some communications adopt an unsecured or clear text format, an attacker that can access network data paths to read and interpreter transmitted data. Some examples of these attacks are [2]:

1) Eavesdropping: it is also known as sniffing or snooping and it monitors the network traffic. A strong encryption services is required to protect the network data.
2) Data Modification: It modifies transmitted data in a way that cannot be detected by the sender or the receiver.
3) Identity or IP Address Spoofing: It constructs IP packets that appear to originate from valid addresses to modify, reroute, or delete some network data. It is supported by specialized libraries.
4) Denial-of-Service Attack (DoS): It shuts down applications or network services by flooding them by invalid traffic. This can block request of a legal user to access network resources.
5) Man-in-the-Middle Attack: It inserts a distinct entity between two communicating components to capture and modify their communications.

Host based attacks are enabled by vulnerabilities in the host operating system or in the applications such as missed checks on input size, input validation errors, race conditions, privilege-confusion bugs. Typical examples are [3]:

1) Buffer overflow: It violates memory safety by exploiting the lack of controls on the size of a parameter. A write in the attacker program overruns the boundary of a memory buffer to overwrite adjacent memory positions.
2) Rootkit: It installs software components to hide a malicious processes running on the node and that grants to the attacker a privileged access to the system.
3) Format string: This attack crashes a program or executes harmful code. It exploits the lack of control on some user inputs such as the format string parameter in some C functions, such as printf().

Several libraries have been developed to support the implementation of an exploit, e.g. a code fragment that automates, at least partially, an attack. As an example, Metasploit [4] is a consistent and reliable library of constantly updated exploits for distinct operating systems and applications as well as a complete environment to develop new tools to automate every aspect of a penetration test. It simplifies the development of attack vectors to extend its exploits, payloads, encoders to create and execute more advanced and specialized attacks.

2.1.2 DDoS Attacks

Distributed Denial of Service (DDoS) attacks [5] aims at disrupt the service quality of a system and are strongly related to clouds because their effectiveness increases if an attacker can use the massive amount of resources in a cloud. Figure 1 shows the four elements of DDoS attacks [6] namely:

1. The attacker machine.
2. The handlers: They run some malware and act as an intermediate interfaces to control the agents and route to them the attacker commands. The attacker controls these hosts as a result of a previous attack.
3. The agents or zombie hosts: also these hosts are controlled by the attacker. They run some malware that either implements an attack on behalf of the attacker (botnets) or generates a stream of packets towards the target system.
4. The victim or target system.

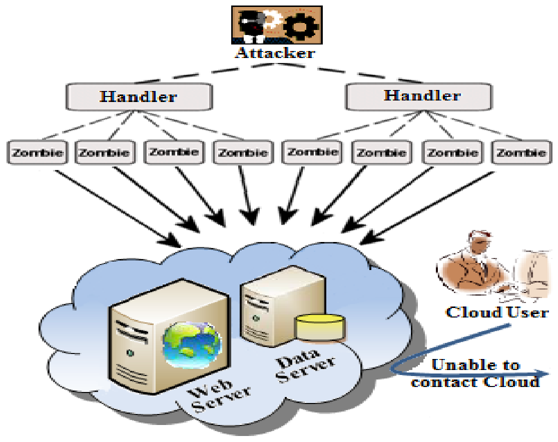


Figure.1: The DDoS Strategy

The implementation of any kind of DDoS attack includes the following stages:

(1) Search of vulnerable hosts to act as handlers and zombies. The attacker scans hosts in various networks and can exploit a standard vulnerability scanner such as Nessus [8].
(2) Compromising the vulnerable hosts: The attacker exploits the vulnerabilities returned by the scanner to attack some hosts and stealthy install some malware.
(3) Communication, Broadcasting, and flooding: The attacker communicates to one or more handlers to manage and configure the zombies or issue attack commands. The handler broadcasts any received commands to hundreds or even thousands of zombies that start flooding the network of the target system until the attacker stops the attack.

In our experiments, DDoS attacks are implemented through both the LOIC library [7] and the CPU Death Ping library [9]. LOIC is a powerful free DOS and DDOS attacking tool, it attempts to open several connections to the same target host and continuously floods this host with fake network packets, or with HTTP requests that lead to a service disruption. A DDOS attack runs LOIC through multiple zombies. LOIC supports two modes of operation: the manual mode, where the user manually fills any input parameter, and the automatic mode, where the attacks are remotely controlled. The CPU Death Ping library is a DDoS attacking tool that opens multiple floods to a large number of hosts and continuously floods them with fake packets and HTTP requests to reduce their bandwidth and their performance.

2.2 Cloud IDSs: State of the Art

We briefly review some recent proposal on intrusion detection for clouds. [10] proposes an IDS based on mobile agents (MAs). Its most important weaknesses are the performance and the security issues related to MAs. [11] proposes a theoretical framework to deal with attacks targeting any cloud service model but it does not correlate the alerts from components in the cloud infrastructure. [12] investigates the effect of DDoS on a cloud system and proposes an IDS based on the behavioral threshold that signals an attack if the requests of a legal user are outside the normal user range. The threshold is automatically determined as a dynamic variable based upon the network position and pressure traffic. To simplify the discovery of legal users, several solutions may be integrated with the IDS such as load balancing of the network traffic and a honeypot [13]. The latter analyzes the collected data to discover the attacker signatures. The IDS does not correlate network events in distinct virtual zones of the cloud. Furthermore, no deployment in a real cloud system is described and no evaluation is reported on the accuracy or the performance of the IDS. [14] uses an IDS sensor such as the version of Snort [15] installed on VMware virtual ESX [16] machine that sniffs in-bound traffic. Snort matches in bound packets against several intrusion patterns. If a match occurs, all the traffic from the same IP address is dropped. No evaluation of the accuracy and performance of this solution is presented. Furthermore, network events are not correlated to discover attacks against several virtual zones. [17] proposes a cooperative IDS that reduces the impact of DoS attack in each cloud region. Several IDS components are distributed across these regions and a cooperative module receives their alerts. Then, a majority vote determines the trustworthiness of these messages. This system avoids any single point of failure but its accuracy is not satisfactory. Furthermore, it has not been evaluated against a DDoS attack.

The analysis of current proposals confirms that a defense strategy for cloud systems introduces some further requirements with respect to traditional ones. To be adopted in clouds, an IDS should: (1) be distributed and scalable, (2) avoid single points of failure, (3) correlate the user behaviours in distinct environments, and (4) integrate different service models.

2.3 Intrusion Detection Message Exchange Format (IDMEF)

The IDMEF [18] is an XML standard format for messages exchanged among IDSs. Its data model is an object-oriented representation of the alert data that the intrusion detection analyzers exchange with the management system. It provides a standard, coherent representation of alerts and it describes the relationship between simple and complex alerts. As shown in Figure 2, IDMEF Message is the top-level class and it has two subclasses that define the message type namely, Alerts and Heartbeat. A heartbeat message signals the current status of the IDS analyzer to the central manager or the other way around. Heartbeats are sent with a predefined frequency, e.g., every 10 minutes. The absence of a Heartbeat message indicates that the analyzer or its network connection has failed. The Alert message is the response message from an IDS analyzer and its information is used to integrate and correlate the alerts from different IDSs. The integration is based on the similarity of one or more of the data model subclasses as following:

(a) Attack name or signature given by the classification subclass.
(b) Times of creation and of analysis. These two times depend upon the characteristics of the firing IDS. Hence, two alerts might be considered similar even though their times of creation and of analysis are completely different.

(c) Source and target. The structure of the source and target subclasses are similar; they might be described by a node IP address or host name, a user name, a process name and services.

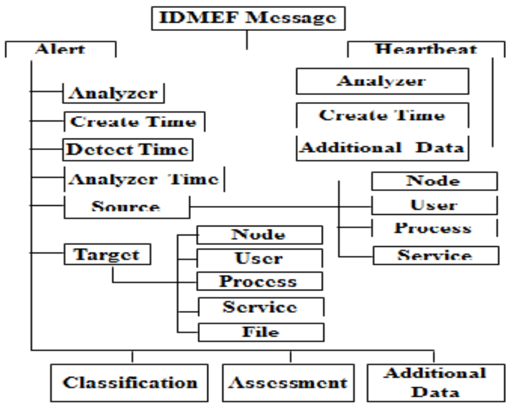


Figure 2: IDMEF Data model

3. THE HIERARCHICAL ARCHITECTURE OF OUR CLOUD IDS

This section briefly outlines the hierarchical structure of the proposed Cloud IDS as well as its implementation models. The proposed Cloud IDS integrates knowledge techniques and behavior based ones to detect and stop attacks of distinct classes, e.g. masquerade, host, network, and DDoS, against all cloud models. To cover any cloud model, we have defined and implemented two implementation models of the IDS namely, CIDS [19] and CIDS-VERT [1]. CIDS applies the Independent detection model [1] and works efficiently with small and private cloud networks. Instead, CIDS-VERT applies the Centralized-Backup detection model [1] to be more scalable and to efficiently protect large clouds such as public and hybrid ones. With reference to the CIDS-VERT model, see Figure 3, this paper evaluates the detection accuracy of host, network and DDoS attacks using two deployment models, Centralized and Distributed.

