DIPARTIMENTO DI INFORMATICA
DOTTORATO DI RICERCA IN INFORMATICA

PH.D. THESIS

Cloud Computing Security, An Intrusion Detection System for Cloud Computing Systems

Hesham Abdelazim Ismail Mohamed

SUPERVISORS:

Prof. Fabrizio Baiardi
Dipartmento di Informatica, Pisa University, Italy
President of the council of information security

Prof. Salim Hariri
Electrical and Computer Engineering Department
University of Arizona, USA.
The Director of NSF Center: Cloud and Autonomic Computing

June, 2013

To the most precious inspiration of my life:
My parents and my brothers and sisters

General Acknowledgments

In the name of Allah, Most Gracious, Most Merciful

I would like to express my deepest respect and most sincere gratitude to all those who made this dissertation possible by their support specially my advisors, Prof. Fabrizio Baiardi and Prof. Salim Hariri, for the source of knowledge, guidance, and support that they have been to me over the past years.

I would like to give my special thanks to the Computer Science Department of the University of Pisa for their logistic and financial support to study and apply practical work abroad in colleague of Engineering of University of Arizona in USA and for giving me the opportunity to have an international PH.D. title through the collaboration with University of Arizona. I give my special thanks to the president of the doctorate school, Prof. Pierpaolo Degano, for his kind support. I also give my sincere gratitude to the members of the NSF Center: Cloud and Autonomic Computing Center in University of Arizona for helping me to complete the practical work of my dissertation. I also would like to thank the internal PH.D. committee members, Prof. Fabrizio Luccio and Prof. Maurizio Bonuccelli for their help, suggestions, and advices. I give great thanks to my university in Egypt, Fayoum University, and the dean of the faculty Prof. Nabila Hassan for facilitating the procedures to study in this scholarship.

Most of all, I would like to thank my parents, my brothers, and sisters for their encouragement and emotional support. This dissertation is dedicated to them.

Hesham Abdelazim Ismail Mohamed

Thesis Acknowledgments:

- This PH.D. is achieved through an international joint program with a collaboration between University of Pisa in Italy (Department of Computer Science, Galileo Galilei PH.D. School) and University of Arizona in USA (College of Electrical and Computer Engineering).
- The PH.D. topic is categorized in both Computer Engineering and Information Engineering topics.
- The thesis author is also known as "Hisham A. Kholidy" as referred to in our publications underlying the thesis work.
- The thesis supervisors and committee members are:

1) Prof. Fabrizio Baiardi (Department of Computer Science, Pisa University, Italy)	Supervisor
2) Prof. Salim Hariri (Department of Electrical and Computer Engineering, University of Arizona, USA)	Supervisor
3) Prof. Aris M. Ouksel (Department of Computer Science, Information and Decision Sciences, University of Illinois at Chicago, USA)	Committee Member (External Reviewer)
4) Prof. Manish Parashar (Department of Electrical and Computer Engineering, Rutgers, The State University of New Jersey, USA)	Committee Member (External Reviewer)
5) Prof. Fabrizio Luccio (Department of Computer Science, Pisa University, Italy)	Committee Member (Internal Reviewer)
6) Prof. Maurizio Bonuccelli (Department of Computer Science, Pisa University, Italy)	Committee Member (Internal Reviewer)
7) Prof. Sherif Abdelwahed (Department of Electrical and Computer Engineering, Mississippi State University, USA)	External Examiner
8) Prof. Chiara Bodei (Department of Computer Science, Pisa University, Italy)	Internal Examiner
9) Prof. Sebastiano Vigna (Department of Computer Science, University of Milan, Italy)	External Examiner
10) Prof. Riccardo Scateni (Department of Mathematics and Computer Science, University of Cagliari, Italy)	External Examiner

Table of Contents

Table of Figures

Figures of Introduction

Figures of Chapter 1

Figures of Chapter 2

Figures of Chapter 3

Figures of Chapter 4

Figures of Chapter 5

Figures of Chapter 6

Figures of Chapter 7

Figures of Chapter 8

Figures of the conclusion

Table of Tables

Tables of Chapter 2

Tables of Chapter 3

Tables of Chapter 4

Tables of Chapter 5

Tables of Chapter 6

Tables of Chapter 7

Table of Equations

Table of Algorithms

List of Abbreviations

ANN:	Artificial Neural Network
ASSL:	Average Signature Session Length
BSM:	Basic Security Module
CIDD:	Cloud Intrusion Detection Dataset
CIDS:	Our proposed Cloud Intrusion Detection System
CIDS-VERT:	Full virtual version of CIDS
CSA:	Cloud Security Alliance
DDoS:	Distributed Denial of Service Attack
DDSGA:	Data-Driven Semi-Global Alignment
DIDS:	Distributed Intrusion Detection System
DoS:	Denial of Service attack
FPR:	False Positive Rate
FTP:	File Transfer Protocol
GDP:	Generalized Delta Procedure
GIDS:	Grid based Intrusion Detection Systems
GUI :	Graphic User Interface
HIDS:	Host-based Intrusion Detection System
HIMAN	Hisham and Abdelrahman Grid Middleware System
HXen:	Hosted Xen project
IaaS:	Infrastructure as a Service
IDMEF:	Intrusion Detection Message Exchange Format
IDS:	Intrusion Detection System
IPAM:	Incremental Probabilistic Action Modeling
LACS:	Log Analyzer and Correlator System
MAIDS:	Mobile Agent Intrusion Detection Systems
MAs:	Mobile Agents
MCE:	Minimum Conditional Entropy
MFTG:	Maximum factor of test gaps
NAC:	Number of Asymptotic Computations
NF:	Normalized Factor
NIDS:	Network-based Intrusion Detection System
NIST:	National Institute of standards and Technology
NN:	Number of cloud nodes running user VMs
OS:	operating system
OSSEC:	Open Source Host-based Intrusion Detection System
PaaS:	Platform as a Service
R2L:	Remote to Local attack

ROC:	The Receiver Operator Characteristic curve
SaaS:	Software as a Service
SC:	System Calls
SE:	Security Events
SGA:	Semi-Global Alignment
SMTP:	Simple Mail Transfer Protocol
SVM:	Support vector machine
TBA:	Trace Backward Algorithm
TLU:	Threshold Logic Unit
TMBO:	Top-Matching Based Overlapping
TOS:	Type of Service
TPR:	Positive Rate
TSL:	Test Session Length
TSLSA:	Test Session Length with Sensitive Action
U2R:	User to Root attack
VM:	Virtual Machines
VMM:	Virtual Machine Monitors

Abstract

Cloud computing is widely considered as an attractive service model because it minimizes investment since its costs are in direct relation to usage and demand. However, the distributed nature of cloud computing environments, their massive resource aggregation, wide user access and efficient and automated sharing of resources enable intruders to exploit clouds for their advantage. To combat intruders, several security solutions for cloud environments adopt Intrusion Detection Systems. However, most IDS solutions are not suitable for cloud environments, because of problems such as single point of failure, centralized load, high false positive alarms, insufficient coverage for attacks, and inflexible design.

The thesis defines a framework for a cloud based IDS to face the deficiencies of current IDS technology. This framework deals with threats that exploit vulnerabilities to attack the various service models of a cloud system. The framework integrates behaviour based and knowledge based techniques to detect masquerade, host, and network attacks and provides efficient deployments to detect DDoS attacks.

This thesis has three main contributions. The first is a Cloud Intrusion Detection Dataset (CIDD) to train and test an IDS. The second is the Data-Driven Semi-Global Alignment, DDSGA, approach and three behavior based strategies to detect masquerades in cloud systems. The third and final contribution is signature based detection. We introduce two deployments, a distributed and a centralized one to detect host, network, and DDoS attacks. Furthermore, we discuss the integration and correlation of alerts from any component to build a summarized attack report. The thesis describes in details and experimentally evaluates the proposed IDS and alternative deployments.

Key Words: cloud computing, security, intrusion detection, attacks, masquerade, dataset, masquerade detection, sequence alignment, system calls, security events, NetFlow, feature extraction, DDoS.

Introduction

Cloud computing is a large-scale distributed computing paradigm driven by economies of scale and outsourcing, where a pool of abstracted, virtualized, dynamically-scalable, managed computing, storage, platforms, and services are delivered on demand over the Internet. Cloud computing technology is enabling IT managers to provision services to users faster and in a much more flexible and cost-effective way without having to re-design or update the underlying infrastructure [4].

Clouds in general provide services at three different levels [2] defined by what is called "SPI" models or Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). In SaaS, a consumer can use applications running on a cloud infrastructure. In PaaS, the consumer can deploy onto the cloud infrastructure consumer-created or acquired applications developed through programming languages and tools supported by the provider. In IaaS, the provider offers a large amount of interconnected computing nodes to run a consumer-created network of virtual machines. Four deployment models exist for cloud services, with derivative variations that address specific requirements namely [4]: Public, Private, Community, and Hybrid Cloud. In a Public Cloud, the cloud infrastructure is made available to the general public or a large user group. In a Private Cloud, the cloud infrastructure is operated for a single organization. In a Community Cloud, the cloud infrastructure is shared by several organizations to support a specific community that has shared concerns, and in Hybrid Cloud, the cloud infrastructure interconnects two or more clouds (private, community, or public).

Given the benefits of cloud computing, its broad appeal is not surprising. However, this new approach does raise some concerns. Chief among them is securing data in the cloud. Security controls in cloud computing are, for the most part, the same ones that any IT environment can apply. However, because of cloud service and operational models and the technologies to enable these services, cloud computing may present new risks and threats to an organization. Furthermore, due to their distributed nature, cloud computing environments are easy targets for intruders looking for possible vulnerabilities to exploit. The impact of intrusions in cloud systems is potentially very large, as intruders can exploit for their advantage the

massive resource aggregation, wide user access, efficient and automated resource allocation of a cloud. The Cloud Security Alliance has defined seven top threats to cloud computing systems [1] namely:

1. Abuse and Nefarious Use of Cloud Computing.
2. Insecure Interfaces and APIs.
3. Malicious Insiders.
4. Shared Technology Issues.
5. Data Loss or Leakage.
6. Account or Service Hijacking.
7. Unknown Risk Profile.

Some of these threats can be handled by an Intrusion Detection System (IDS). An IDS is a software or a hardware system that monitors and analyzes events in a computer system or network to discover signs of security problems. As attacks increase in number and severity, IDSs have become a necessary addition to the security infrastructure of most organizations. There are three main categories of IDSs based on the protection objectives, namely [2]:

- Host-based Intrusion Detection System (HIDS), where sensors to detect an intrusion are focused on a single host.
- Network-based Intrusion Detection System (NIDS), where sensors are focused on a network segment.
- Distributed Intrusion Detection System (DIDS) which integrates both types of sensors (i.e., HIDS and NIDS).

According to the underlying technology and the characteristics of the system to be protected, DIDS can be categorized as Mobile Agent Intrusion Detection Systems (MAIDS), Grid based Intrusion Detection Systems (GIDS), and recently Cloud based Intrusion Detection Systems.

Several deficiencies of current IDSs solutions hinder their adoption in a cloud environment. As an example, an attack against a cloud can be silent for a host-based IDSs (HIDS), because cloud-specific attacks may not leave traces in the operating system of the node running the host-based IDS. Since in a cloud computing environment distinct users share both computing and communication resources, attacks may be originated within the infrastructure itself. Hence, an attack originating in the cloud can be directed against resources of the same cloud. This increases the complexity of attack

detection for a NIDS as all the communications that implement the attack are among cloud nodes. Current distributed IDSs have shown their effectiveness in some large scale networks, but their adoption in cloud computing is still challenging. The complex architecture of a cloud infrastructure and the distinct kinds of users lead to different requirements and possibilities for an IDS. As an example, a distributed hierarchical IDS may be scalable but it has the problem of single point of failure because if any part of an internal node is disabled, a branch of the IDS will be unreliable. Furthermore, some IDSs lack the flexibility to support distinct cloud architectures. Another deficiency is that several IDSs detect attacks through either the behaviour base technique or the knowledge base one. Instead, a good IDS should integrate both techniques because the latter is efficient in detecting known attack patterns with low false positive alarms, i.e. an alert almost always signals a real attack, but it does not detect unknown attacks or even trivial modification of known attacks. Instead, the knowledge based technique detects unknown attacks but it has high false positive alarms.

Hence, a proper defense strategy needs to:

1. Support distinct cloud computing environments.
2. Be distributed, resilient and with no single point of failure.
3. Protect the intrusion detection components.
4. Isolate the host from any vulnerability from the executed tasks.
5. Integrate behaviour and knowledge base techniques to increase attack coverage.
6. Collect and correlate user behaviours from all environments in the cloud system.

Building a new cloud based IDS is a new challenging area of research. Few papers have addressed cloud IDSs in general and some have proposed some frameworks for cloud systems but without any real implementation and most of them do not satisfy the previous requirements.

The main goal of this thesis is to develop a framework for a cloud based intrusion detection system that satisfies the previous requirements and deals with the following attacks:

1. Masquerade attacks: they abuse the privileges of a legitimate user to illegally use a service's abundant resources. Only a behaviour based analysis can detect them.
2. Host based attacks: they may be consequences of the previous attacks and result in an observable user behavior anomaly or leave some trails at the VM operating system.
3. Network-based attacks: They generally result in an observable user behavior anomaly or leave some trails at the Network packets.
4. Distributed Attacks: they are implemented with the help of tools or exploit scripts and include denial-of-service attacks, probes, and worms. They may leave their trails at several locations of a cloud's infrastructure.

The proposed framework assumes a fully distributed architecture to provide a scalable and elastic solution. To avoid a single point of failure, the framework distributes the processing tasks among cloud nodes. It isolates these tasks from the cloud node system, by executing them in a VM monitored by a VM monitor. The framework achieves a high coverage of attacks by integrating both knowledge and behaviour based techniques. It also provides an audit system to support the adoption of the framework in distinct cloud computing environments. This system also collects and correlates the user behaviours from the cloud VMs. The framework integrates the alerts from different IDSs and builds a final summarized attack report.

To support several deployment models we have introduced two versions of the framework, the Cloud based Intrusion Detection System, CIDS, and its full virtual version, CIDS-VERT. CIDS P2P architecture hinders scalability but it achieves a high performance and low network overhead in small or private clouds. CIDS-VERT is ideal for hybrid and public clouds due to its better scalability and controllability. However, it consumes a large amount of resources because it reserves some management VMs for the detection and management tasks. The diagram in Figure 1 shows the main contributions of the thesis and the corresponding chapters.