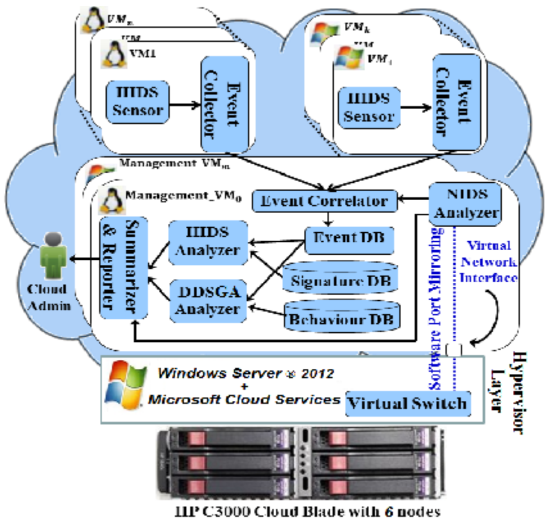


Figure.3: CIDS-VERT architecture

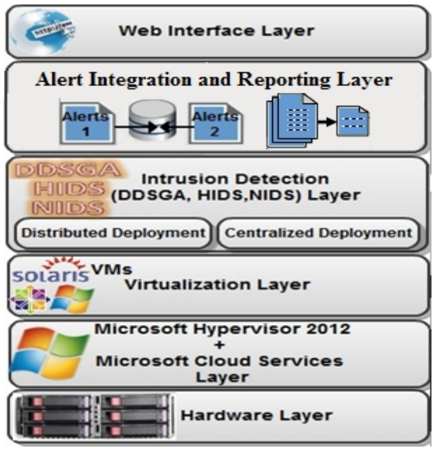

Web Interface Layer
Alert Integration and Reporting Layer
Alerts 1
Alerts 2
DDSGA
HIDS
NIDS
Intrusion Detection (DDSGA, HIDS,NIDS) Layer
Distributed Deployment
Centralized Deployment
VMs
Virtualization Layer
Microsoft Hypervisor 2012 + Microsoft Cloud Services Layer
Hardware Layer

Figure.4: The hierarchical architecture of the proposed cloud IDS

As shown in Figure 4, our Cloud IDS has a hierarchical architecture with six layers.

1) Infrastructure Layer:

It is the lowest level that defines the physical specifications of the cloud testbed. In the considered case, the testbed consists of an HP C3000 Cloud blade with six nodes. One node, the head one, works as a front side interface for the blade and has a Quad core 2.3 GHz CPUs, 4 GB RAM, 80 GB Hard drive, and a SmartArray P400 Controller for Storage Connect. Each of the remaining five computing nodes consists of : Quad core 2.8 GHz CPUs, 16 GB RAM, 80 GB Hard drive, and a Gigabit Ethernet. The head node runs Microsoft GUI windows server 2012 with Microsoft cloud services and Microsoft Hypervisor manager 2012, while each other node runs Microsoft core windows server 2012. The testbed also includes a 24 port Procurve Switch (10/100/1000 ports) for data networks and another 24 port Procurve Switch (10/100 ports) for console management.

2) The Hypervisor and Cloud Service Layer:

Its components manage the virtual VMs, virtual switches, and virtual SAN driver. Furthermore, it provides system security functions such as Isolation, Inspection, and Interposition. Our cloud IDS can work with different frameworks, among them VMware cloud [16], Microsoft private cloud [20], Open Stack [21], Eucalyptus [22]. In the considered testbed, this layer contains several components such as Microsoft windows server 2012 with its Hypervisor and Microsoft cloud software and tools.

3) Virtualization Layer:

It defines the mapping of the VMs onto the physical cores. In the testbed, each node hosts 3 VMs that runs, respectively, Windows XP Professional SP3, UNIX (Solaris) and Linux (Centos) to provide a full heterogeneous environment. Each VM is mapped onto one core of the Quad core processor and 3 GB RAM. Each VM runs a HIDS sensor and event collector component to collect the events and logs from the VM operating system. The VM forwards events and logs to a centralized management VM that analyzes them through DDSGA [23] and HIDS analyzers. In some VMs, a NIDS component works as a sensor or a server. One VM runs the CPU Death Ping, LOIC, and the Metasploit libraries. Section 6.1 explains the attack scenarios in details.

4) Intrusion Detection Layer:

This is the layer where we deploy the main three IDS components: the HIDS, the NIDS and the Data Driven Semi-Global Alignment, DDSGA [23] we have developed to detect masquerade attacks. In the experiments, the HIDS is OSSEC [24], an open source HIDS, and the NIDS is Snort. This layer also defines the two deployment models, Centralized and Distributed, mentioned before. Section 4 outlines the deployment of all the IDS components in the VMs.

5) Alert Integration and Reporting Layer:

This layer integrates and correlates the alerts from host and network IDSs. To this purpose, it uses the IDMEF format. It also highlights, among the large number of integrated alerts, the few important ones with high risk to simplify the handling of attacks. Sections 5.1 and 5.2 detail the integration and correlation processes.

Layer 6: Web Interface Layer:

To offer a central location to admin and manage the IDS components, this layer runs in the management VM and handles the web pages to visualize the IDS charts and dashboards, see also Section 6.3.

4. THE DISTRIBUTED AND CENTRALIZED DEPLOYMENTS

To evaluate our Cloud IDS, we have partitioned the cloud infrastructure into two virtual zones, VZ_1 and VZ_2, to distribute the DDoS Zombies into two distinct virtual cloud networks. This increases the complexity of detecting the DDoS attack. In the virtual networks, the VMs are connected to virtual switches through virtual NIC cards, see Figure 5. Virtual switches are connected to physical switch ports through the port mirroring facilities provided by the Hypervisor layer. VZ_1 consists of two nodes, $node_0$ and $node_2$, each running three VMs with distinct operating systems. Another VM is mapped onto $node_2$ to run the Metasploit and LOIC attack libraries. VZ_2 consists of three nodes, $node_{1,}$ $node_3$, and $node_4$, each hosting 3 VMs as in $VZ_{1.}$ The following subsections explain the two proposed deployments and Section 6 outlines the experiments using the two deployment options.

4.1 The Distributed Deployment

As implied by its name, the distributed model aims to spread the detection overhead among several VMs in the cloud. Then, the final decision is taken by correlating the outputs of the IDS sensors in these VMs. For each virtual zone, HIDS and NIDS components are distributed among the corresponding VMs. The HIDS consists of two main components, an agent and a server. The agent is a sensor that collects all events from the VM operating system and forwards them to the server component in a specific VM in the zone. Agents are installed in all VMs except those with the HIDS servers. An HIDS server analyzes all collected events and exchanges its alerts with the server in the second zone so that the administrator can collect all the alerts from any server VM. The NIDS component works as a server that monitors the traffic through the virtual switch. It communicates its detection score to the NIDS server in the second zone that correlates the scores to take the final decision about the detection of network attacks. To avoid a single point of failure, the VM that hosts the HIDS and NIDS servers is backed up by another VM in a distinct node in the same zone. The backup VM is always updated with the status of the active server through the exchange of heartbeat messages. Hence, it acts as a hot spare to recover any failure in the active VM. Both the VM that runs the NIDS and HIDS servers and the corresponding backup VMs are referred as “management VMs” in the CIDS-VERT framework in Figure 3.

As shown in Figure 5, VM0, hosted in cloud $node_0$ of VZ_1, is the active management VM. It runs both the Snort and the OSSEC servers and it is backed up to VM6 in $node_2$. The Snort server is attached to a promiscuous port on the virtual switch to mirror all traffic. The OSSIC server is connected to all OSSEC agents in the other VMs. In the same way, VM4, hosted in $node_1$ of VZ_2, runs the OSSEC and Snort servers and is backed up by VM10 hosted in $node_2$. The signature databases in both VM0 and VM4 are simultaneously updated and the two VMs exchange the notification alarms. Furthermore, the detection scores of the Snort servers in both the VMs are correlated to compute the final detection score.

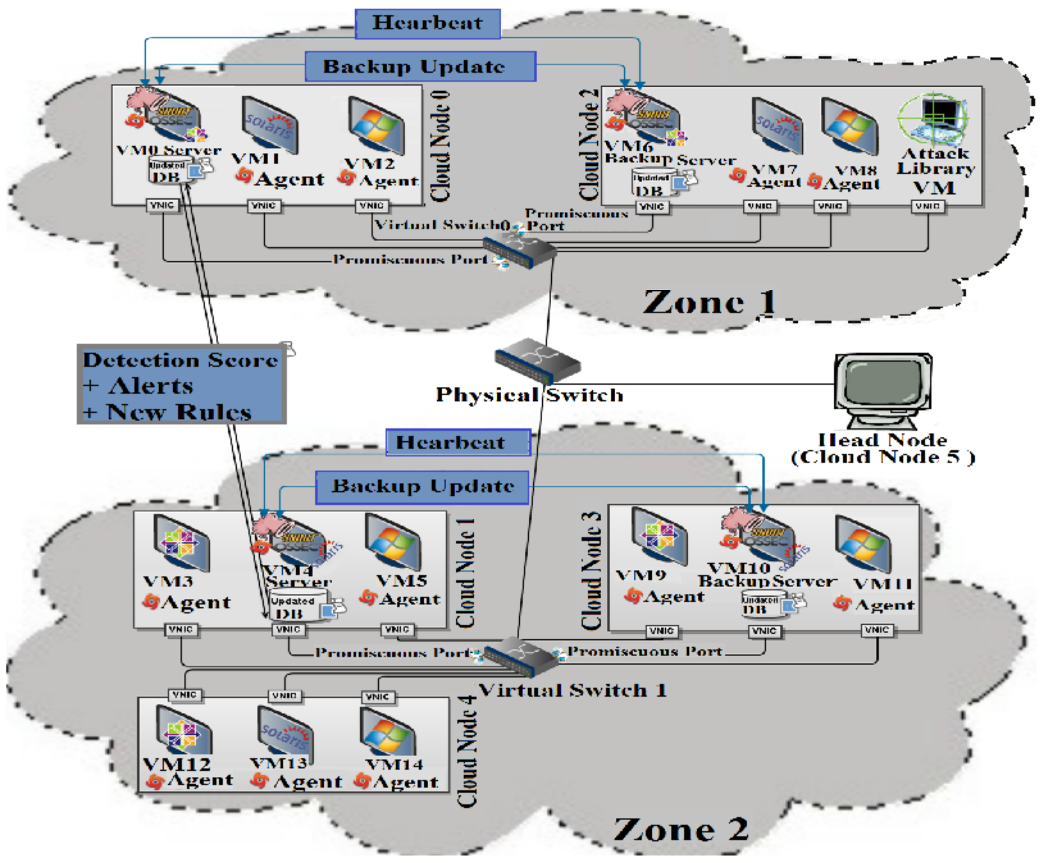


Figure.5: The distributed deployment

4.1.1 DDoS Detection scenario using the Distributed Deployment

The attacker may use several VMs in distinct virtual cloud zones as DDoS zombies to generate a stream of packets towards one or more intended victim machines.

To detect DDoS attacks in the distributed deployment, each Snort server matches the incoming traffic in each zone against pre-defined rules to take some appropriate responses, i.e., drop packet and trigger an alert. The Snort servers exchange three parameters through the CIDS-VERT communication facility namely: the detection score computed by each Snort server, the notification alerts, and the new signature rules. An update of the Snort signature database as in [7] may enable Snort to detect the DDoS attacks by the LOIC and CPU Death Ping libraries. The following rules detect specific behaviors for each protocol [7]:

1. The UDP traffic at specific port, e.g. port 80, is analyzed and the number of connections that are opened in a short time interval is compared against a specific threshold.
2. The TCP rule checks whether an ACK TCP flag is set and check the packets size.
3. The HTTP rule is similar to the TCP one but it checks the packet contents, rather than the size.

When each Snort server has computed its detection score, the Voting Score (*VS*), i.e. the final detection decision, is computed by integrating the detection scores as in Equation 1.

$$VS = \begin{cases} 1, & V(a) > 1/2 \\ F, & V(a) = 1/2 \\ 0, & V(a) < 1/2 \end{cases} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$

Where:

- *V(a)*: Voting of alert *a*. It is the percentage of IDSs that have sent the alert *a*.
- *F*: A flag to denote a relation between, *c*, the number of packets of a given type, e.g., HTTP, UDP, or TCP, a host may send in a given interval, and a threshold *t*, where,

$$F = \begin{cases} 1, & c > t \\ 0, & c \le t \end{cases}$$

Figure 6 shows the DDoS detection scenario in the distributed deployment.

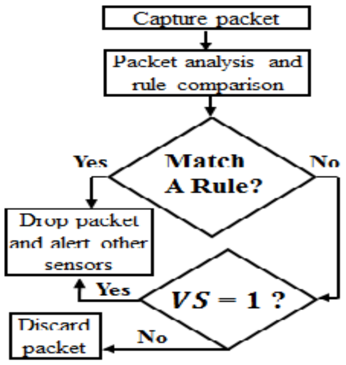

Capture packet
Packet analysis and rule comparison
Match A Rule?
Yes
No
Drop packet and alert other sensors
Yes
VS = 1 ?
No
Discard packet

Figure.6: The DDoS detection flowchart in distributed deployment

4.1.2 Evaluation of the distributed deployment

The main advantages of the distributed deployment are:

1- It distributes the computational overhead among several cloud VMs.
2- A reduced network overhead because network events are not forwarded to a central location.
3- No single point of failure because each Snort server is backed up by another VM.

The disadvantages are:

1- The overall detection time is long because it integrates the outputs of several IDSs.
2- Accuracy is lower than in the centralized deployment because of the integration.

4.2 The Centralized Deployment

This model uses the same components of the other one and it does not change the HIDS functionalities. Instead, it forwards all network packets to a centralized database where they are analyzed by one Snort server. This changes both how Snort servers analyze and collect the packets and the way to backup both the centralized database and the Snort server. In considering the first change, we recall that Snort can run in three different modes: sniffer, packet logger, and detection. In the distributed deployment, all Snort servers run in the detection mode. Instead, in the centralized deployment, the Snort servers run in packet logger mode to log the packets to a centralized database. The VM hosting the database is backed up by another VM in the other zone to avoid a single point of failure. Each Snort server is connected to the central database and to the VM that backups the database. Both servers run in the detection mode to match the packets in the database against Snort rules. They communicate their alerts to the Snort servers in the other zone. Furthermore, each server has a backup in another VM in the same zone.

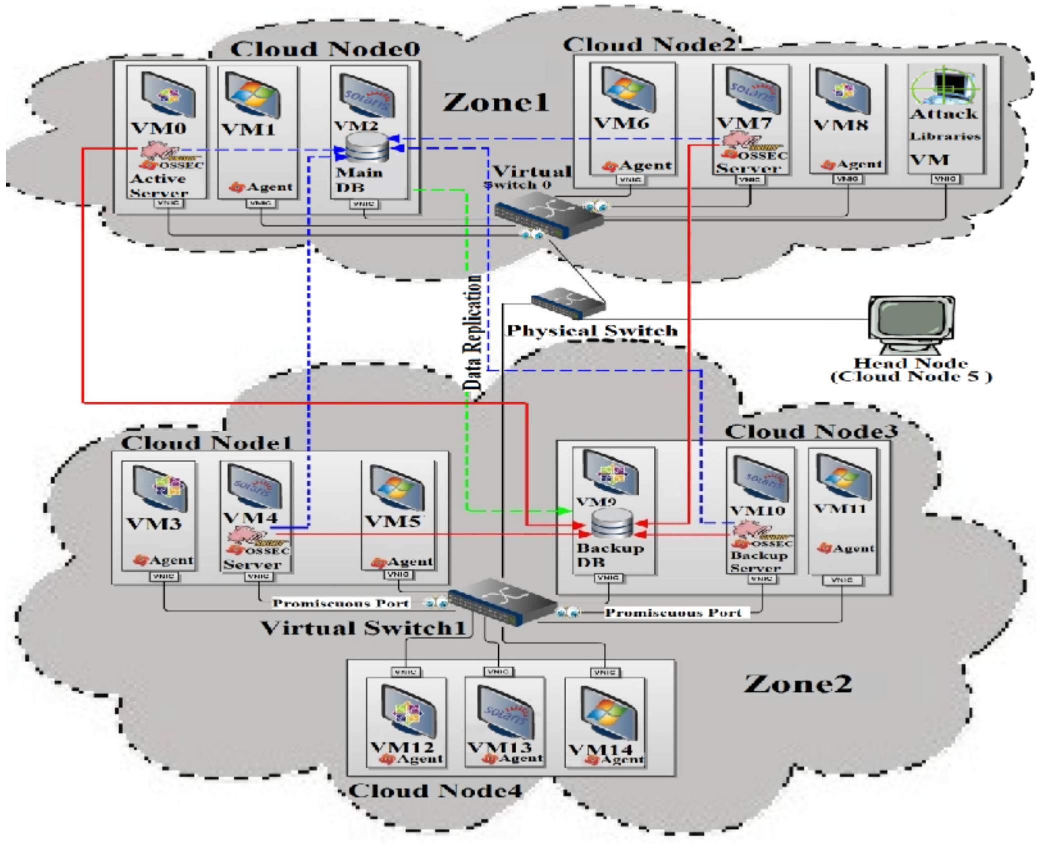


Figure.7: The centralized deployment

As shown in Figure 7, VMs 12 and 19 host, respectively, the centralized database and the backup one. VM0 runs the active Snort server in VZ_1 while VM7 runs the backup. In the same way, VM10 runs the active Snort server in VZ_2 and VM4 the backup. Just one Snort server in the system analyzes the packets in the database and exchanges the heartbeat messages with its backup. The dashed lines in the figure connect the IDSs to the active centralized database, while the solid lines connect them to the backup one.