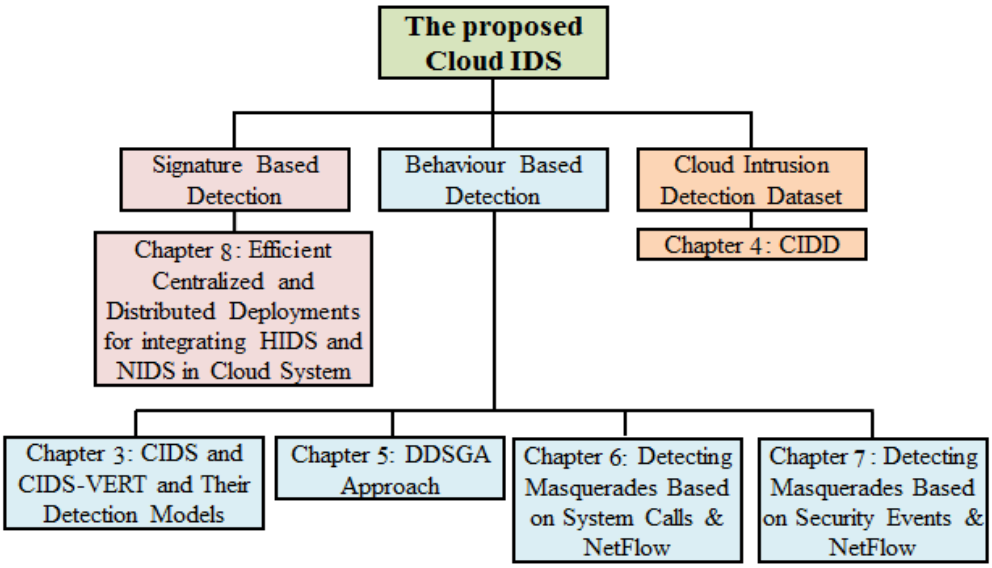


Figure.1: Contributions of the thesis and chapters organization

The first contribution is CIDD, the first cloud intrusion detection dataset that can be used to train and test any cloud IDS. It consists of both knowledge and behavior based audit data and has real instances of host and network based attacks and masquerades. CIDD provides complete diverse audit parameters from several environments to evaluate the effectiveness of detection techniques.

The second contribution is related to behaviour based detection. The thesis defines the Data-Driven Semi-Global Alignment, DDSGA approach [DDSGA], and three detection strategies. DDSGA detects masquerade attacks in the cloud by aligning the sequence of the current session to the previous sequences of the same user. Then, it labels misalignment areas as anomalous and the presence of several anomalous areas is a strong indicator of a masquerade attack. DDSGA tolerates small mutations in user command sequences by allowing small changes in the low-level representation of the commands functionality. It also tolerates changes in user behaviour by updating the user signatures according to the current behaviours. We show that DDSGA improves both accuracy and performance with respect to current solutions. We apply three detection strategies based on DDSGA to detect masquerade attacks in clouds. The first one applies DDSGA to sequences of correlated audits from the VMs operating systems. This strategy is applied independently to two kinds of audits, system calls and

security events. The second strategy analyzes NetFlow data from the network environment. The third strategy improves the overall detection by integrating the first two strategies through a neural network. The evaluation has also considered the correlation of the behavior of the same user in distinct cloud nodes. We have evaluated three alternative correlation models through both CIDS and CIDS-VERT: Audit Exchange, Independent, and Centralized-Backup. These models are detailed in Chapter 3 that shows how the Independent model is the ideal one in combination with CIDS for small and private cloud, while the Centralized-Backup model is the most suitable one in combination with CIDS-VERT for large clouds such as public and hybrid ones.

Finally, the third contribution is related to signature based detection. We introduce a hierarchical architecture that supports two deployments, a distributed and a centralized one for the proposed IDS. The deployments use host based and network based IDSs that exploit signature based analysis techniques. The hierarchical architecture overcomes some limitation of current IDSs and can efficiently detect host, network, and DDoS attacks. Furthermore, we discuss the integration and the correlation of alerts that come from all detection components.

We briefly outline the content of the various chapters:

Chapter 1: Background and Main Concepts.

It introduces the main concepts underlying the thesis together with DDoS attacks and the related work on their detection in clouds. Finally, it outlines the software tools that the thesis uses.

Chapter 2: Intrusion Detection and Related Works.

It introduces the definition, architecture and techniques of intrusion detection systems, and surveys the previous works on intrusion detection, masquerade detection techniques, and the current intrusion detection datasets.

Chapter 3: CIDS and CIDS-VERT Frameworks and their Correlation Models.

It describes the components, architecture, testbed, pros and cons of both CIDS and CIDS-VERT frameworks. Furthermore, it discusses the proper

framework for each cloud deployment model. Finally, it details the three correlation models, Audit Exchange, Independent, and Centralized-Backup.

Chapter 4: Cloud Intrusion Detection Dataset (CIDD).

After discussing the major challenges to build a cloud dataset, it introduces the Log Analyzer and Correlator System (LACS) that helps in building CIDD dataset. Then it describes in details the distribution of attacks and masquerades in CIDD and compares CIDD against other publicly available datasets.

Chapter 5: Data-Driven Semi Global Alignment (DDSGA).

It introduces the DDSGA approach and describes its three main phases namely, configuration, detection, and update. It outlines the modules and experimental results for each phase and compares DDSGA against other approaches.

Chapter 6: Detecting Masqueraders through System Calls and NetFlow Data.

It introduces three strategies to detect masquerade attacks. The first one is based on sequences of system calls audits. We outline how it builds a consistent user profile from the features extracted from the system calls. Then, we adapt DDSGA to the extracted features by considering the CIDS and CIDS-VERT frameworks using the three correlation models in Chapter 3. The second strategy is based on NetFlow data. We outline the main features extracted from the NetFlow data and the adaption of DDSGA to these features. The third detection strategy integrates the other two through a neural network. After evaluating the three strategies in isolation, the chapter compares their accuracy and performance.

Chapter 7: Detecting Masqueraders through Security Events and NetFlow Data.

Also this chapter considers the three strategies of Chapter 6 but with different audit data and distinct operating system. The first detection strategy uses sequences of security events. We outline the main features extracted from the security events and how DDSGA is adapted to these features by considering the CIDS and CIDS-VERT frameworks with the Independent and Centralized-Backup correlation models respectively. Lastly, we evaluate the efficiency and the computational performance of this strategy. The

second and third strategies are evaluated as in Chapter 6. Finally, this chapter compares the strategy based on security events and the one that uses system calls introduced in Chapter 6.

Chapter 8: Efficient Deployments of HIDS and NIDS using A Hierarchical Architecture of CIDS-VERT

It discusses the detection of host, network, and DDoS attacks by signature based techniques. It introduces a hierarchical architecture of CIDS-VERT framework that supports two deployments: Distributed and Centralized one, and outlines their relative advantages. Furthermore, it discusses the approaches to integrate, correlate, and summarize distinct alerts from the signature based techniques i.e., HIDS and NIDS. Finally, this chapter shows some experimental results and evaluates the accuracy of the proposed deployments.

Conclusion and Future Work:

This section draws conclusion remarks and outlines future work.

Chapter 1

Background and Main Concepts

This chapter introduces the main concepts underlying the thesis namely the definition of cloud computing systems from the unique perspectives of IT network and security. Furthermore, we introduce the cloud essential characteristics and deployment and service models. We also present cloud security definition, the risks a cloud user should assess before committing and top threats to clouds. We also present some miscellaneous concepts regarding Virtual Machines (VM), Virtual Machine Monitors (VMM), system calls, security events, NetFlow, Entropy, Threshold Logic Unit (TLU), and the host based, network based attacks. Finally, this chapter discusses DDoS attacks and their detection in cloud systems and outlines the software tools we use.

1.1. Cloud Computing

Each of the current definitions of cloud systems addresses cloud systems from a distinct perspective. Here we assume the perspectives of IT network and security.

According to NIST (National Institute of standards and Technology) [2], "Cloud computing ('cloud') is an evolving term that describes the development of several existing technologies and approaches to computing into something different. Cloud separates application and information resources from the underlying infrastructure, and the mechanisms used to deliver them". According to Ian Foster et al. [4], "Cloud computing is a large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet". There are a few key points in this definition. First, cloud computing is a specialized distributed computing paradigm; it differs from traditional ones in that

1) It is massively scalable.
2) Can be encapsulated as an abstract entity that delivers different levels of services to customers outside the cloud.

3) It is driven by economies of scale, and 4) the services can be dynamically configured via virtualization or other approaches and delivered on demand.

NIST defines cloud computing in terms of five essential characteristics, three service models, and four deployment models. They are summarized in visual form in Figure 1.1 and explained below as in [2, 4].

1.1.1. Essential Characteristics of Cloud Computing

Cloud services exhibit five essential characteristics that demonstrate their relation to, and differences from, traditional computing approaches [2]:

- **On-demand self-service.** A consumer can unilaterally provision computing capabilities as needed and automatically, without human interaction with a service provider.

- **Broad network access.** Computing capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g. mobile phones, laptops, and PDAs) as well as other traditional or cloud based software services.

- **Resource pooling.** A provider pools computing resources to serve several consumers using a multi-tenant model, which dynamically assigns and reassigns physical and virtual resources according to consumer demand. There is a degree of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources.

- **Rapid elasticity.** Capabilities can be rapidly and elastically provisioned, in most cases automatically, and rapidly released to quickly scale out and scale in. For a consumer, the capabilities appear to be unlimited and can be purchased in any quantity at any time.

- **Measured service.** Cloud systems automatically control and optimize resource usage by leveraging a metering capability according to the type of service. Usage can be monitored, controlled, and reported, providing transparency for both the provider and the consumer.

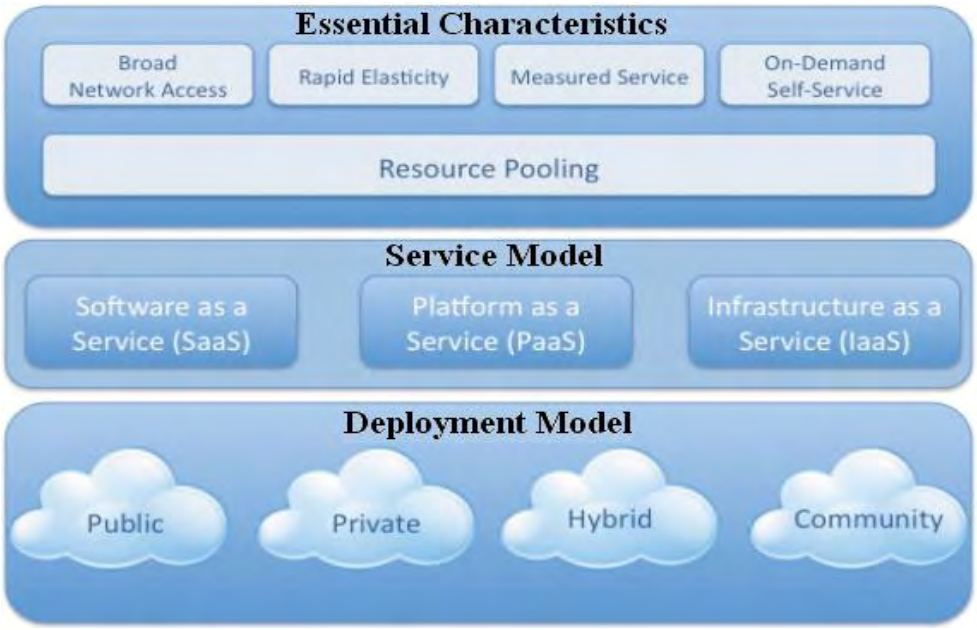


Figure 1.1: NIST Visual Model of Cloud Computing Definition

1.1.2. Cloud Service Models

In general, clouds offer services at three different levels [4]: IaaS, PaaS, and SaaS. However, some providers can expose services at multiple levels.

- **Software as a Service (SaaS)** delivers software that is remotely accessible by consumers through the Internet with a usage-based pricing model. E.g., Live Mesh from Microsoft allows files and folders to be shared and synchronized across multiple devices.
- **Platform as a Service (PaaS)** offers a high-level integrated environment to build, test, and deploy custom applications as in Google's App Engine [7]. Inside this layer resides the middleware system, a portable component for both grid and cloud systems. Examples include WSO2 Stratos [5], Windows Azure [6], and our middleware HIMAN [8, 9, and 10].

- **Infrastructure as a Service (IaaS)** provisions hardware, software, and equipments to deliver software application environments with a resource usage-based pricing model. Infrastructure can scale up and down dynamically based on application resource needs. Typical examples are Amazon EC2 (Elastic Cloud Computing) Service [11], Eucalyptus [12], Microsoft Private Cloud [13].

1.1.3. Cloud Deployment Models

There are four deployment models for cloud services, with derivative variations that address specific requirements:

(1) **Public Cloud.** The cloud is made available to the general public or a large industry group and is owned by an organization selling cloud services.

(2) **Private Cloud.** The cloud is operated solely for a single organization. It may be managed by the organization or by a third party, and may exist on-premises or off- premises.

(3) **Community Cloud.** The cloud is shared by several organizations to support a specific community that has shared concerns. It may be managed by the organizations or by a third party and may exist on-premises or off-premises.

(4) **Hybrid Cloud.** The cloud infrastructure consists of two or more clouds (private, community, or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability.

1.2. Cloud Computing Security

Cloud computing may adopt the same control of any IT environment. However, the cloud service models, the operational models, and the supporting technologies change the risk landscape for an organization with respect to traditional IT. The next section outlines seven risks a user should consider before committing and seven top threats to cloud computing systems.

1.2.1. Seven Risks to be analyzed before Committing

There are seven possible risks a user should assess before committing [14]:

- Privileged user access: sensitive data should be processed outside the enterprise only with the assurance that they are only accessible and propagated to privileged users.
- Data segregation: is the user data should be fully segregated from data of other users.

- Regulatory compliance: a cloud provider should have external audits and security certifications and the infrastructure should comply with regulatory security requirements.
- Data location: the cloud provider should commit to storing and processing data in specific jurisdictions and to obey local privacy requirements on behalf of the customer;
- Recovery: the provider should offer an efficient replication and recovery mechanism to fully exploit the potentials of a cloud in the event of a disaster;
- Investigative support: support should to be ensured for forensics and investigation with a contractual commitment.
- Long-term viability: a user data should be accessible even when the provider is acquired by another company or the user moves to another provider.

1.2.2. Top Seven Threats to Cloud Computing

We briefly highlight seven threats that CSA (Cloud Security Alliance) [1] ranks and that apply across all of the different cloud computing models.

Threat #1: Abuse and Nefarious Use of Cloud Computing

The top threat that CSA identifies is the abuse and nefarious use of cloud computing. This is related to the use of botnets to spread spam and malware. Attackers can infiltrate a cloud system, by abusing the relative anonymity behind the cloud registration system and usage models. Then, they can upload malware and use the power of the cloud to attack other machines. The CSA suggests to:

1. Monitor public blacklists for one's own network blocks.
2. Use a stricter initial registration and validation processes.
3. Enhanced credit card fraud monitoring and coordination.

Threat #2: Insecure Interfaces and APIs

The CSA cautions against unsure application programming interfaces between applications for interoperability. The CSA suggests to:

1. Analyze the security model of cloud provider interfaces.
2. Ensure strong authentication and access controls are implemented in concert with encrypted transmission. Some Grid and Cloud portals

can be used for this target e.g. Nubifer [15], Ubuntu Portal [16], and our HIMAN-GP [17].
3. Understand the dependency chain associated with the API.

Threat #3: Malicious Insiders

Organizations need to assess the risk on the service provider's end and demand segregation of duties to prevent a malicious insider from accessing data. The CSA suggests to:

1. Enforce strict supply chain management and conduct a comprehensive supplier assessment.
2. Specify human resource requirements as part of legal contracts.
3. Require transparency into overall information security and management practices, as well as compliance reporting.
4. Determine security breach notification processes.

Threat #4: Shared Technology Issues

Cloud users have to be aware of vulnerabilities in shared technologies, such as VMs, communications systems or key management technologies. A zero-day attack can use these technologies and quickly spread across a public cloud and expose all data within it. The CSA suggests to:

1. Implement security best practices for installation/configuration.
2. Monitor environment for unauthorized changes/activity.
3. Promote strong authentication and access control for administrative access and operations.
4. Enforce service level agreements for patching and vulnerability remediation.
5. Conduct vulnerability scanning and configuration audits.

Threat #5: Data Loss or Leakage

There are several alternative ways to compromise data. Deletion or alteration of records without a backup is an obvious example. A cloud increases the risk of data compromise, due to risks and challenges which are either unique to cloud, or more dangerous because of the architectural or operational characteristics of a cloud environment.

The CSA suggests to:

1. Implement strong API access control.
2. Encrypt and protect integrity of data in transit. There are many encryption schemes for high performance systems e.g., GridCrypt [18] and our "Ultra GridSEC" [19, 20, 21].
3. Analyzes data protection at both design and run time.
4. Implement strong key generation, storage and management, and destruction practices.
5. Contractually demand providers wipe persistent media before it is released into the pool.
6. Contractually specify provider backup and retention strategies.

Threat #6: Account or Service Hijacking

Cloud users need to be aware of account service and traffic hijacking. Examples for attacks that may cause these threats are: man-in-the-middle, phishing, spam campaigns, and DDoS. Cloud solutions add a new threat to the landscape. If an attacker gains access to a user credentials, then she can eavesdrop on activities and transactions, manipulate data, return falsified information, and redirect the user clients to illegitimate sites. The CSA suggests to:

1. Prohibit the sharing of account credentials between users and services.
2. Leverage strong two-factor authentication techniques where possible.
3. Employ proactive monitoring to detect unauthorized activity.
4. Understand cloud provider security policies and SLAs.

Threat #7: Unknown Risk Profile

One of the tenets of cloud computing is the reduction of hardware and software ownership and maintenance costs to allow companies to focus on their core business strengths. This has clear financial and operational benefits, which must be weighed carefully against the contradictory security concerns when the migration to a cloud is driven by expected saving only by groups who may lose track of security issues. Information about who is sharing an infrastructure may be pertinent, in addition to network intrusion logs, redirection attempts and/or successes, and other logs. An IDS is the ideal tool for this threat, as it can deal with all suggestions of CSA like:

1. Disclosure of applicable logs and data.
2. Partial/full disclosure of infrastructure details (e.g., patch levels, firewalls, etc.).
3. Monitoring and alerting on necessary information.

1.3. Virtual Machines

A virtual machine (VM) is as an efficient and isolated duplicate of a real one [22]. Typical applications of VMs include the development and testing of new operating systems, simultaneously running distinct operating systems on the same machine, and server consolidation [23].

A "virtual machine" is a fully protected and isolated copy of the underlying physical machine's hardware that gives to its users the illusion of a dedicated physical machine. Figure 1.2 illustrates the traditional organization of a virtual machine system. The virtual machine monitor, VMM, is a software layer that takes complete control of the machine hardware and creates VMs, each of which behaves like a complete physical machine with its own operating system (OS).

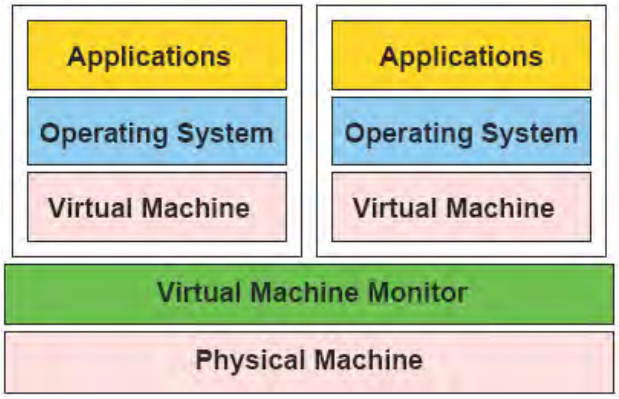


Figure 1.2: A virtual machine monitor

To maximize performance, the VMM gets out of the way whenever possible and allows a VM to execute directly on the hardware, albeit in a non-privileged mode. The monitor regains control anytime the VM tries to perform an operation that may affect the correct operation of other VMs or of the hardware. The monitor safely emulates the operation before returning control to the VM. The result of a complete machine virtualization is the creation of a set of virtual

computers that runs on a physical computer. An OS that runs in a VM is a guest OS. Since VMs are isolated from each other, the crash of a guest OS does not affect other VMs. [23]

VMMs build some useful properties for system security, among them [24, 25]:

- Isolation: Software running in a VM cannot access or modify the monitor or other VMs.
- Inspection: The VMM can access the entire VM state.
- Interposition: The VMM can intercept and modify operations issued by a VM.

There are two classical approaches to organize VMs [26, 27]:

- A type II VMM runs on top of a hosting operating system and then spawns higher level virtual machines. Examples include the JavaVM, Dot Net environment, Virtualbox [28] and Hosted Xen project (HXen) [29]. These VMMs monitor their VMs and redirect requests for resource to appropriate APIs in the hosting environment. Figure 1.3-A depicts type II VMM.
- A type I VMM, or hypervisor runs directly on the hardware without the need of a hosting OS. Examples include the mainframe virtualization solutions offered by Amdahl and IBM, and on modern computers by solutions like VMware ESX [30], Xen [31] and Windows virtualization. Figure 1.3-B depicts type I VMM.

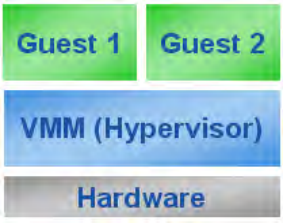


Figure 1.3-A: type II VMM. Figure 1.3-B: type I VMM (Hypervisor)

1.4. System Calls

A system call [32] is a request for an action of an OS on behalf of a user program. System calls provide an essential interface between a process and

the OS like the API. On Unix-like systems, this API is usually part of an implementation of the C library (libc). On Windows systems, it is part of the Native API. Several tools can record the system calls sequences, the most common ones are the UNIX tool "strace" and the Linux audit daemon "auditd". System calls can be categorized into five main categories [32]:

1. Process Control
2. File management:
3. Device Management:
4. Information Maintenance:
5. Communication:

1.5. The Event Logs

An event is a notification to the user or an entry added to a log that denotes any significant occurrence in hardware, software, and system components of a local or a remote computer [33]. The event log service records application, security, and system events. Event logs support the prediction, the identification and the diagnosis of system problems. Monitoring security events helps in detecting attacks and threats. Each OS has a specific auditing or event log service. In Windows systems it is the "Event Viewer" service [33]. The log entry consists of two parts, (a) the header information and (b) event description.

a) Event Header:

This header records [33]:

- Date: The date the event occurred.
- Time: The time the event occurred.
- User: The name of the logged user when the event occurred.
- Computer: The name of the computer where the event occurred.
- Event ID: An event number that identifies the event type.
- Source: The source of the event. This can be the name of a program, a system component, or an individual component of a large program.
- Type: The type of event.
- Category: A classification of the event by the event source.

b) Event Description

The description of an event depends on its type. Events can be classified into one of the following types [33]:

- Information: It describes the successful operation of a task, For example, an Information event is logged when a network driver loads successfully.
- Warning: It does not imply an urgent necessity but it may indicate a future problem. For example, a Warning message is logged when disk space starts to run low.
- Error: It signals a significant problem, such as the failure of a critical task. For example, the startup of an important process failed.
- Success Audit (Security log): It records the successful execution of a security action. For example, a user logs onto/off the computer.
- Failure Audit (Security log): It signals a partial failure of a security action. For example, a user cannot log onto the computer.

Chapter 6 details security events and their important features.

1.6. NetFlow Data (Network Flows)

NetFlow is a network protocol developed by Cisco Systems to collect network flows. A network flow is a unidirectional sequence of packets that share seven values [34, 35]:

- Ingress interface.
- Source IP address
- Destination IP address
- IP protocol
- Source port for UDP or TCP, 0 for other protocols
- Destination port for UDP or TCP, type and code for ICMP, or 0 for other protocols
- IP Type of Service (TOS). Based on TOS values, a packet would be placed in a prioritized outgoing queue, or take a route with appropriate latency, throughput, or reliability.

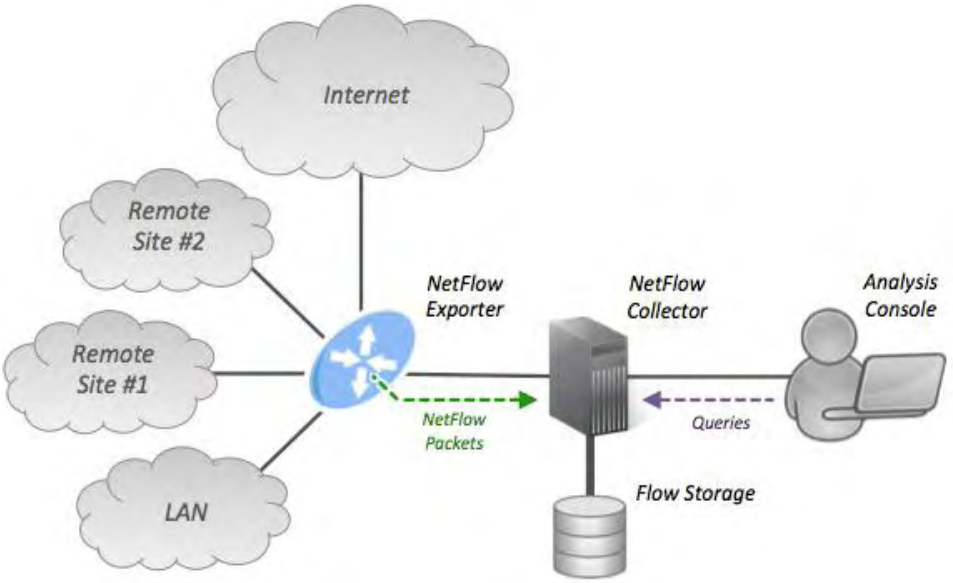


Figure 1.4: NetFlow architecture [34]

Figure 1.4 shows the export of NetFlow records. The router outputs a flow record when it determines that the flow is finished. It does this by flow aging: when the router sees new traffic for an existing flow it resets the aging counter. Flow record can be transmitted at a fixed interval even if the flow is still ongoing. Several software tools support NetFlow recording such as, Cisco NetFlow [36], vSphere [37], and sFlow [38]. A NetFlow record can contain a wide variety of information about the flow traffic such as [34]:

- Input interface index.
- Output interface index or zero if the packet is dropped.
- Timestamps for the flow start and finish time, in milliseconds since the last boot.
- Number of bytes and packets observed in the flow
- Layer 3 headers:
  - Source & destination IP addresses
  - Source and destination port numbers for TCP,UDP, SCTP
  - ICMP Type and Code.
  - IP protocol
  - Type of Service (ToS) value
- For TCP flows, the union of all TCP flags observed over the life of the flow.

- Layer 3 Routing information:
  - IP address of the immediate next-hop along the route to the destination
  - Source & destination IP masks.

1.7. Entropy

The definition of the entropy is a quite broad and general and it is expressed in terms of the application field. In information theory, entropy measures the amount of information that is missing before reception and it is also referred to as Shannon entropy [39]. The conditional entropy (or equivocation) [40] is one of the information entropy categories that quantifies the amount of information to describe the outcome of a random variable *Y* if the value of another random variable *X* is known. The entropy of *Y* conditioned on *X* is written as *H(Y|X).* If *H(Y|X=x)* is the entropy of the variable *Y* conditioned on the variable *X* taking the value *x*, then *H(Y|X)* is the average of *H(Y|X=x)* over all possible values of *X*. Equation 1.1 [40] formally defines conditional entropy given a discrete random variable *X* with support $\mathcal{X}$ and *Y* with support $\mathcal{Y}$.

H(Y|X) = 0 if and only if the value of *Y* is completely determined by the value of *X*. Conversely, *H(Y|X)* = *H(Y)* if and only if *Y* and *X* are independent.

$$
\begin{aligned}
H(Y|X) &\equiv \sum_{x \in \mathcal{X}} p(x)\, H(Y|X=x) \\
&= \sum_{x \in \mathcal{X}} p(x) \sum_{y \in \mathcal{Y}} p(y|x)\, \log \frac{1}{p(y|x)} \\
&= -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x,y)\, \log p(y|x) \\
&= -\sum_{x \in \mathcal{X}, y \in \mathcal{Y}} p(x,y) \log p(y|x) \\
&= \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} p(x,y) \log \frac{p(x)}{p(x,y)}.
\end{aligned}
\tag{1.1}
$$

Chapters 6 and 7 use conditional entropy to measure the regularity of the training data for each user.

1.8. The Artificial Neural Network (ANN)

An ANN [41] is a mathematical function that consists of some artificial neurons that receives and sums their inputs. Usually the sums are weighted, and the sum is passed through a non-linear function, a transfer or activation function.

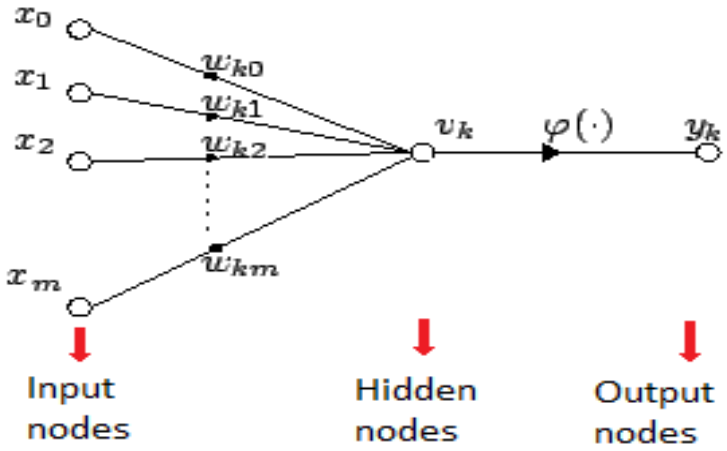


Figure 1.5: The basic structure of the artificial neuron

As shown in Figure 1.5, the basic structure of the ANN consists of three types of nodes, input, hidden, and output. The artificial neuron receives m + 1 inputs with signals x_0 through x_m and weights w_0 through w_m. Equation 1.2 defines the output of the kth neuron:

$$y_k = \varphi\left(\sum_{j=0}^{m} w_{kj} x_j\right) \tag{1.2}$$

Where, φ is the transfer function that translates the input signals to output signals.