4.2.1 DDoS Detection scenario using the Distributed Deployment

The centralized deployment can easily detect DDoS attacks, because a single location collects the packets and network events to match them against a central database with pre-defined rules. This improves the detection accuracy and enables the cloud administrator to monitor the system from a central management VM. Figure 8 shows the DDoS attack detection scenario in the centralized deployment option.

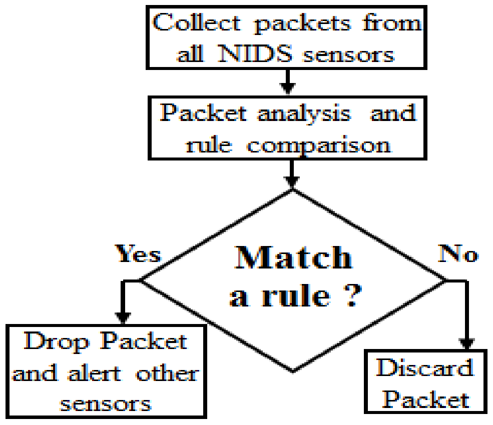

Collect packets from all NIDS sensors
Packet analysis and rule comparison
Match a rule ?
Yes
No
Drop Packet and alert other sensors
Discard Packet

Figure.8: The DDoS detection flowchart in centralized deployment

4.2.2 Evaluation of the Centralized deployment

The main advantages of the centralized deployment are:

1- A short overall detection time because one IDS runs the analysis in one location.
2- The central analysis results in a better accuracy.
3- No single point of failure because of the backup VMs

The corresponding disadvantages are:

1- A high computational overhead for the central VM.
2- A huge overhead for the cloud network because all network packets and events are forwarded to a central VM.

5. Alerts Integration, Correlation, and Risk Assessment

5.1 Alerts Integration:

This layer collects alerts from several detectors i.e., Snort and OSSIC analyzers, and integrates them through two processes namely: normalization and prioritization.

- **The normalization process** formats all alerts from any detector into a unified format, the IDMEF protocol format, to simplify their analysis and correlation in the next layer. The main idea underlying this process is to extract alerts information from the fields with different names and data formats and to represent them in a consistent and common format based on the IDEMF protocol format. Sometimes, more information is added to the normalized alert based on the data source details or on additional data fields in the original alert, e.g., impact severity and sub-event id. Examples of the formatted fields are: the source and target addresses, sub-event id (sid), analyzer, time, priority, classification, and some additional information.

- **The prioritization process** Since each detector has its own prioritization system, e.g., Snort alerts have a maximum priority of 3 while OSSEC alerts have maximum priority of 12, this process maps the priority of the alerts into a single priority range from 0 to *n*, where *n* is a maximum priority as defined by system administrators.

In the following examples, we explain both processes in the cases of the “ICMP PING NMAP” attack and OSSEC with SSHD “brute force” attack. The original alert information is in bold font in the normalized alert:

Snort Alert:

```
1998-06-05:11:12.452605 [**] [122:5:0] (portscan) ICMP PING NMAP [**]
[Classification: Attempted Information Leak][Priority: 3] {ICMP} 192.168.0.1 ->
192.168.0.10
```

Normalized Snort Alert in IDMEF format:

```
<?xml version="1.0" ?> <IDMEF-Message version="1.0"> <Alert ident="12773">
<Analyzer analyzerid="snort00" model="snort" </Analyzer> <CreateTime
ntpstamp="0xb9225b23. 0x9113836a">1998-06-05T11:55:15Z</CreateTime> <Source><Node>
<Address category="ipv4-addr"> <address>192.168.137.1
</address></Address></Node></Source> <Target><Node> <Address category="ipv4-
addr"><address>192.168.137.10</address></Address> </Node></Target> <Classification
origin="vendor-specific"> <name>msg=ICMP PING NMAP</name> </Classification>
<Classification origin="vendor-specific"> <name> sid=384</name>  </Classification>
<Classification origin="vendor-specific"> <name> class= Attempted Information Leak
</name>  </Classification> <Classification origin= "vendor-specific">
<name>priority=3</name> </Classification> <Assessment> <Impact severity="high" />
</Assessment><AdditionalData meaning="sig_rev" type="string" >5</AdditionalData>
<AdditionalData meaning="Packet Payload" type="string">
[illegible]
```

OSSEC Alert:

```
Received From: (csd-wiki) 141.142.234.100->/var/log/secure Rule: 5712 fired (level
10) -> "SSHD brute force trying to get access to the system."
```

Normalized OSSEC Alert in IDMEF format

```
<?xml version="1.0" ?> <IDMEF-Message version="1.0"> <Alert ident="12773">
<Analyzer analyzerid="OSSEC00" model="OSSEC" </Analyzer> <CreateTime ntpstamp=
"0xb9225b23. 0x9113836a">1998-06-05T11:55:15Z</CreateTime> <Source><Node>
<Address category= "ipv4-addr"> <address>192.168.137.1
</address></Address></Node></Source> <Target> <Node> <Address category="ipv4-
addr"><address>192.168.137.10 </address> </Address> </Node></Target>
<Classification origin="vendor-specific"> <name>msg= SSHD brute force trying to
get access to the system </name> </Classification> <Classification
origin="vendor-specific"> <name>sid=5710 </name>  </Classification>
<Classification origin="vendor-specific"> <name>class= ssh-failed </name>
</Classification> <Classification origin="vendor-specific">
<name>priority=10</name> </Classification> <Assessment> <Impact severity="high"
/> </Assessment> <AdditionalData meaning="sig rev"
```

5.2 Alerts Correlation and Summarization:

Alert correlation and summarization is a technique that correlates a large number of normalized alerts from different detectors to highlight the few critical alerts. This technique looks for evidences of an alert to discover it signals a true attack and it correlates logically related alerts. Alerts are logically related if they denote the same attack signature, have the same source and destination addresses, and are close in time. These alerts may also denote a step of a multi-stage or compound attack [25] that consists of set of steps performed by one attacker. In this way, the correlation process: (a) Reduces false positives alerts; (b) Summarizes the huge number of alerts to the cloud administrator; and (c) Deals efficiently with multi-stages attacks. The correlation engine is implemented by OSSIM, an open source system [26] that uses a tree of logical conditions (rules) or AND/OR tree, see Figure 9.

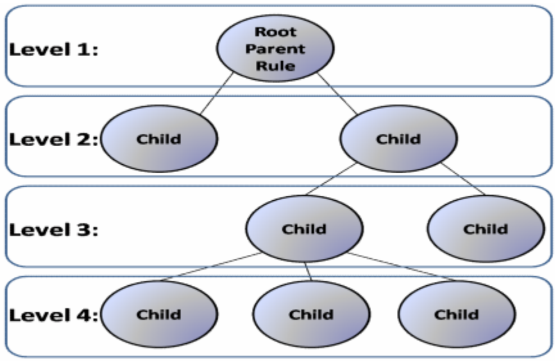


Figure 9: An example for a correlation tree

In the first step of the correlation process, the engine checks if the root parent rule at level 1 is matched, if no, the process stops. Otherwise, the correlation engine jumps to level 2. If no rule in this level is matched the process stops. Otherwise, the engine jumps to the next level and repeats the matching till the end of the tree. The implementation performs ANDING between levels and ORING between level's nodes or Childs. Furthermore, a reliability value is computed in each level to determine the final risk as detailed in Section 5.3.

In the following, we show two correlation examples. The first one describes how the correlation engine helps to detect a brute force attack against an SSH Server [26]. In this example, the alerts are produced by similar analyzers, i.e., Snort. The second example outlines how the correlation engine helps to detect the Reverse Shell attack. In this example, the alerts are produced by different analyzers, i.e., OSSEC and Snort.

- Example 1: The correlation engine builds a four levels correlation tree to detect the brute force attack against an SSH Server, see Figure 10. The following rule is an example of the Childs of this correlation tree; it is the left child of level 2.

```
<rule type="detector" name="SSH Successful Authentication (After 1 failed)"
reliability="1" occurrence="1" from="1:SRC_IP" to="1:DST_IP" port_from="ANY"
time_out="15" port_to="ANY" plugin_id="4003" plugin_sid="7,8"/>
```

Where:

– plugin_id: A unique numerical identifier of the tool (data source) that provides these events.
– plugin_sid: A numerical identifier of the sub-events within the tool (plugin).
– type: type of the rule.
– name: describes what the system expects to collect to satisfy this rule.
– occurrence :Number of events which match rule conditions.
– time_out: Waiting time before the rule expiration.
– from & to : Source IP & Destination IP respectively.
– sensor: the firing sensor of the event.
– port_from & port_to: Source port and Destination port respectively.
– protocol: TCP, UDP, ICMP and etc.
– Reliability: is used to compute the risk value as described in section 5.3.