1.8.1. ANN Transfer Function

Four types of transfer functions are commonly used, Unit step (threshold), sigmoid, piecewise linear, and Gaussian.

1) Unit step (threshold) function:

The output is one of two values depending on whether the total input is larger than a threshold x. Figure 1.6 shows the shape of this function.

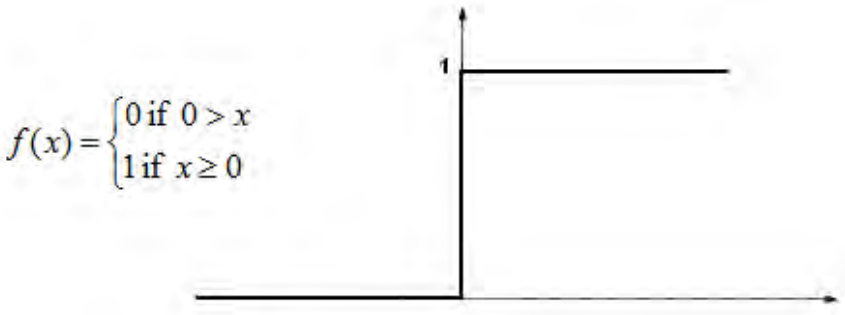


Figure 1.6: The Unit step (threshold) transfer function

2) Sigmoid function:

The sigmoid function consists of 2 functions, logistic and tangential. The logistic function has a range 0..1, while the range of the tangential one is -1.. +1. Figure 1.7 shows the shape of this function.

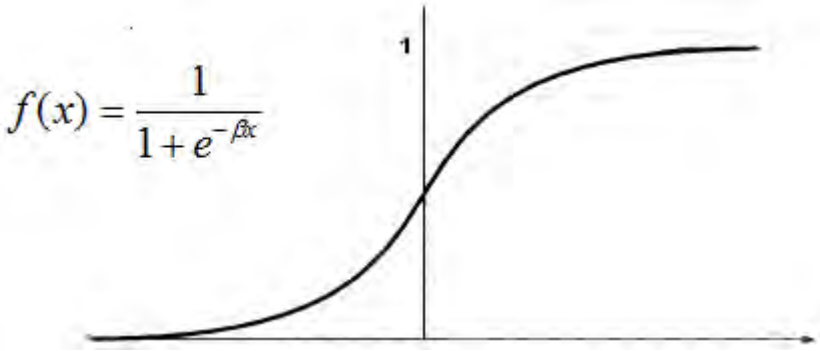


Figure 1.7: The sigmoid transfer function

3) Piecewise Linear function:

The output of the Piecewise Linear function is proportional to the total weighted output. Figure 1.8 shows the shape of this function.

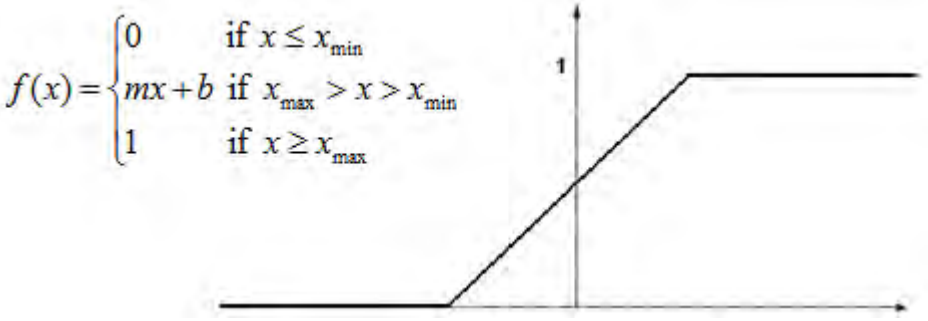


Figure 1.8: The Piecewise Linear transfer function

4) Gaussian function:

Gaussian functions are continuous curves with a bell shape. The node output is interpreted in terms of class membership (1/0), depending on how close the net input is to a chosen average value. Figure 1.9 shows the function shape.

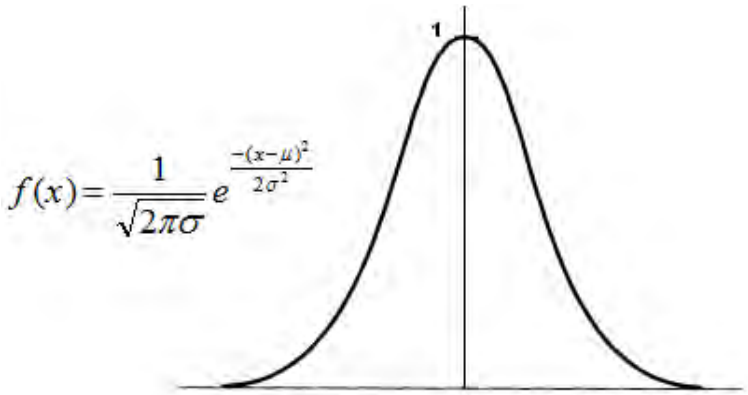


Figure 1.9: The Gaussian transfer function

1.8.2. Threshold Logic Unit (TLU):

TLU [42] is a simple type of ANN similar to the threshold transfer function that uses a simple model with binary inputs and outputs, some restrictions on the possible weights, and a more flexible threshold value. Any boolean function can be implemented by networks of such ANN.

1.8.3. ANN Types

There are different types [41] of neural networks, but they are generally classified into feed-forward and feed-back networks.

A feed-forward network is a non-recurrent network where the signal travels in one direction. Input data is passed onto a layer of processing elements where each element computes a weighted sum of its inputs that feed the next layer. The process continues through all the layers to compute the final output. The output layer sometime uses a threshold transfer function.

A feed-back network has feed-back paths that send the signal in both directions using loops. All connections between the neurons are possible and this may result in loops. Because of these characteristics, this type of networks is a non-linear dynamic system which changes continuously until it reaches an equilibrium. Feed-back networks are often used in associative

memories and optimization problems. We use this type of networks to train our IDS system to adjust the weights of the ANN.

1.9. Host, Network, and DDoS Attacks

Attacks utilize network media and manipulate computing and/or network resources to severely degrade the performance of the services of an ICT network 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. [43, 44]

Penetration Type: Penetration can be carried out as an outsider or as an insider. Insiders are legal users that are conducting malicious activities through their accounts or by 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 network before launching 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 or passive. 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 this type of attacks are hard to observe because they leave the minimum footprint. Active attacks aim 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 behavior 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: It prevents services for the users by limiting or denying their access to system 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 proper 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 attacks. Section 1.9.2 describes a popular attack in this class, the Distributed Denial of Service (DDoS).

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 spreads across a network and targets hosts or network resources to cause dysfunction, data loss, or data theft.

Attacks against an information system can also be classified according to the number of involved computers. An attack that may involve even a large number of computers is the DDoS ones outlined in Section 1.9.2. Attacks can also be classified into network or host ones according to the mechanism or the type of vulnerabilities they exploit.

[45] presents classification criteria based on attack surfaces of the cloud computing scenario participants as shown in Figure 1.10.

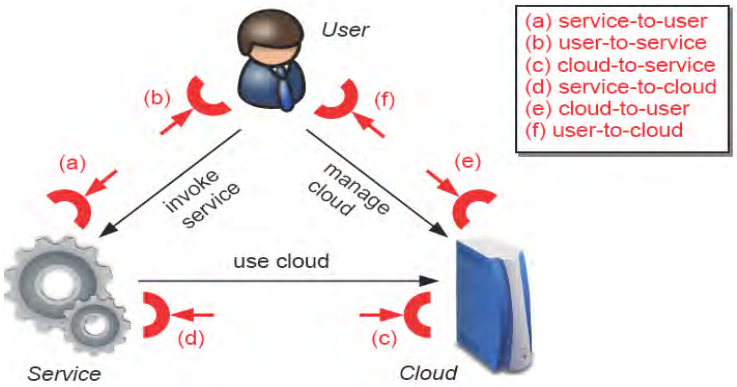


Figure 1.10: A Taxonomy for attacks on cloud services

(a) Service-to-User: includes attacks in common client-server architectures, e.g. buffer overflow attacks, SQL injection, or privilege escalation.
(b) User-to-Service: includes attacks in the common environment of client program, e.g. browser-based attacks, attacks on browser caches, or Phishing attacks on mail clients.
(c) Cloud-to-Service: includes attacks of a service instance against its hosting cloud system, e.g. the resource exhaustion attacks, or attacks on the hypervisor.
(d) Service-to-Cloud: incorporates attacks of a cloud provider against a service, e.g., availability reductions, privacy related attacks or even malicious interference. This category is by far the most critical one, as the provider can implement them in a rather simple way and attack impacts are tremendous.
(e) Cloud-to-User: includes user attacks against the interface of the cloud system to control the provided services and that enables the customers to add new services or change the number of service instances.
(f) User-to-Cloud: involves every kind of attack that targets a user and that originates from the cloud system. It is similar to the phishing attempts to trigger a user into manipulating cloud-provided services.

1.9.1. Host and Network Attacks and Their Libraries

We briefly classify attacks into network and host ones and describe the libraries that support their implementation.

Network attacks exploit vulnerabilities in the communication protocols or in the interconnection structure to attack the integrity and confidentiality of communications. As an example, since most communications adopt an unsecured or clear text format, an attacker that can access network data paths can also read and interpreter the traffic these paths transmit. Some examples of these attacks are [43]:

(1) Eavesdropping: it is also known as sniffing or snooping. This attack monitors the network traffic.
(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 builds IP packets that appear to originate from valid addresses to modify, reroute, or delete some data. It is supported by specialized libraries.

(4) Denial-of-Service Attack (DoS): It shuts down applications or network services by flooding them with invalid traffic. This can prevent legal user from accessing network resources.

(5) Man-in-the-Middle Attack: This attack inserts a distinct entity between two communicating components to capture and modify their communications.

Host based attacks are enabled by vulnerabilities in the host OS or in the applications. Some classes of these attacks are [Host-attack]:

(1) Buffer overflow: It violates memory safety to overwrite adjacent memory positions. It exploits the lack of controls on the size of a parameter

(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: It can crash a program or execute harmful code. It exploits the lack of control on user inputs such as the format string in some C functions.

Several libraries have been developed to support host and network attacks. As an example, Metasploit [46] is a consistent and reliable library of constantly updated exploits for network, OSs and applications. An exploit is a code fragment to automate, at least partially, an attack. Metasploit defines a complete environment to develop new tools and automate every aspect of an attack. It simplifies the development of attack vectors to extend its exploits, payloads, encoders to create and execute more advanced and specialized attacks against a target system.

1.9.2. DDoS Attacks

Distributed Denial of Service (DDoS) attacks [47] are a class of attacks that disrupt the service quality of a system. It is worth considering these attacks in relation with clouds because their effectiveness increases if an attacker can use the massive amount of resources in a cloud.

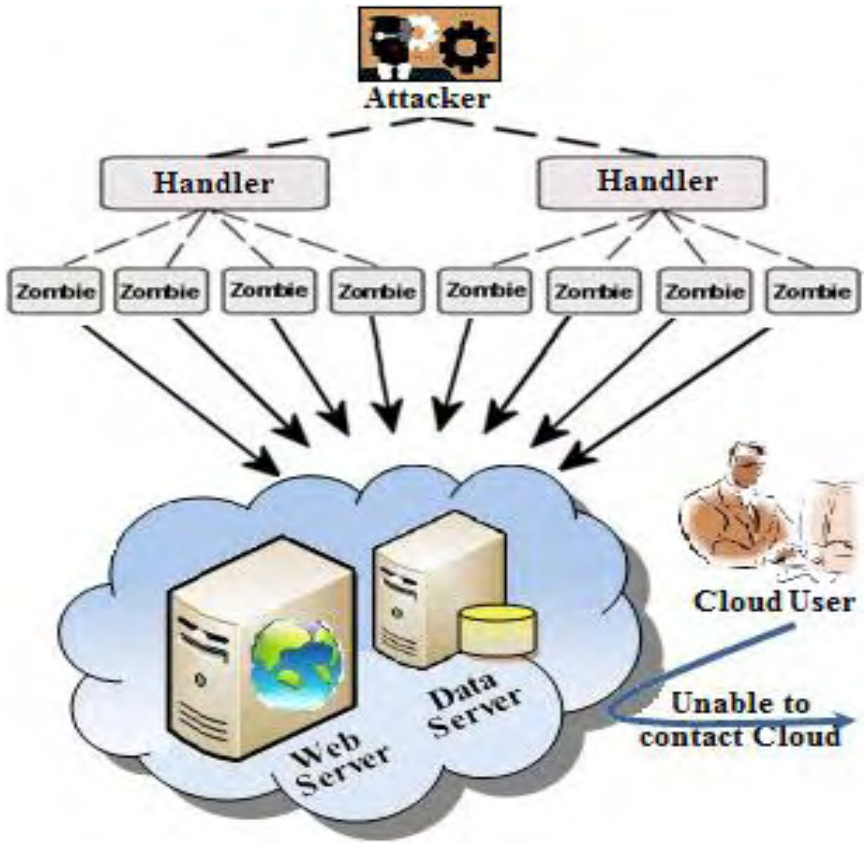


Figure.1.11: The DDoS Strategy.

Figure 1.11 shows the four elements of DDoS attacks [48] namely:

(1) The attacker machine.

(2) The handlers: these are hosts controlled by the attacker as a result of a previous attack. They run some malware and act as an intermediate interfaces to control the agents and route to them the attacker commands.

(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.

While several kinds of DDoS attacks exist, any implementation of these attacks includes the following stages:

(1) Search of vulnerable hosts to act as handlers and zombies. This step can exploit a standard vulnerability scanner such as Nessus [51].

(2) Compromising the vulnerable hosts: The attacker exploits the vulnerabilities returned by the scanner to attack some vulnerable hosts and stealthy install some malware.

(3) Communication, broadcasting, and flooding: The attacker communicates a command to one or more handlers. Then, 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 sends a stop command.

In the experiments, we implement DDoS attacks through the LOIC library [49]. LOIC is one of the most powerful free DOS and DDOS attacking tool, it attempts to open several connections to the same target host and continuously floods it with false TCP or UDP packets, or with HTTP requests that lead to a service disruption. A DDOS attack runs LOIC through multiple zombies. Another library we have used is the CPU Ping Death library [50]. It 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.

1.9.3. Current DDoS Detection Techniques in Cloud systems.

We briefly review some IDSs that have recently been proposed to detect DDoS attacks in clouds.

[52] investigates the effect of DDoS on clouds and proposes an IDS based on the behavioral threshold. The IDS assumes that a user is attacking the system if the user requests are outside the normal user range. The threshold is automatically determined as a dynamic variable according to 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 [53]. The latter discovers the attacker signatures by analyzing the collected data. 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 the accuracy and the performance of the IDS are not evaluated.

[54] uses an IDS sensor such as the version of Snort [55] installed on VMware ESX [56] machine that sniffs both in-bound and out-bound traffic to detect DoS attacks. Snort analyzes in-bound packets and looks for several intrusion patterns. If at least one matches, it drops all the packets from the same IP address. The accuracy and performance of this solution is not evaluated. Furthermore, also this solution does not correlate network events to discover attacks against several virtual zones.

[57] proposes a cooperative IDS that reduces the impact of DoS attack in each cloud regions. Several IDS components are distributed across these regions and a cooperative module receives their alert messages. Then, a majority vote determines the trustworthiness of these alerts. 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 solutions confirms that a defense strategy for clouds against DDoS attacks introduces some further requirements with respect to those for traditional systems. To be adopted in clouds, a solution needs to:

(1) Be distributed and scalable,

(2) Avoid single points of failure,

(3) Correlate the user behaviours in distinct environments.

(4) Integrate different service models.

1.10 Software Tools Used in the Thesis Work

In the following, we highlight the software tools to build and deploy the proposed framework.

A. Cloud Management Software

In our practical deployments, we used Microsoft Private Cloud [13] in our CID-VERT framework and VMware system in CIDS framework to control the deployment of the VMs and applications and to manage the creation of the virtual networks.

1) Microsoft Private Cloud

Microsoft private cloud [13] offers traditional IaaS services, such as VMs on demand and supports for deploying multi-tier applications, monitoring and updating those applications, and automation services. It relies on several different System Center 2012 components and supports multiple type-1 hypervisors, see Figure 1.12, such as: Microsoft Hyper-V [13], VMware ESX/ESXi [56], and Citrix XenServer [58]. It also supports conventional compute, storage, and networking hardware along with pre-packaged hardware configurations that conform to the Hyper-V Cloud Fast Track specification. [59]

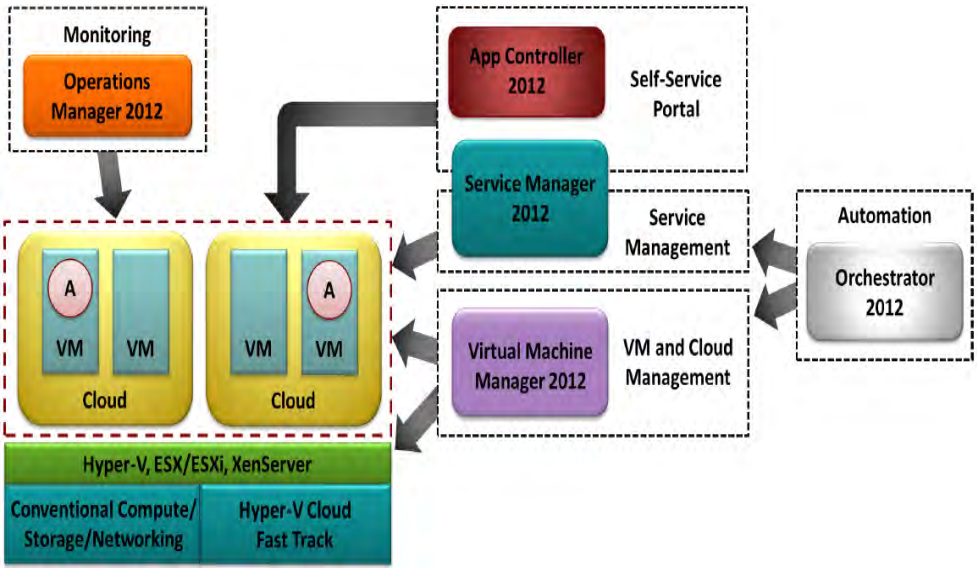


Figure 1.12: The main components of Microsoft private cloud. [59]

The components that Microsoft private cloud relies on are [59]:

(1) System Center Virtual Machine Manager (VMM) 2012: it provides the fundamental services for creating and managing clouds as well as to deploy and update VMs and applications.

(2) System Center App Controller 2012: it is a self-service portal for requests made directly to a private cloud created with VMM 2012.

(3) System Center Service Manager 2012: it provides automated IT service management and an associated self-service portal.

(4) System Center Orchestrator 2012: it provides a way to automate interactions among other management tools such as VMM 2012 and Service Manager.

(5) System Center Operations Manager 2012: it monitors VMs, applications, and other aspects of a private cloud. Then, it fires actions to fix problems it detects.

All these technologies depend on Windows Server 2012 and Active Directory.

2) VMware Workstation

VMware Workstation [60] is a type-2 hypervisor that enables users to set up multiple VMs that supports the following functions [60]:

(1) Bridging existing host network adapters.

(2) Share physical disk drives and USB devices with a virtual machine.

(3) Simulate disk drives.

(4) Save "snapshots" for the VMs which can later be restored to return the virtual machine to the saved state.

B. Intrusion Detection Software

In our deployment, we used some open source IDSs and tools to detect host, network, and DDoS attacks based on the signature based analysis techniques namely, OSSEC, Snort, and OSSIM.

1) OSSEC

OSSEC [61] is an Open Source Host-based Intrusion Detection System. It performs log analysis, file integrity checking, policy monitoring, rootkit detection, real-time alerting and active response. It runs on most OSs and it has two types of installation, Local and Agent-Server. In the local installation, OSSEC only protect a local machine. Instead, the Agent-Server installation protects the machines of the network. The agents are installed in several hosts systems to report back to a central OSSEC server to aggregate the information from the agents, analyze it and fires alerts.

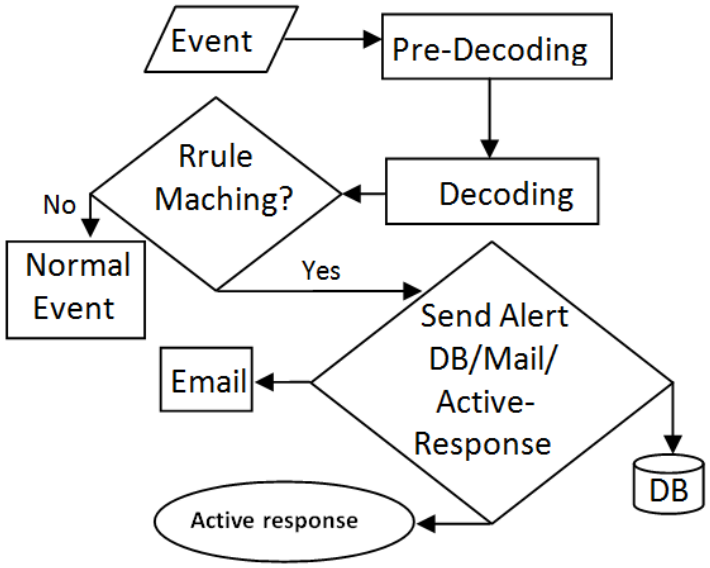


Figure 1.13: The Analysis flow chart of OSSEC

As shown in Figure 1.13, the analysis processes includes pre-decoding, decoding, rule matching, and alerting.

- The Pre-decoding process extracts the static information such as the event message, the location or the program name.
- The Decoding process extracts non static information such the attributes of the regular expression that defines each field.
- The Rule Matching process applies the Rule Matching Engine to determine if the received event matches any stored rules to fire an alert.
- The Alerting process determines where the rules should be sent. Alerts can be emailed to the user or logged into database.

2) Snort

Snort [55] is an open source network intrusion detection system that can log network packets. It uses a rule-based language that integrates signature, protocol, and anomaly inspection methods. Snort consists of five main components [55], see Figure 1.14, namely, Packet Decoder, Preprocessors, Detection Engine, Logging and Alerting, and the output module.

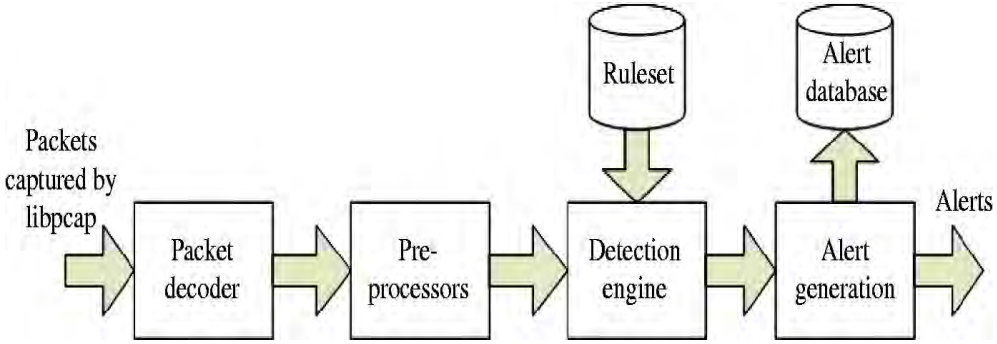


Figure 1.14: Snort Architecture

The analysis processes of Snort are summarized below:

(1) It captures packets from network using "LibPCap" component.

(2) The packet decoder component receives packets from different types of network interfaces (Ethernet, SLIP, PPP…), prepares a packet for processing and fits it at the data structure.

(3) The preprocessor component prepares data for the detection engine. It also handles defragmentation and TCP streams and detects anomalies in packet headers.

(4) The detection engine, the most important component, detects if any intrusion activity exists in a packet by applying a rule-based string matching algorithm. The algorithm dissects the packet and applies rules on different parts of the packet. If a packet matches any rule, appropriate action is taken. Otherwise no action is taken.

Finally, the Output Module processes alerts and logs and generates the final output according to the user policy and the packet content.

3) OSSIM

OSSIM [62] provides a common framework for the deployment, configuration, and management of security tools including IDS sensors. It offers event collection, normalization, correlation and incident response. We modified two modules from OSSIM, the normalization and correlation to integrate all alerts from different IDSs analyzers in the cloud i.e., OSSEC and Snort alerts by applying the IDMEF protocol. We detail the two modules in Chapter 8.

Chapter 2

Intrusion Detection and Related Works

This chapter introduces the intrusion detection systems definition, architecture and techniques. Then, it reviews previous works on intrusion detection systems and masquerade detection. Finally, it discusses the current intrusion detection datasets and their deficiencies for cloud systems.

2.1 Intrusion Detection Systems

Intrusion detection [63] is the process of monitoring and analyzing the events occurring in an ICT system to detect signs of intrusions. Intrusions are defined as attempts to compromise the confidentiality, integrity, availability of a system component, or to bypass a security mechanism. They may be generated by attackers accessing the systems from the Internet or by authorized users who attempt to gain additional privileges or misuse their privileges.

Intrusion Detection Systems (IDSs) are software or hardware components that automate the monitoring and the analysis. There are several compelling reasons to adopt IDSs [63]:

(1) To prevent illegal behaviors by increasing the perceived risk of discovery and punishment.

(2) To detect attacks and other security violations not prevented by other security measures.

(3) To detect and deal with the preambles to attacks

(4) To document existing threat to an organization.

(5) To act as quality control for security design and administration.

(6) To provide useful information about intrusions that do take place, allowing improved diagnosis, recovery, and correction of root causes.

2.1.1 Intrusion Detection System Architecture

At a very macroscopic level, an IDS can be described [64] as a detector that processes three kind of information from the system to be protected (Figure 2.1):

(1) Long-term information depending upon the technique to detect intrusions, e.g. a knowledge base of attacks.
(2) Configuration information about the current system state.
(3) Audit information describing the system events e.g., C2 audit trail, the syslog in the UNIX world, the event log in Windows NT.

The detector removes unnecessary information from the audit trail and presents a synthetic view of security-related user actions. A decision is then made according to the probability that these actions are symptoms of an intrusion.

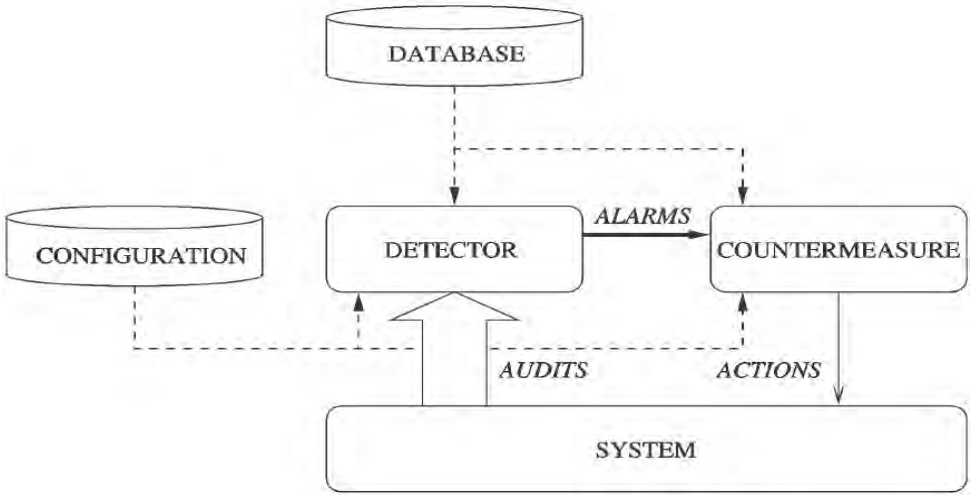


Figure 2.1: Simple Intrusion Detection System

The following three measures of the efficiency of an IDS have been highlighted in [68]

(1) **Accuracy.** An inaccurate IDS flags as anomalous or intrusive a legitimate action in the environment.

Any IDS has four possible outcomes defined by the IDS reaction matrix, see Table 2.1. The outcomes are known as. True negatives (TN) as well as true positives (TP) correspond to a correct IDS operation when events are successfully labeled as normal and attack, respectively. False positives (FP) refer to normal events predicted as attacks, while false negatives (FN) are attacks incorrectly predicted as normal events. [65, 66]

Table.2.1: Possible status for an IDS reaction [65, 66]

		Predicted	
		Normal	Attack
Actual	Normal	True Negative(TN)	False Negative(FN)
	Attack	False Positive(FP)	True Positive(TP)

The following equations compute the rate of these reactions to quantify the IDS performance [67]:

$$\text{True Negative Rate (TNR)} = \frac{\mathbf{TN}}{\mathbf{TN+FP}} = \frac{\mathbf{no.\ true\ alerts}}{\mathbf{no.alerts}} \quad (2.1)$$

$$\text{True Positive Rate (TPR)} = \frac{\mathbf{TP}}{\mathbf{TP+FN}} = \frac{\text{no. detected attacks}}{\text{no. observables attack}} \quad (2.2)$$

$$\text{False Positive Rate (FPR)}: \frac{\mathbf{FP}}{\mathbf{TN+FP}} = 1 - \frac{\mathbf{TN}}{\mathbf{TN+FP}} \quad (2.3)$$

$$\text{False Negative Rate (FNR)}: \frac{\mathbf{FN}}{\mathbf{TP+FN}} \quad (2.4)$$

$$\text{Accuracy} = \frac{\mathbf{TN+TP}}{\mathbf{TN+TP+FN+FP}} \quad (2.5)$$

$$\text{Precision} = \frac{\mathbf{TP}}{\mathbf{TP+FP}} \quad (2.6)$$

(2) **Performance.** The performance of an IDS is the rate at which it processes audit events. A poor performance prevents real-time detection.