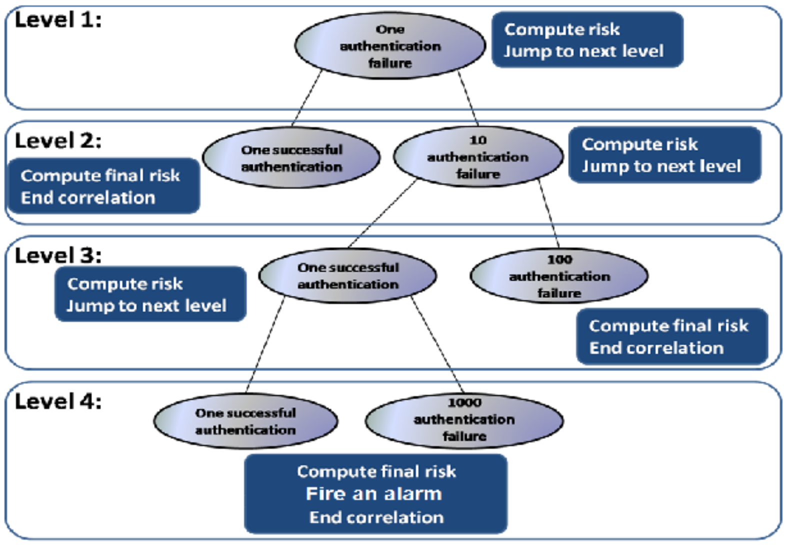


Figure 10: A four levels correlation tree to detect a brute force attack against an SSH Server.

As shown in Figure 10, the correlation tree consists of four levels:

- Level 1**:** This level has a root rule that will be matched by receiving one authentication failed alert. Then, the reliability value is updated and risk is computed, then the correlation engine jumps to the next level.
- Level 2: This level consists of two rules with two possible actions. The first left child rule is matched by a successful authentication alert. Then, the correlation engine updates the reliability value and computes the risk then stops. Otherwise, if 10 authentication failure alerts are received, the second right child rule will be matched. Then, the correlation engine updates the reliability value and computes the risk and jumps to level 3. The engine will evaluate both level 3 and 4in the same way and finally fires an alarm.
- Example 2: The correlation engine detects the Reverse Shell attack by correlating the alerts from both OSSEC and Snort. This is a multi-stages attack that is implemented according to the scenario in Figure 11 [25]:

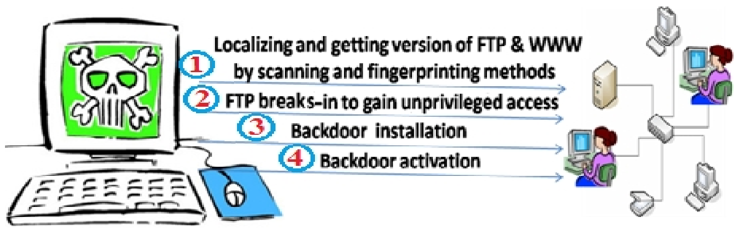

1 Localizing and getting version of FTP & WWW by scanning and fingerprinting methods
2 FTP breaks-in to gain unprivileged access
3 Backdoor installation
4 Backdoor activation

Figure 11: The Reverse Shell attack scenario

The attack is detected as following**:** For each step of the considered scenario, the appropriate IDS, i.e., OSSEC or Snort, fires an alert. While an analysis of each individual alert may be useless, their correlation conveys useful information. The correlation engine applies both the normalization and prioritization processes mentioned before. The final correlation tree consists of the following four levels:

- Level 1 root rule: This rule is matched by *s*canning and fingerprinting alerts from Snort system. Then, the engine updates the reliability value and computes the risk before passing to level 2.
- Level 2: This rule is matched by suspicious ftp logins alerts from OSSEC. Then, the engine updates the reliability, the risk and jumps to level 3.
- Level 3: This rule is matched by a file uploading alert from OSSEC. Then, the correlation engine updates the reliability, the risk and jumps to level 4.
- Level 4: This rule is matched by a Snort alert that denotes the activation and access shell using reverse TCP. Then, the correlation engine updates the reliability value, computes the risk and fires an alarm.

5.3 Risk Assessment:

The correlation engine updates the risk value at each correlation level. This value is computed for each group of alerts through Equation 2 and it denotes how dangerous these alerts are.
$RISK = (Asset * Priority * Reliability)/NF$……………………………. (2)
Where:

- Asset denotes how valuable the attacked resource is. Asset value ranges between (0-*A*), where *A* is the maximum value for the assessment. The Asset value is set by the user in the IDS configuration phase.
- Priority ranges between (0-*P*) where *P* is the maximum priority value and it denotes how dangerous the alert is. This value is set by the firing IDS and is adapted in the prioritization phase as detailed in Section 5.1.
- Reliability (0-*R*) is the probability that the attack defined in a correlation level is real. This value changes in each correlation level as detailed in section 5.2.
- NF is a normalized factor based on *A*, *P*, *R* and maximum risk value (*M*), where $NF = (A * P * R) / M$

Risk Assessment Methodology:

As detailed in Section 5.2, risk is computed when matching the alerts from each IDS in the cloud against the rules in each correlation level. The computation is repeated at each level because the correlation can stop at any level. Once an alert matches a rule, its reliability value will be changed according to the weight of each rule based on the attack signature. When the risk becomes greater than or equal to one, an alarm will be fired. Figure 12 shows the correlation and risk assessment processes with *N* correlation levels each with different number of rules denoted by *k* and *m*.

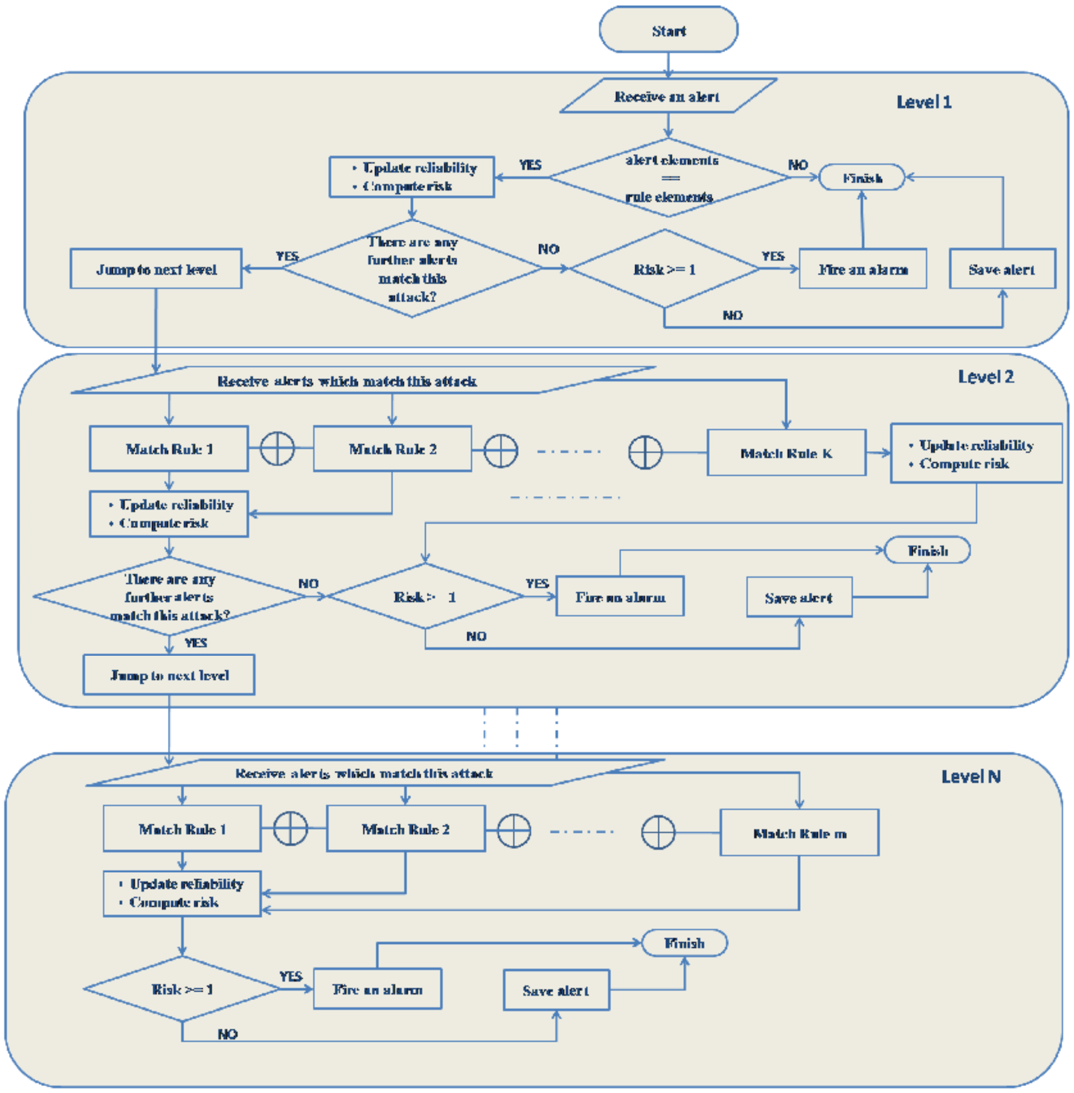


Figure 12: The correlation and risk assessment flowchart

The following example shows how each correlation level computes the risk value. This example highlights the correlation tree in Example 1 of Section 5.2. We highlight some rules of the correlation tree to detect the “SSH Brute Force” attack. [26]

First Level: The rule of this level is matched by receiving one SSH Authentication failure.

```
<rule type="detector" name="SSH Authentication failure" reliability="0"
occurrence="1" from="ANY" to="ANY" port_from="ANY" port_to="ANY"
plugin_id="4003" plugin_sid="1,2,3,6,9"/>
```

The rule matches if all the elements of an alert match those of the current rule. These elements are plugin_id, plugin_sid, type, name, from, to, port_from, port_to, etc.