(3) **Completeness.** An incomplete IDS fails to detect an attack. The evaluation of this measure is rather complex due to the lack of global knowledge of attacks.

Two further properties are defined in [64]:

(1) **Fault tolerance or resilience.** An IDS should resist to attacks, particularly to denial of service. This is important because most IDSs run on top of commercial OSs or hardware which are vulnerable to attacks.

(2) **Timeliness.** The performance of an IDS should enable an early reaction to prevent the attacker from subverting the audit source or the IDS itself. The corresponding performance measure encompasses both the native performance of the IDS and the time to propagate the information and react to it.

2.1.2 Intrusion Detection Methods and Techniques

There are two distinct approaches to detect an intrusion:

A. The search for evidence of attacks based on knowledge accumulated from known attacks.
B. The search for deviations from a model of normal behavior based upon observations of a system during a known normal state.

The first approach is referred to as misuse detection or detection by appearance. The second trend is referred to as anomaly detection or detection by behavior. We denote the first approach as knowledge-based intrusion detection because it describes more precisely the adopted technique. The second approach is characterized as behavior-based intrusion detection. We highlight both approaches in the next sections with their relative techniques as in [64]

A. Knowledge-based intrusion detection

Knowledge-based intrusion detection techniques exploit the knowledge available about specific attacks and system vulnerabilities. The IDS stores and manages information about these vulnerabilities and looks for attempts to exploit them. When it detects an attempt, it triggers an alarm. In other words, any action that is not explicitly recognized as an attack is accepted. Therefore, knowledge-based intrusion detection systems may achieve a good accuracy. However, they achieve completeness only if their knowledge of attacks is updated regularly.

Knowledge-based approaches have the potential for very low false alarm rates, and the IDS may implement a detailed contextual analysis that simplifies preventive or corrective actions.

Drawbacks include the difficulty of gathering information on attacks and keeping it abreast with new vulnerabilities. Knowledge-based method uses different techniques namely: expert systems, signature analysis, and state transition analysis.

- **Expert systems.** These IDSs contain set of rules that describe attacks [69]. Audit events are translated into facts carrying their semantic meaning in the expert system and the inference engine draws

conclusions using these rules and facts. This method increases the abstraction level of the audit data by attaching a semantic meaning to it. Rule-based languages [70] are a natural tool for modeling the knowledge about attacks. This approach allows a systematic browsing of the audit trail in search for evidence of attempts to exploit known vulnerabilities.

- **Signature analysis.** Signature analysis follows the same knowledge-acquisition approach as expert systems. However, these IDSs exploit in a different way the knowledge because the method decreases the semantic level of the attack description by transforming it into information that can be found in the audit trail. For example, attack scenarios might be translated into the sequences of audit events they generate, or into patterns of data in the system audit trail. The implementation of this technique can be very efficient and it is therefore adopted by commercial IDSs. As in any knowledge-based approaches the main drawback is the update to keep up with the stream of new vulnerabilities and attacks.
- **State-transition analysis.** This technique [71] was implemented first in UNIX and later in other environments. It is conceptually identical to model-based reasoning: it describes attacks as a set of goals and transitions, but represents them as state-transition diagrams.

B. Behavior-based intrusion detection

Behavior-based intrusion detection techniques assume that an intrusion can be detected by observing a deviation from the normal or expected behavior of the system or the users. The model of normal or valid behavior is extracted from information collected by various means. Then, the IDS compares this model with the current system activity and raises an alarm when it observes a deviation In other words, anything that does not correspond to a previously learned behavior is considered intrusive. Therefore, the intrusion-detection system might be complete, but its accuracy poses complex issues.

Behavior based approaches can detect attempts to exploit new and unforeseen vulnerabilities and they can even contribute to the (partially) automatic discovery of new attacks. They are less dependent on OS specific

mechanisms and can also help to detect "abuse of privileges" attacks that do not actually exploit any security vulnerability. A high false alarm rate is generally cited as their main drawback because the learning phase may not cover any possible behavior. Also, behavior can change over time, introducing the need for periodic on-line retraining, resulting either in the unavailability of the IDS or in false alarms. If the information system is under attack when the IDS is learning what is acceptable behavior, the behavior profile may contain intrusive behavior, which is then not detected as anomalous. The method uses different techniques namely: statistics, expert systems, neural network, and user intention identification.

- **Statistics.** Statistics is the most widely used tool to build behavior-based IDSs [72]. The user or system behavior is measured by a number of variables sampled over time. Examples include the login and logout time of each session, the resource duration, and the amount of processor-memory-disk resources consumed during the session. The time sampling period ranges from a few minutes to about one month. The original model keeps averages of all these variables and detects whether thresholds are exceeded based on standard deviations.
- **Expert systems.** They are useful for policy-based usage profiles but less efficient than the statistical approach to process large amounts of audit information.
- **Neural networks.** Knowledge-based intrusion detection use neural networks to learn attack traces and seek them in the audit stream. Currently, a neural network cannot propose an explanation of the attack because there is no reliable way to understand what triggered the association. Therefore, IDSs use neural networks to learn the behavior of actors in the system e.g. users, daemons. Experiments that used a neural network to predict user behaviors [73] have shown that the behavior of UNIX root users is extremely predictable because of the very regular activity generated by automatic system actions or daemons. Furthermore, the behavior of most users is also predictable but that of a very small fraction of users is unpredictable.

- **User Intention Identification**. User Intention Identification [74] is a technique [75] that models the normal behavior of users in terms of their high-level tasks. Then, these tasks are refined into actions related to the audit events observed on the system. The analyzer pairs each user with a set of tasks the user can perform. Whenever a user action does not fit the task pattern, an alarm is issued.

2.1.3 Intrusion Detection Message Exchange Format (IDMEF)

The Intrusion Detection Message Exchange Format (IDMEF) [76] is a XML standard format for messages exchanged among IDSs. This model is an object-oriented representation of the alert data that the intrusion detection analyzers transmit to the management systems. It provides a standard, coherent representation of alerts and it describes the relationship between simple and complex alerts.

As shown in Figure 2.2, IDMEF Message is the top-level class and it has two subclasses, Alerts and Heartbeat. A heartbeat message signals the current status of the IDS analyzer to the central manager or the other way around and they are sent with a predefined frequency. The absence of a Heartbeat message denotes a failure of the analyzer or of its network connection. The Alert message is a response 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. The two times are based upon the characteristics of the firing IDS. Hence, two alerts might be considered similar even though their times of creation and of analysis differ.

c) Source and target. The structures of the source and target subclasses are similar; they might be described by an IP address or a host or user name.

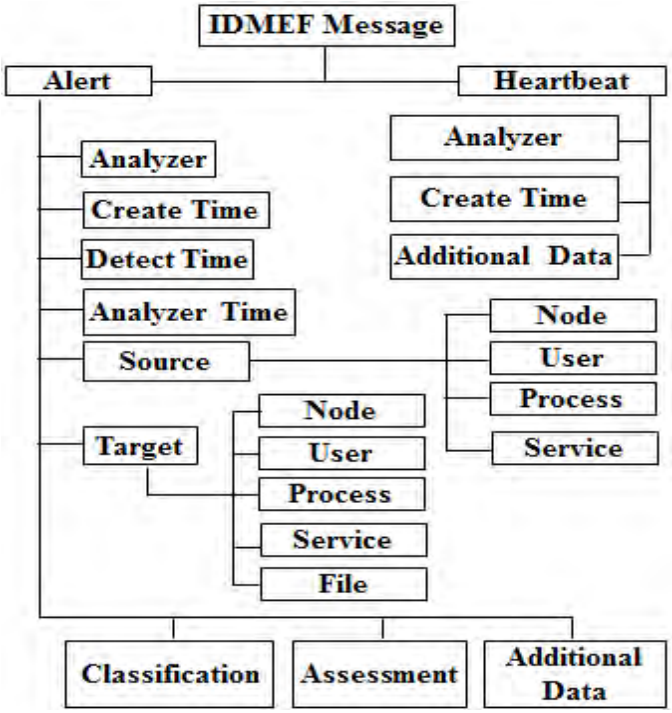


Figure 2.2: IDMEF Data model

2.1.4 Related Work in Intrusion Detection Systems

IDS technology has been proposed as an efficient security measure and is nowadays widely adopted for securing critical IT-Infrastructures. According to the protected objectives, IDSs can be categorized to three main categories namely [77]:

(1) Network-based Intrusion Detection Systems (NIDS).
(2) Host-based Intrusion Detection Systems (HIDS).
(3) Distributed Intrusion Detection Systems (DIDS).

The latter contains both types of sensors (i.e., HIDS and NIDS)

Cloud based IDS is a new trend of researches which extends distributed IDSs. A few papers have proposed some IDSs frameworks for cloud systems. Some of these frameworks target SaaS service model, the others adapt some traditional techniques such as mobile agents. These papers do not discuss an implementation of the proposed frameworks or sometimes cover only one service model. This section describes all the previously mentioned categories of IDSs and reviews previous works on this theme.

(1) Network-based Intrusion Detection Systems

A Network-based IDS (NIDS) detects attacks by capturing and analyzing network packets. By listening on a network segment or switch, a NIDS can monitor the network traffic among the hosts connected to the segment. NIDSs often consist of a set of single-purpose sensors or hosts at various points in a network. These components monitor network traffic, implement a local traffic analysis and reports attacks to a central management console [63].

The best strategy to secure a large-scale network is to partition it into smaller networks using switches. Separate network segment are then protected through security technology such as firewalls and IDSs [78]. Figure 2.3 gives an example of NIDSs deployment.

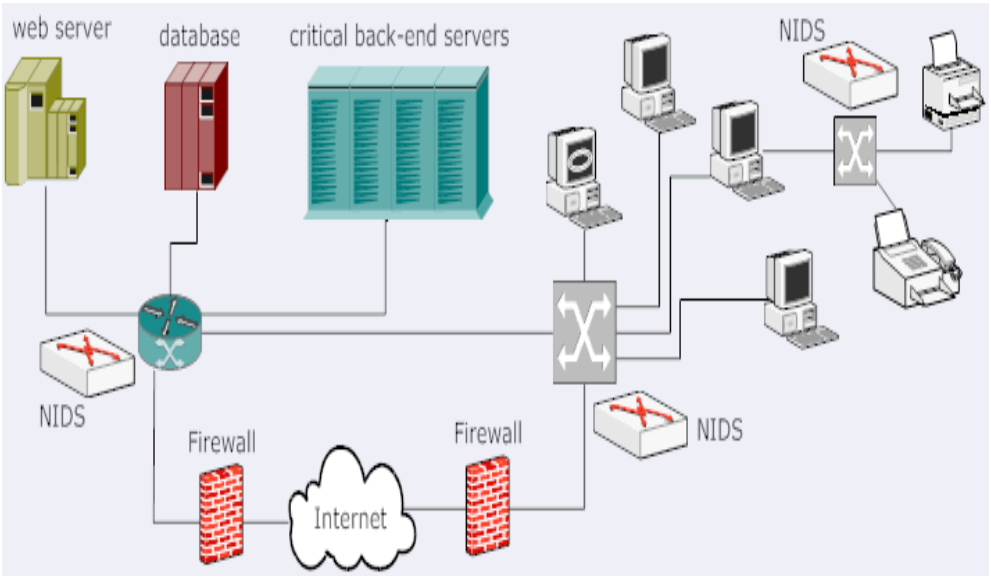


Figure 2.3: An example of network-based intrusion detection system

The advantages of NIDSs are [63]:

(1) A few well-placed NIDSs can monitor a large network.

(2) Their deployment has little impact on an existing network.

(3) They can be made very robust against attack and even made invisible to most attackers.

The disadvantages [63] are:

(1) NIDSs cannot analyst all the packets in a large or busy network. Hence, they may fail to recognize an attack launched during periods of high traffic.

(2) Several advantages of NIDSs do not apply to switch-based networks. While switches subdivide networks into many small segments, sometime they do not provide universal monitoring ports where all the traffic is mirrored. This limits the monitoring range of a NIDS sensor.

(3) NIDSs cannot analyze encrypted information. This problem is increasing as more organizations (and attackers) use virtual private networks.

(4) Most NIDSs can discover that attack is attempted but cannot tell whether or not it was successful. Hence, if a NIDS detects an attack, administrators have to manually investigate whether it was successful.

(5) Some NIDSs become unstable and may crash if network-based attacks involve fragmenting packets.

(2) Host-based Intrusion Detection Systems

A Host-based intrusion detection system (HIDS) is installed on a host to monitor suspicious events occurring within it. In other words, a HIDS resides on network end-points. Unlike NIDSs, HIDSs monitor not only malicious network traffic but also events within the protected host.

An HIDS is rather powerful [78] because it is designed to operate on a specific host such as web or a mail server. Hence, it may be integrated with the software node and be designed to communicate with other network components and OSs.

Furthermore, HIDSs can complement NIDSs because they can analyze packets at the application ends and inspect encrypted traffic [78]. Since an HIDS detects attacks inside its local host, it could not detect attacks from outside its boundaries. Figure 2.4 gives an example of HDS deployment and Table 2.2 summarizes advantages and disadvantages of HIDS and NIDS.

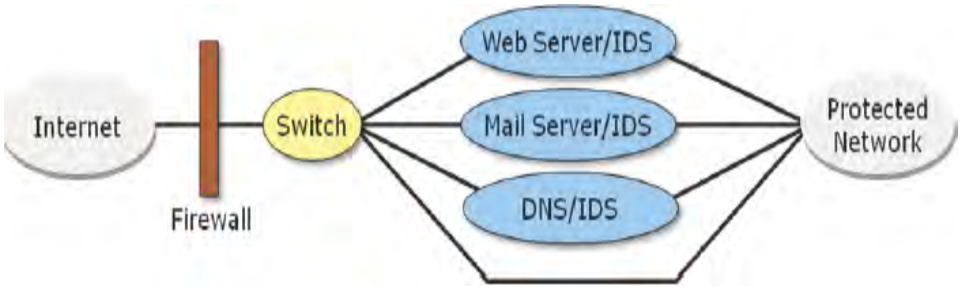


Figure 2.4: An example of Host-based intrusion detection system

Table.2.2: Evaluation of HIDS and NIDS

NIDS	HIDS
Better for detecting attacks from outside	Better for detecting attacks from inside that NIDS cannot analyze
Examines packet headers & entire packet	Does not see packet headers
Host independent	Host dependent
Bandwidth dependent	Bandwidth independent
Slow down the networks that have IDS clients installed	Slow down the hosts that have IDS clients installed
Detects network attacks, as payload is analyzed	Detects local attacks before they hit the network
Not suitable for encrypted and switches network	Well-suited for encrypted and switches network
Does not perform normally detection of complex attacks	Powerful for analyzing a possible attack because of relevant information in database
High false positive rate	Low false positive rate
Examples: Snort [55], Cisco Guard XT [79]	OSSEC[61], Samhain [80], Osiris[81], and eEye Retina [82]

(3) Distributed Intrusion Detection Systems

A distributed IDS (DIDS) consists of multiple IDSs over a large network. The IDSs interact in a hierarchical architecture with either several servers or a unique central server [DIDS-Symantec]. Figure 2.5 shows the tree structure of a hierarchical architecture where circles represent network nodes and arrows denote the information flows between different types of nodes. The leaf nodes represent network-based or host-based collection points. They gather information that is transmitted to internal nodes, which aggregate information from multiple nodes. Further aggregation, abstraction and data reduction occurs at higher level nodes until reaching the root node. This node

is a command and control system that evaluates attack signatures, issues responses and reports to an operator console where an administrator can manually assess status and issue commands. The hierarchical structures make the IDS vulnerable to direct attacks. Several points of failure exist in the IDS that have no redundant communication lines or the capability to dynamically reconfigure relationships if a key component fails. The IDS may also still be vulnerable because current implementations do not apply survivability techniques such as redundancy, mobility, or dynamic recovery [84, 85]. Some known examples of DIDS are EMERALD [86], INBOUNDS [87]. Sometimes, see Figure 2.6, the IDS collector components over a large network communicate with a central server to simplify network monitoring, incident analysis, and instant attack data. The system works as a centralized IDS but with a collection of distributed collector components.

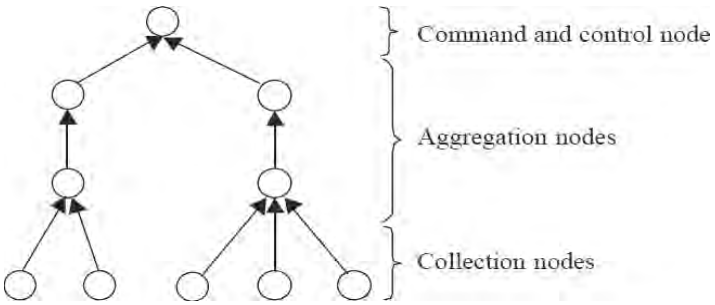


Figure 2.5 Hierarchical DIDS

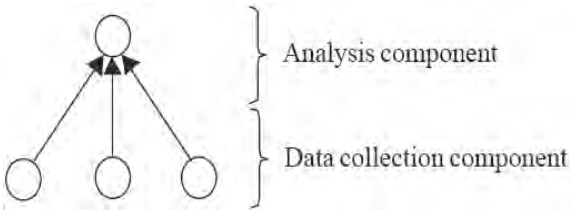


Figure 2.6 Unique central server

The data collection component in Figure 2.6 receives information from the audit logs and the host internal interfaces or from the network packets. Then, it transmits information to a centralized analysis component in another machine (i.e., a server or a dedicated machine) that analyzes it [78]. This architecture is effective for small numbers of monitored nodes. The centralized analysis limits the system's scalability: as more collection components are added, the processing load on the analysis component

increases with the overhead on the machine running this component. Also this architecture represents a single point of failure. Some known examples of this type of DIDS are NIDES [88], ARMD [89], Stalker [90], and UNICORN [91].

DIDS can be categorized as Mobile Agent Intrusion Detection Systems (MAIDS), Grid based Intrusion Detection Systems (GIDS), and Cloud based Intrusion Detection Systems. We will review them below.

(a) Mobile Agent Intrusion Detection Systems (MAIDS)

The DIDS architecture does not scale well for large networks since any new component increases the load on the DIDS director, and the data flowing to the director can consume most of network bandwidth. MAIDS address these scalability problems by using Mobile Agents (MAs) for decentralized data analysis.

A software agent is a software entity which functions continuously and autonomously in a given environment. It can execute activities in a flexible and intelligent manner that responds to changes in the environment, learns from its experience and cooperates with other agents [92]. MAs are a type of software agent with the capability to move from one host to another. For mobile agents to be useful for intrusion detection, a MA platform has to be installed on most, if not all, hosts and network devices. There are different functional and performance requirements [93] to enable the MAs to successfully detect intrusions. Figure 2.7 shows the movement of an agent among several platforms.

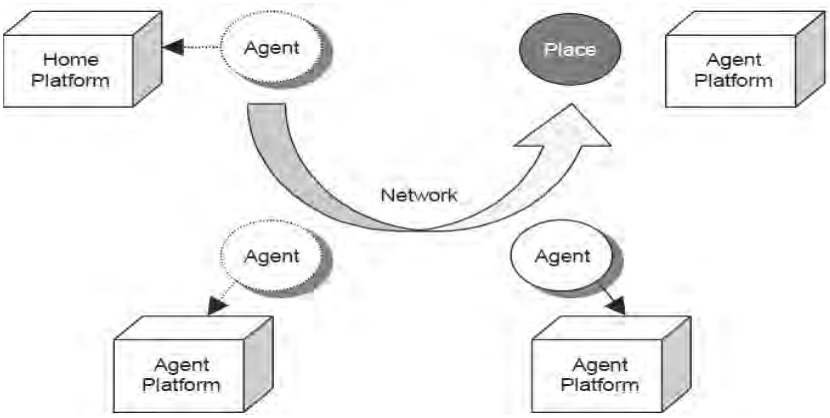


Figure 2.7: An Agent System Model

The platform where an agent originates is referred to as the home platform. Normally, this is the most trusted environment for an agent. One or several hosts may comprise an agent platform that may support multiple locations or meeting places where agents can interact. The main advantages of mobile agent for IDS are reported in [93].

While MAs are a powerful tool, their implementation has been hindered by security considerations that are critical for IDSs, with the result that most security research in this field has concentrated upon the architecture to provide security for mobile agents [93]. The adoption of MAs for IDS poses several problems [93]:

(1) The security issues related to MAs: there are different security threats for MAs namely, agent-to-agent, agent-to platform, platform-to-agent, and other-to-agent platform. The agent-to-agent category represents the set of attacks where agents attack other agents by exploiting their security weaknesses. The agent-to-platform category represents the attacks the agents launch against a platform. The platform-to-agent category represents the attacks where platforms compromise the security of agents. The other-to-agent platform category represents the set of attacks where external entities, including agent platforms, threaten the security of an agent platform.

(2) The performance issue: MA software will generally hinder rather than help an IDS to rapidly process events and detect attacks. MA runtime environments implemented in slow interpreted languages may slow down MAIDS.

(3) The code size issue: IDS services require a large amount of code. If agents have to implement specific tasks on multiple OSs then the code base may become extremely large. The code size may limit the MAID functionality because an agent transfer takes a long time and a large amount of computing and network resources.

(4) Lack of a priori knowledge: Large enterprise networks include several distinct hardware platforms, running several OSs, each with different configurations and applications. It is not trivial for MAs to have a priori knowledge about a system and still remain lightweight. Hence, in a large enterprise, the required priori knowledge may prohibit a rapid agent transfer.

(5) Coding and Deployment Difficulties: MAs that are developed in-house or purchased from trusted vendors are likely to undergo the same software engineering methods as their non-mobile counterparts to assure the quality of code. This historically produces code with several faults. The capability of MA, such as moving and cloning, increase the complexity of design and development. Hence, MAIDS will be even more prone to faults than their non-MA counterparts.

A comparison between previous related works for MAIDS is outlined in [94].

(b) Grid based Intrusion Detection System (GIDS)

The heterogeneity of grid systems and their geographical spread over boundaries and organizational structures lead to potential security issues. The underlying network infrastructure of a grid can be the target of an attack. Attacks against any network or host in a grid can also be considered as attacks against the grid, because they affect its security aspects. Grid systems are susceptible to specific attacks because of their new protocols and services. Grid attacks mostly target to [96, 97]:

(1) Processes running in kernel space e.g., OS daemons.

(2) Processes running outside kernel space e.g. grid middleware, grid applications, and any non-grid applications running with either root or user privileges.

(3) Grid protocols stack and network devices.

Some works related to GIDS are outlined in the following. [98] describes a grid-based IDS architecture where agents located at grid nodes collect and transmit host audit data to storage and analysis servers. This centralized solution is not scalable. [99] proposes an efficient and scalable solution for storing and accessing audit data collected from grid nodes, but it does not discuss how to use the data to identify intrusions. [100] proposes a solution that integrates the grid system with an IDS to analyze data from the grid network. However, these approaches cannot detect grid-specific attacks, because they cannot capture high-level data to identify grid users. The Grid Intrusion Detection Architecture (GIDA) [101] solves the scalability problem by distributing the intrusion detection task among several analysis servers.

Both [98] and [101] focus on the detection of anomalies in the interaction of grid users with resource but neither architectures provide protection against the host and network attacks. [102] proposes a Performance based Grid Intrusion Detection System (PGHIDS). This IDS uses the abundant resources of a grid to detect intrusion packets, but it does not detect attacks to the grid itself and it acts as a NIDS, rather than a Grid-based IDS because it only looks for network attacks. GHIDS is an IDS [103] to defend computational grids against misusing of shared resource. It integrates a HIDS in a grid environment to protect against typical OS attacks, but it does not consider middleware vulnerabilities. [104] proposes a high-level Grid-based IDS built on the functionality of lower-level HIDS and NIDS. However, both traditional HIDS and NIDS are not precisely suitable for grid specific-attacks. For example, traditional HIDS identifies an intruder not with grid user ID but with local user ID. Hence, this IDS cannot identify grid intruder precisely and the information without grid user ID is less useful for the behavior analysis of a grid user. Furthermore, some characteristic of grid-specific attacks differ from those of traditional ones. Hence, the adoption of standard HIDS to detect grid attacks will results in high missing rate. The same authors have proposed another framework [105] for both grid and cloud systems. They increase the scalability by balancing among all nodes the load to analyze the intrusions and by removing the centralization deficiency from the IDS in [104].

They also enhanced the coverage of attacks by applying both knowledge-base and behaviour-base techniques, but their solution lacks several features related to the cloud system like virtualization, utilization, and deployment of cloud environments. Since the solution has been applied to a specific grid middleware, the proposed framework is more suitable to grid systems than to clouds. In [106] we proposed a new job analyzer component based on stack inspection methodology to work inside a GIDS that can be applied to different existing grid systems e.g., Condor [107], Globus [108] and our HIMAN system [8, 9]. The job analyzer component considers access permissions of submitted tasks. Its functionality is shown in Figure 2.8:

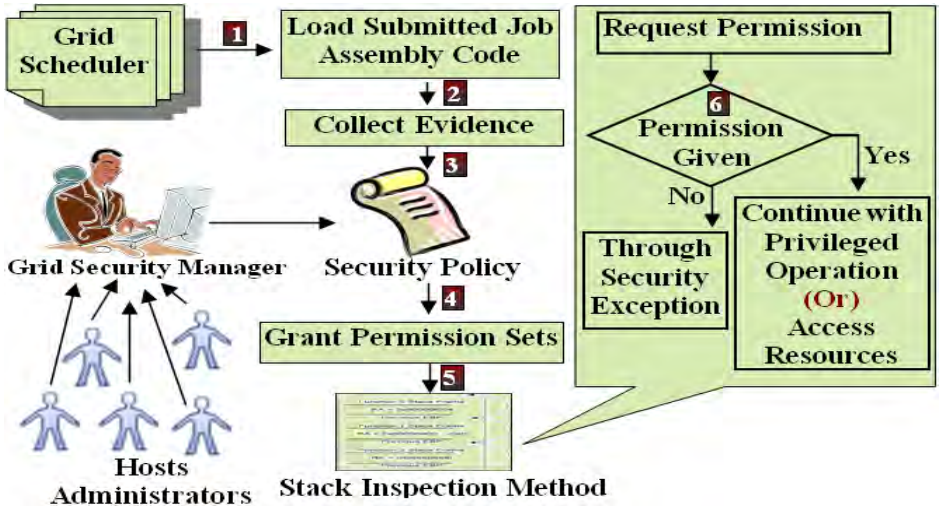


Figure.2.8: A flowchart for GIDS job analyzer component

(1) The scheduler loads the submitted job to the host machine
(2) Collects evidence from the assembly by host machine
(3) Evaluates and tests evidence against the grid security policy.
(4) Uses the output of the previous evaluation step to build the permission sets that enable the requests in next step.
(5) Checks the requests for permission using the stack inspection methodology [109, 106].
(6) If the submitted job and its callers have been granted the requested permission then the operation can proceed otherwise, a security exception is raised.

Table 2.3 summarizes previous proposals for GIDS

Table.2.3: Comparing characteristic of previous related works for GIDS

IDS Reference	Knowledge-based technique	Behaviour based technique	Data Source	Host-based IDS	Network-based IDS	Valid for Grid
GIDS2003	NO	NO	N/A	No	Yes	No
Grid-wide2005	Yes	No	Network	No	Yes	Yes
GIDA2005	No	Yes	Grid Network	Yes	No	Yes
GIDS2005	Yes	No	Network	No	Yes	Yes
GHIDS2006	Yes	No	Host	Yes	No	Yes
GIDS2008	No	Yes	Host, Network	Yes	Yes	Yes

There are several reasons that make it difficult to apply solutions of grid based IDS to Cloud based IDS:

(1) The different service models (SaaS, PaaS, and IaaS) with different types of threats and distinct kinds of users' requirements.

(2) The scalability issue, because most current GIDSs exploits either a hierarchical or a centralized architectures.

(3) Most GIDSs do not integrate the knowledge base and the behaviour base techniques.

(4) They use NIDS that cannot deal with the encrypted data while most data exchanged among cloud nodes is encrypted.

(5) They do not correlate alerts from different nodes to analyze distributed attacks.

(c) Cloud based intrusion detection system

Intrusions in cloud systems are characterized by the potentially higher performance, consequences, and damages of cloud based intrusions. The deficiencies of current IDSs hinder their application to clouds. Here, we highlight the few papers discussing this topic. [94] proposes an IDS based on MAs technology to provide intrusion detection for cloud applications regardless of their locations and that handles attacks for cloud applications from the SaaS point of view. The proposed IDS tries to solve some of the security problems for MAs by isolating the agents inside VMs that provide secure sandboxes for the MAs. Figure.2.9 shows the proposed architecture.

The proposed hybrid model introduces four main components, namely IDS Control Center (IDS CC), Agency, Application Specific Static Agent Detectors, and Specialized Investigative MA. Static Agents (SA) can implement packet filtering and look for intrusion signatures in the packets. SA generate an alert to IDS Control Center whenever they detect suspicious activities. Then, IDS Control Center will send investigative task-specific Mobile Agent to every agency that sent similar alerts (VM 1 and VM 2 in this example).