If this rule is satisfied, the reliability value is set to zero, while the alert priority and asset values are 4 and 5, respectively. Hence, the risk value will be zero.

Second Level: The rules of this level are matched if 11 SSH authentication failure alerts are received in less than 40 seconds. One of these alerts matches one rule in this level and the other 10 alerts match another rule of this level.

```
<rule type="detector" name="SSH Successful Authentication (After 1 failed)
"reliability="1" occurrence="1" from="1:SRC_IP" to="1:DST_IP" port_from="ANY"
time_out="15" port_to="ANY" plugin_id="4003" plugin_sid="7,8"/>
<rule type="detector" name="SSH Authentication failure (10 times)"
reliability="2" occurrence="10" from="1:SRC_IP" to="1:DST_IP"port_from="ANY"
time_out="40" port_to="ANY" plugin_id="4003"
```

If the previous rules are satisfied, the reliability value is set to 2, while the alert priority and asset values are 4 and 5, respectively. Hence, the risk value is (2*4*5)/25 = 1.6. In this case, if no further alerts received that satisfy this attack, an alarm will fire because the risk is larger than one, otherwise, the engine will jump to the next level. In the same way, the third level rules set the reliability value to 4, and the risk value to (4*4*5)/25 = 3.2 and the fourth level rules set the reliability to 7, and the risk value to (7*4*5)/25 = 5.6. Finally, the engine finishes and an alarm with the final risk value will fire because the risk in this case is larger than one.

6. Experimental Results

6.1 Attack Scenario

To evaluate the detection accuracy of the proposed IDS against the host, network and DDoS attacks, we run an attack scenario using the host and network attacks implemented by the Metasploit library. Instead, the DDoS attacks scenario uses both the LOIC and CPU death ping libraries independently. Figure 13 explains these scenarios where the Metasploit library installed in VZ_1 attacks VM6 in the same zone and VMs 11 and 14 in VZ_2. In the DDoS scenario, both LOIC and CPU death ping libraries have some agents distributed in VM8 in VZ_1 and in VMs 5, 11, and 14 in VZ_2. Each agent attacks one VM in its zone and another VM in the other zone. Consequently, the agents of VM8 attacks VM6 and VM3, and the agents of VM5 attack VM3 and VM6. The agents of VM11 attack VM12 and the VM with the Metasploit library. Finally, the agents of VM14 attack VM12 and the VM with Metasploit library. LOIC floods the system by TCP and UDP packets, while CPU Death Ping floods by ICMP packets and HTTP requests.

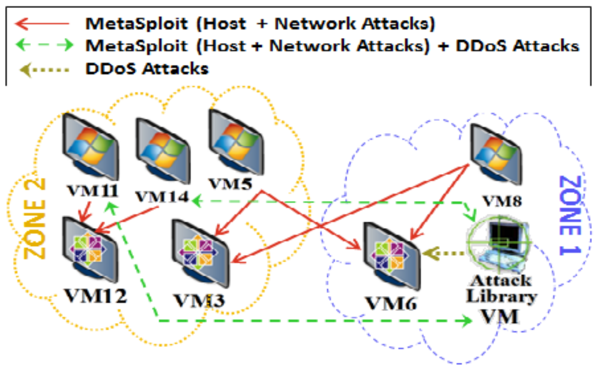

MetaSploit (Host + Network Attacks)
MetaSploit (Host + Network Attacks) + DDoS Attacks
DDoS Attacks
ZONE 2
ZONE 1
VM11
VM14
VM5
VM8
VM12
VM3
VM6
Attack Library VM

Figure 13: The host, network, and DDoS attacks scenarios

6.2 A Performance Evaluation for the Two Deployments

This section evaluates the proposed IDS by analyzing the traffic of each Snort sensor in the Centralized, VZ_1, and VZ_2. Furthermore, it evaluates the accuracy and the computational performance of the Centralized and Distributed deployments. Figure 14 shows the spikes of the DDOS attacks resulted by the TCP floods of LOIC and the HTTP floods of the CPU Death PING library. Figure 15 shows the spikes of the DDOS attacks resulted by the UDP floods of LOIC and the ICMP floods of the CPU Death PING library. Both libraries run the attacks for 10 minutes. Each graph is divided into four parts to show the traffic in the cloud network in different situations. In part A, the cloud system acts in normal mode, and the system replies to legal packets without any problem. In part B the flooding attack starts and the traffic rate increases. Consequently, the cloud system becomes unable to respond to its users. In part C, the IDS starts to handle the attack and blocks the illegal packets. Finally, after detecting and blocking the attacks, in part D, the traffic rate is normal again.

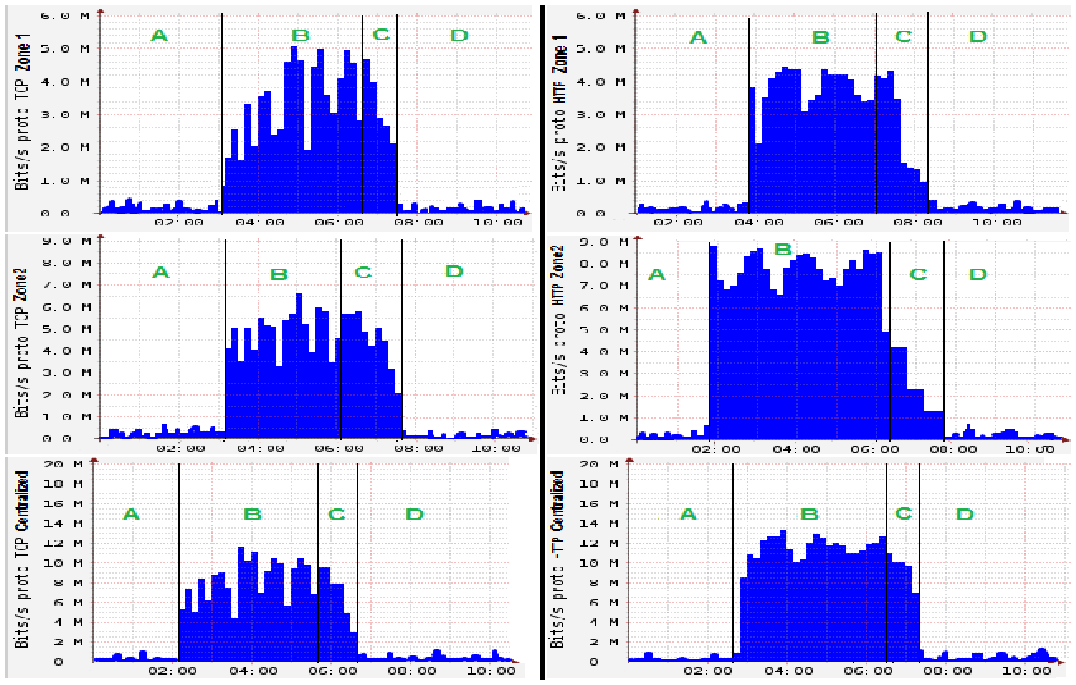


Figure 14: The DDOS by TCP floods using LOIC and the HTTP floods using CPU Death PING

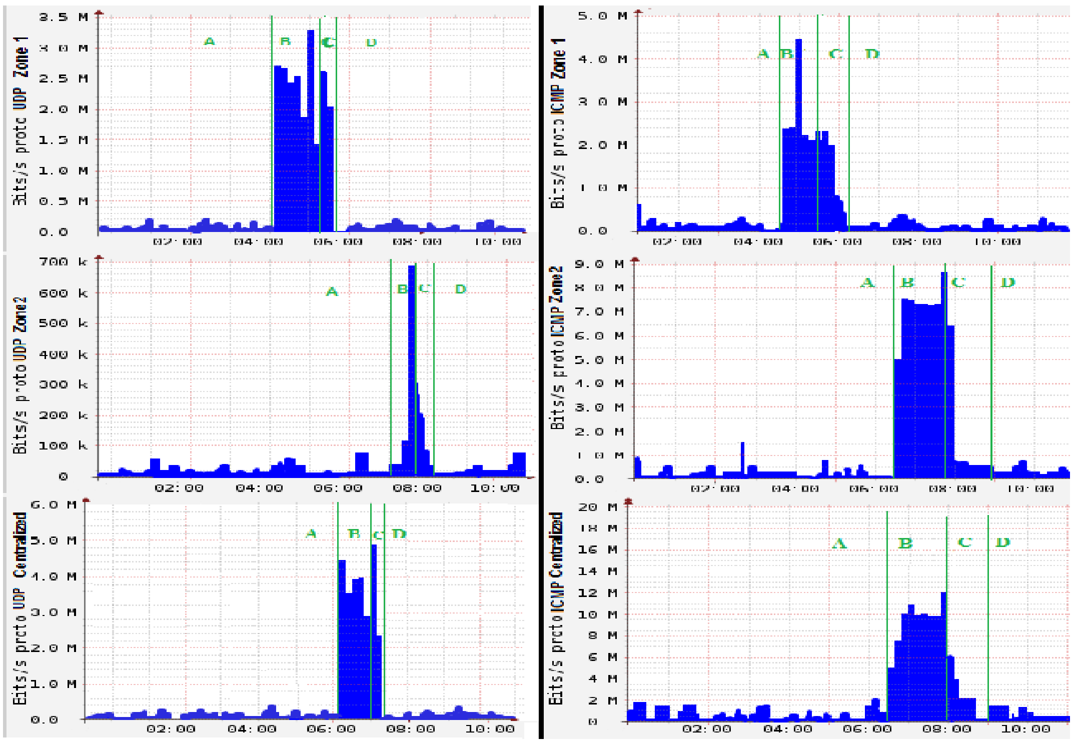


Figure 15: The DDoS by UDP floods using LOIC and ICMP floods using CPU Death PING

The two proposed deployments are compared in terms of the detection accuracy and computation time over 20,000 data packets. Figure 16 shows that the Centralized deployment signals a higher number, 34.3%, of true alerts than the Distributed deployment. The higher accuracy can be explained as the Centralized deployment takes the detection decision based on all events centralized in one location. However, Figure 17 shows a lower computation time, 28.8%, of the Distributed deployment than the Centralized one. In fact, the Distributed deployment distributes the detection overhead among several sensors in the cloud and it can directly drop any packet that matches a rule without any further analysis.

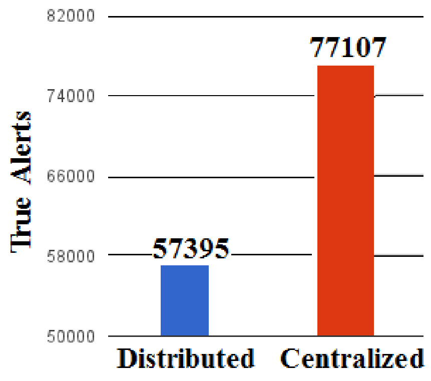


Figure 16: Number of true alerts signaled by the Centralized and Distributed Deployment

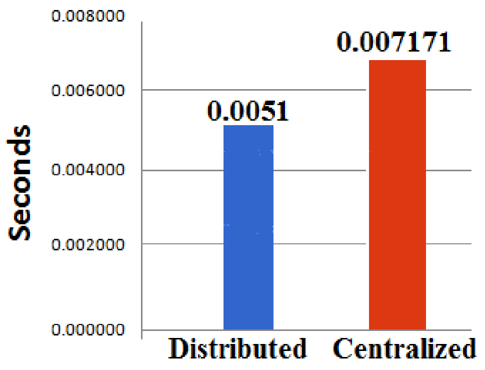
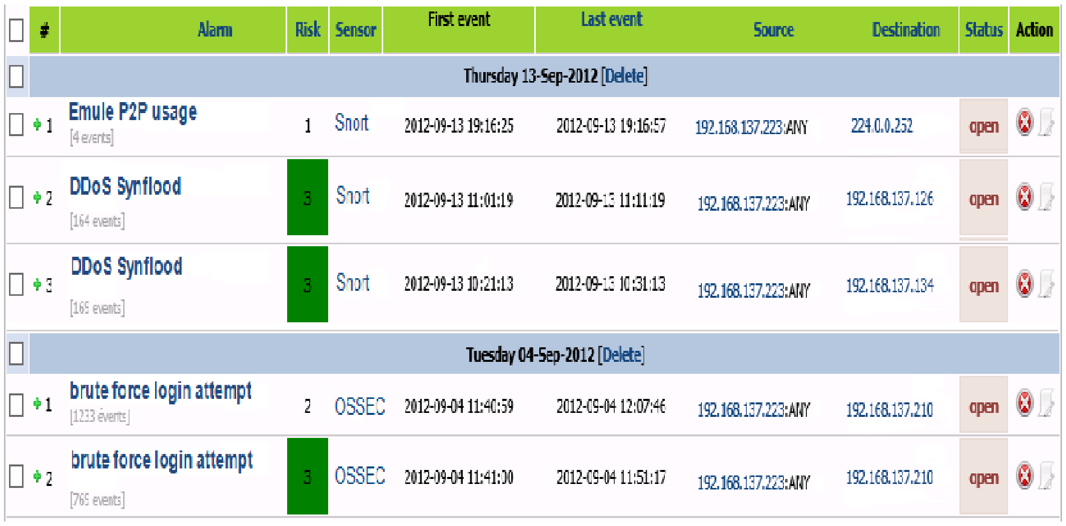


Figure 17: The computation time of the Centralized and Distributed Deployment

6.3 HIDS and NIDS detection outputs

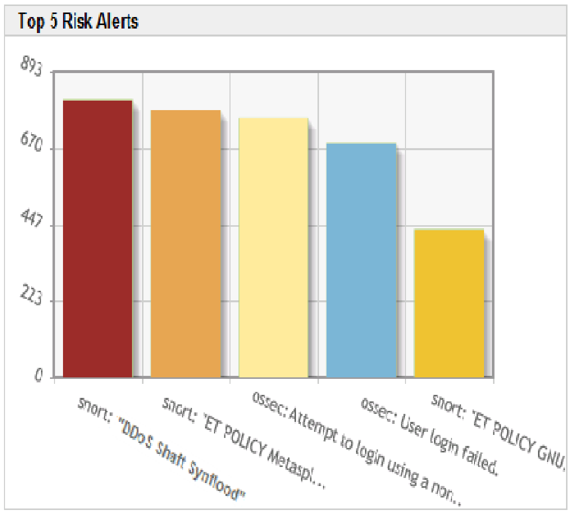
The Web Interface layer offers a visual tool to manage and admin the IDS components and to display the detected attacks. Figure 18 shows a snapshot of some detected attacks with their corresponding risk values after correlating the alerts from OSSIC and Snort IDSs. We use the attack libraries of VM "192.168.137.223" and other VMs as in Section 6.1 to attack the cloud.



#	Alarm	Risk	Sensor	First event	Last event	Source	Destination	Status	Action
Thursday 13-Sep-2012 [Delete]									
1	Emule P2P usage [4 events]	1	Snort	2012-09-13 19:16:25	2012-09-13 19:16:57	192.168.137.223:ANY	224.0.0.252	open	
2	DDoS Synflood [164 events]	3	Snort	2012-09-13 11:01:19	2012-09-13 11:11:19	192.168.137.223:ANY	192.168.137.126	open	
3	DDoS Synflood [165 events]	3	Snort	2012-09-13 10:21:13	2012-09-13 10:31:13	192.168.137.223:ANY	192.168.137.134	open	
Tuesday 04-Sep-2012 [Delete]									
1	brute force login attempt [1233 events]	2	OSSEC	2012-09-04 11:40:59	2012-09-04 12:07:46	192.168.137.223:ANY	192.168.137.210	open	
2	brute force login attempt [765 events]	3	OSSEC	2012-09-04 11:41:30	2012-09-04 11:51:17	192.168.137.223:ANY	192.168.137.210	open	

Figure 18: A snapshot of detected host and network attacks in the cloud system

Figure 19 shows the top 5 alerts with high risk value that were fired by both OSSEC and Snort IDSs in different locations of the cloud. Figure 20 shows the top 10 VMs that signaled multiple alerts.



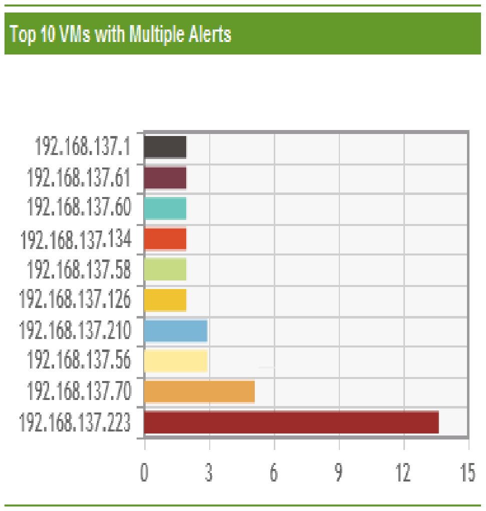


Figure 19: The top 5 alerts with high risk value fired by OSSEC and Snorts.

Figure 20: The top 10 VMs with multiple alerts in the cloud system.

7. Conclusion and Future Work

DDoS attacks can severely degrade the availability of cloud services. These attacks and those that may violate the integrity and the confidentiality of cloud data make users have little trust in cloud computing. This paper introduced two deployments for HIDS and NIDS IDS in a cloud system, namely, the Distributed and the Centralized Cloud IDSs. The Distributed deployment distributes the computational overhead among several cloud VMs and reduces network overhead with no single point of failure. However, the overall detection time is long and the accuracy of this deployment is lower than the Centralized one. The Centralized deployment has a shorter detection time than the Distributed one and achieves a higher accuracy with no single point of failure. However, it results in a large overhead for the central VM and the cloud network. The experiments show that the Centralized deployment improves the detection rate with respect to the Distributed deployment, while the Distributed deployment has a better detection time. For an efficient detection, we have integrated and correlated the HIDS and NIDS alerts through IMDEF. This helps in detecting multi-stage or compound attacks. Furthermore, it reduces both false alarms and the alerts from HIDS and NIDS.

For the future work, we plan to integrate the behaviour based detection of our DDSGA approach with the current signature based detection techniques. DDSGA can detect anomalous behaviours for both users and hosts in the network. Hence, if the Handler machines or their users have a profile of normal behaviours, DDSGA can compare it against anomalous actions, and block the anomalous user or host. We have given a similar analysis based on the masquerade actions in system calls and NetFlow in [1] and another one based on the security event and NetFlow in [27]. The anomalous actions that can be detected for DDoS include, among others, sending packets with a suspect total length or a number of packets with a total length larger than normal threshold in a specific time range.

ACKNOWLEDGMENT:

This work is partially supported by IBM, AFISR DDDAS award number FA95550-12-1-0241, and National Science Foundation research projects NSF IIP-0758579, NCS-0855087 and IIP-1127873.

References

[1] Hisham A. Kholidy, Fabrizio Baiardi, Salim Hariri, “Detecting Masquerades in Clouds through System calls and NetFlow Data Analysis”, Ready for submission to the IEEE Transactions on Dependable and Secure Computing Journal.
[2] http://technet.microsoft.com/en-us/library/cc959354.aspx
[3] http://en.wikipedia.org/wiki/Attack_(computing)
[4] http://www.metasploit.com/
[5] Stein, Lincoln. The World Wide Web Security FAQ, Version 3.1.2, February 4, 2002. http://www.s3.org/security/faq/ - visited on October 1, 2002.
[6] Zaroo, P.; “A survey of DDoS attacks and some DDoS defense mechanisms”, Advanced Information Assurance (CS 626), 2002
[7] Montoro, R.; “LOIC DDoS Analysis and Detection”, URL: http://blog.spiderlabs.com/2011/01/loic-ddos-analysis-and-detection.html, 2011, Accessed December 1, 2011
[8] http://www.tenable.com/products/nessus
[9] http://www.hackerbradri.com/2012/08/cpu-death-ping-20.html

[10] Amir Vahid Dastjerdi, Kamalrulnizam Abu Bakar, Sayed Gholam Hassan Tabatabaei, “Distributed Instrusion Detection in Clouds Using Mobile Agents”, Third International Conference on Advanced Engineering Computing and Application in Sciences, October 11-16, 2009 - Sliema, Malta

[11] Roschke, S., Cheng, F., Meinel, “Intrusion Detection in the Cloud”, The 8th International Conference on Dependable, Autonomic and Secure Computing (DASC-09) China, Dec. 2009

[12] Aboosaleh Mohammad Sharifi, Saeed K. Amirgholipour1, Mehdi Alirezanejad2, Baharak Shakeri Aski, and Mohammad Ghiami "Availability challenge of cloud system under DDoS attack", Indian Journal of Science and Technology, Vol. 5 No. 6 (June 2012) ISSN: 0974- 6846

[13] Anjali Sardana and Ramesh Joshi, “An auto responsive honeypot architecture for dynamic resource allocation and QoS adaptation in DDoS attacked networks”, Journal of Computer and Communications, July 2009, Vol. 32, P 121384-1399.

[14] Aman Bakshi, Yogesh B. Dujodwala, "Securing Cloud from DDoS Attacks Using Intrusion Detection System in Virtual Machine", Proceedings of the 2010 Second International Conference on Communication Software and Networks(ICCSN '10), P 260-264

[15] Weir, J.; “Building a Debian\Snort based IDS”, URL: http://www.snort.org/docs, 2011. Accessed November 28, 2011

[16] VMware cloud, http://www.vmware.com/solutions/cloud-computing/index.html

[17] Chi-Chun Lo, Chun-Chieh Huang and Ku, J, “A Cooperative Intrusion Detection System Framework for Cloud Computing Networks”. In 2010 39th International Conference on Parallel Processing Workshops.

[18] H. Debar, D. Curry, “The Intrusion Detection Message Exchange Format (IDMEF)”, rfc4765, March 2007.

[19] Hisham. A. Kholidy, Fabrizio Baiardi, "CIDS: A framework for Intrusion Detection in Cloud Systems", in the 9th Int. Conf. on Information Technology: New Generations ITNG 2012, April 16-18, Las Vegas, Nevada, USA. http://www.di.unipi.it/~hkholidy/projects/cids/

[20] Microsoft Private cloud, http://www.microsoft.com/en-us/server-cloud/private-cloud/default.aspx

[21] Open stack, http://www.openstack.org/

[22] Eucalyptus, http://www.eucalyptus.com/

[23] Hisham A. Kholidy, Fabrizio Baiardi, Salim Hariri, “DDSGA: A Data-Driven Semi-Global Alignment Approach for Detecting Masquerade Attacks”, in IEEE Transactions on Dependable and Secure Computing, under review in September 2012.

[24] http://www.ossec.net/main/

[25] "Detection of Multistage Attack in Federation of Systems Environment", Przemysław Bereziński, Joanna Śliwa, Rafał Piotrowski, Bartosz Jasiul- Military Communication Institute

[26] “OSSIM Manual”, http://www.alienvault.com/documentation/index.html

[27] Hisham A. Kholidy, Fabrizio Baiardi, Salim Hariri, “Detecting Masquerades in Clouds through Security Events and NetFlow Data Analysis”, Ready for submission to the IEEE Transactions on Dependable and Secure Computing Journal.

AUTHORS:

Hesham AbdElazim Ismail Mohamed Kholidy, MCP (Microsoft Certified Professional), ETS (Educational Testing Service Certified Professional), he received his B. S. and MSc degrees in computers and information from Helwan University in Egypt in 2003 and 2008 respectively. He is now a researcher in NSF Cloud and Autonomic Computing Center in Electrical and Computer Engineering Dept. at The University of Arizona and a Ph.D. candidate at the University of Pisa in Italy. He is a senior developer and worked as assistant teacher for computer science in Faculty of computers and Information in Fayoum University in Egypt. He got the first position in national programming competition by ITI, Egypt. His research interests include distributed systems including Grid and Cloud computing systems and their security infrastructure, information security, performance evaluation, load balancing, and Elearning Applications.

Fabrizio Baiardi received his degree from Università di Pisa, where is currently a full professor of Computer Science. He chairs the Master Degree on Computer Security and teaches some courses on the security of cloud systems. His mainly research interests include formal tools for the risk assessment and management of cloud systems and critical infrastructures. He is currently involved in several projects on intrusion detection of cloud and SCADA systems.

Salim Hariri received the MSc degree from Ohio State University in 1982 and the Ph.D. degree in computer engineering from the University of Southern California in 1986. He is a professor in the Electrical and Computer Engineering Department at the University of Arizona and the director of the NSF Center: Cloud and Autonomic Computing Center. He is the editor in chief for Cluster Computing: The Journal of Networks, Software Tools, and Applications. His current research focuses on high performance distributed computing, agent-based proactive and intelligent network management systems, design and analysis of high speed networks, and developing software design tools for high performance computing and communication systems and Applications. He has coauthored more than 200 journal and conference research papers and co-author/editor of three books on parallel and distributed computing. He is a member of IEEE Computer Society.

Esraa Mohammed Hashem El-hariri, she works as teaching assistant for computer science in Faculty of computers and Information in Fayoum University in Egypt. She is a member of Scientific Research Group in Egypt (SRGE) and a master student at cairo university. Her research interests include Cloud Computing Security, Intelligent Environment and its applications.

Ahmed M. Yousof, He received his B. S. in Electronics in 1999 and his diploma in Computers and ControlSystem in 2002 from Mansoura University. He works as a senior software engineer at Middle Delta for Electricity Production Co. He is a master researcher at Mansoura University. His research interests include Cloud Computing Security and Open Sources.

Sahar A. Shehata, She works as Information technology Engineer at Middle Delta for Electricity Production Co. She is a master researcher at Mansoura University. Her research interest is Cloud Computing Security.