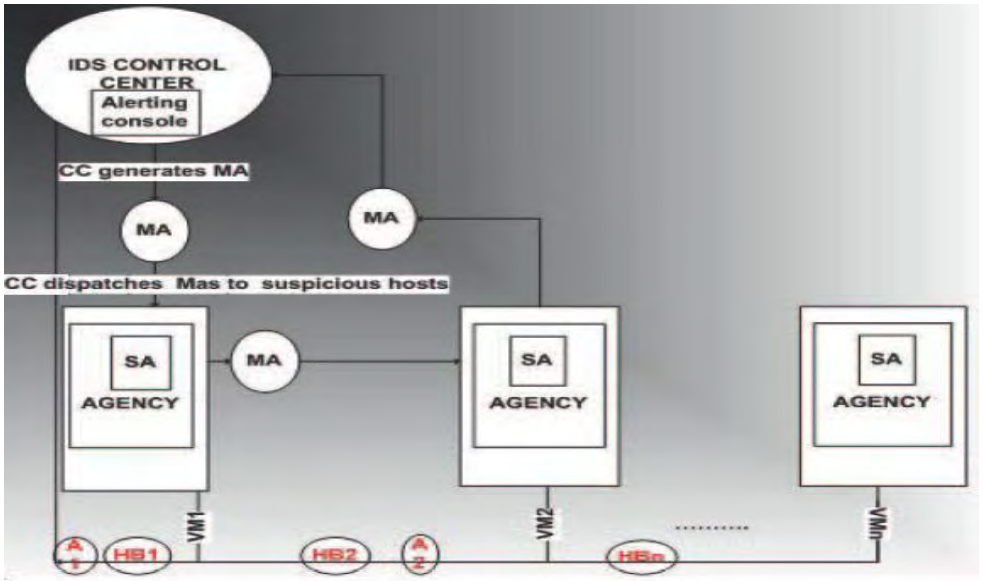


Figure.2.9: The proposed IDS architecture in a subnet

As shown in Figure 2.9, MAs will visit and investigate all those VMs, collect information, correlate it and finally send or carry back the result to IDS Control Center. Consequently, the Alerting Console in IDS Control Center will analyze the incoming information to raise the alarm if it detects an intrusion. Names and identifications of suspected VMs will be black listed and sent to other VMs. This solution is flexible and cost-effective as it tries to further reduce network load by making each MA lighter as it is only responsible for detecting certain types of intrusions.

Nevertheless, there are many deficiencies in this architecture:

(1) The security issues related to MAs mentioned before.

(2) The proposed IDS isolates the MAs from the host environment but it cannot protect them from their generator, i.e. the IDS Control Center environment. Obviously, it is impossible to keep agent private from a malicious runtime system executing the agent [95].

(3) Performance is critical because the MA runtime environments slow down MAIDS. The solution [93] uses MAIDS just for some functions and it implements core IDS by statically located systems: This restricts scalability and opens a single point of failure problem.

(4) The central IDS control component restricts scalability.

(5) It does not correlate the host and network IDSs to handle the encrypted packets.

(6) The proposed IDS does not handle other deficiencies for MA such as lack of a priori knowledge and the Coding and Deployment Difficulties.

(7) It is applied to the cloud client only and it handles intrusions related to the SaaS service model and not the other service models.

[77] proposes a theoretical framework targeting all existing service model. To simultaneously provide multiple benefits from various IDS sensors, they used the Intrusion Detection Message Exchange Format (IDMEF) [76] to enable interoperability among different approaches. They enable the end users to control and configure resources with distinct types of sensors, various configurations of the rule-sets and thresholds to efficiently monitor their virtualized components. Figure 2.10 shows the proposed IDS architecture

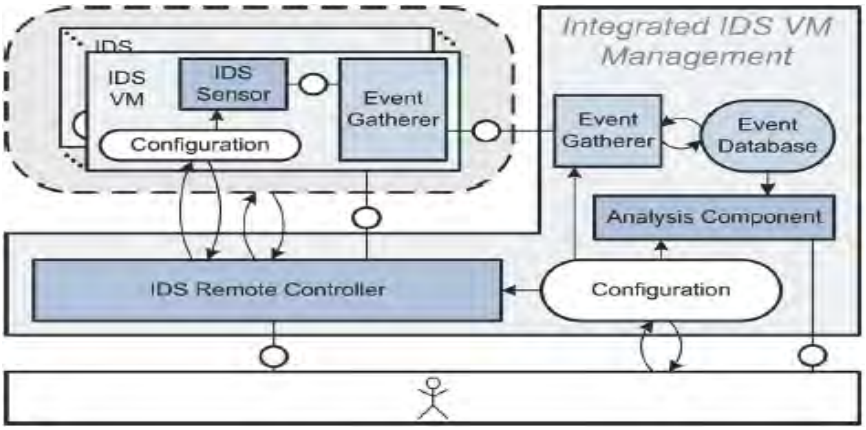


Figure 2.10: The proposed IDS architecture

The architecture includes several IDS Sensor VMs and an IDS Management Unit. An IDS Management Unit consists of four active components: Event Gatherer, Event Database, Analysis Component, and IDS Remote Controller. The Event Database records information about received events. It can be accessed through the Analysis Component. User controls the IDS management through direct interaction and configuration of the core components. The IDS Sensors on the VMs are connected to the Event Gatherer and identify malicious behavior and generates alerts that will be

processed by the Event Gatherer. A sensor can be configured through the IDS Remote Controller. The Event Gatherer collects events from sensors and standardizing the outputs. It also implements the communication between the sensor and the management unit. The Analysis Component represents and analyzes the gathered events [77]. The main deficiencies of this framework are:

(1) It does not correlate alerts from the detectors. The correlation is essential for detecting attacks which leave their trails in distinct cloud locations.

(2) It uses User-Mode-Linux, a type II VMM which only runs on Linux based systems.

(3) The centralized ID unit that manages all other IDSs raises performance and scalability problems.

(4) No component evaluates threats probabilities from the other nodes and compares them against a threshold to support the scheduler decision on the running tasks and their relevant users.

The analysis of previous work confirms that, a proper defense strategy for clouds needs to:

(1) Be distributed to avoid any single point of failure and increase robustness.

(2) Protect the intrusion detection components from the intrusions.

(3) Be scalable to not reduce elasticity.

(4) Have a flexible architecture to be applied to distinct architectures.

(5) Increase attack coverage by integrating both behaviour and knowledge base techniques.

(6) Consider the utilization and deployment in cloud computing by handling different service models and user requirements.

This is discussed in the next chapter with reference to the proposed framework.

2.2 Masquerade Attacks and Detection Techniques

A masquerader is an insider or outside attacker who authenticates as a legal user by stealing the user credentials or by attacking the authentication

service. To understand the masquerader actions, we consider alternative implementations of this attack [110]. Among them, we recall duplication or ex-filtration of user password, the installation of backdoors, eavesdropping and packet sniffing, spoofing and social engineering attacks. Some of these actions may leave some trail in log files that, after the fact, can be linked to some user actions. Here a log analysis by a host-based IDS remains the state-of-the art to detect these actions. Attacks that do not leave an audit trail in the target system may be discovered by analyzing the user behaviors through masquerade detection. Traditional security technologies such as firewalls, IDSs, or authentication protocols are useless because, an attacker can access all the user privileges.

Masquerade detection gathers user information and builds a profile for each user through information such as login time, location, session duration, and commands issued. Then, user logs are compared against the profiles and a mismatching behavior is designated as an attack. The detection of masquerade attacks is quite difficult because even the legitimate daily activities can easily become malicious according to its context. This increases the false positive rate [111]. Masquerade detection is more challenging in cloud systems, since they include a massive amount of resources and users can have different activities in several VMs. Hence, to build a profile, we have to correlate these activities. All the approaches reported in Section 2.2.3 analyze user behaviors according to the sequences of actions in distinct environments i.e., UNIX, Windows, or Network. Possible actions include user command, system calls, a network operation and the name of a window or of a file. To evaluate the detection techniques, we highlight some concepts such as ROC curve and Maxion Townsend Cost [112]. After that, we review current masquerade detection approaches.

2.2.1. The Receiver Operator Characteristic Curve

The Receiver Operator Characteristic (ROC) curve [113] graphically represents a classification system as its discrimination threshold is varied. The ROC is also known as a Relative Operating Characteristic curve, because it compares two operating characteristics, True Positive Rate (TPR), accuracy or Hit ratio and the False Positive Rate (FPR) as the criterion changes. We use this curve to evaluate attack detection accuracy against false positive rate. The plot is obtained by varying the detection threshold and

other detection parameters. Figure 2.11 shows how the ROC curve measures the tradeoff between false positives rates and correct detections.

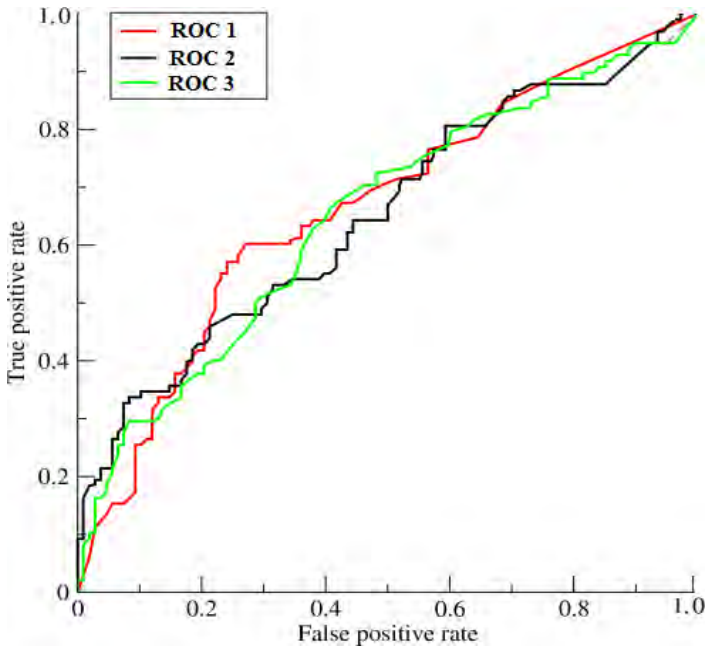


Figure 2.11: Examples for three ROC curves

2.2.2. The Maxion Townsend Cost

Maxion and Townsend [112] created a scoring formulation to rate a masquerade detection algorithm. The formula evaluates the cost of the detection algorithm in terms of a relation between the false alarms and misses, where the miss rate is equals to (100 – Hit Ratio). The overall "goodness" of each of several detection methods can be ranked by this function. While there is a wide consensus in the literature that a false alarm should be more expensive than a miss, it is difficult to determine how much more expensive. According to the experiments that the authors did using the seven masquerade detection approaches detailed in section 2.2.3, a rigorous evaluation requires that the cost of a false alarm to be 6 times that of a miss. Hence, the final cost function is given in Equation 2.7:

$$\text{Percentage Cost} = 6 \times \text{False-Positive-Rate} + \text{Miss Rate} \quad (2.7)$$

2.2.3. A literature Study for Masquerade Detection

None of the current proposals to detect masquerade attacks has achieved the level of accuracy for a practical deployment. This review highlights masquerade detection techniques based on the analysis of user audits. These audits have been collected by several profiling methods from different environments e.g. UNIX, Windows and/or Network environments. We will see later in Chapter 3, how our dataset, CIDD, is collected using different profiling methods so that its data can be used with different detection techniques.

1) Masquerade Detection in UNIX Environments

In UNIX environment, the sources of audit data to build signature patterns are user commands, programs, and system calls. In this review, we highlight and compare several masquerade detection approaches based on UNIX commands in the user dataset called "SEA" described in Section 2.3.1. The considered approaches are: Uniqueness, Naïve Bayes One-step Markov, Hybrid Multi-Step Markov, compression, Incremental Probabilistic Action Modeling (IPAM), sequence-match, Support vector machine (SVM), Recursive Data Mining with SVM, Naïve Bayes classifier, Episode based Naïve Bayes, Naïve Bayes and Weighted Radial Basis Function, Adaptive Naïve Bayes and sequence alignment algorithms.

- **Uniqueness:** This approach [114] assumes that commands not previously seen in the training data may indicate a masquerade. Moreover, the fewer users that are known to use that command, the more indicative that command is of a masquerade. Uniqueness is a relatively poor performer in terms of detection, but it is the only method able to approach the target false alarm rate of 1%.
- **Naïve Bayes One-step Markov:** This approach [115] builds transition matrices from one command to the next for each user's training and testing data. It raises an alarm when there is a considerable difference between the training data transition matrix and the testing data one. It achieves a good performance in terms of correct detections, but failed to get close to the desired false alarm rate.
- **Hybrid Multi-Step Markov:** This method [116] is based on Markov chains. If the test data contain too many commands that did not appear in the training data, a Markov model may be useless and a simple

independence model with probabilities estimated from a contingency table of users versus commands may be more appropriate. [114] toggled between a Markov model and the simple independence one. This method has one of the best performances.

- **Compression:** the main underlying idea [114] is that new data from a user compresses at about the same ratio as old one from the same user. Instead, data from a masquerading user will compress at a different ratio. This approach is the worst performer.
- **The Incremental Probabilistic Action Modeling (IPAM):**It predicts the sequence of user commands according to [117] the one-step command transition probabilities estimated from the training data. Too many wrong predictions signal a masquerade. IPAM's performance ranks with the lowest ones.
- **Sequence-match:** It computes a similarity match between the user profiles and the corresponding sequence of commands. Any score lower than a threshold indicates a masquerader [118]. Its performance on the SEA dataset is not very high.
- **Support vector machine (SVM):** Support vector machine refers to a collection of machine learning algorithms designed for binary classification. SVM classifies data by determining a set of support vectors, the training inputs that outline a hyper plane in feature space [119]. SVM has shown a good performance, it is relatively easy to use and is relatively insensitive to the number of data points and can potentially learn a large set of patterns. However, it has a high false alarm rate and a low detection rate. Furthermore, it has to update user behavior model when a user profile changes.
- **Recursive Data Mining with SVM:** Szymanski et al [120] proposed a recursive mining approach that finds the frequent patterns in the sequence of user commands, encodes them with unique symbols and rewrites the sequence using the new coding. This approach uses a one-class SVM classifier for masquerade detection but it has to mix user data that may be complex in real-world.
- **Naïve Bayes classifier:** Maxion and Townsend [112] applied a Naïve Bayes classifier widely used in text classification tasks and that classifies sequences of user-command data into either legitimate or masquerader.

The method has not yet achieved the level of accuracy required for practical deployment.

- **Episode based Naïve Bayes:** Dash et al [121] introduced an episode based Naïve Bayes technique that extracts meaningful episodes from a long sequence of commands. The Naïve Bayes algorithm identifies these episodes either as masquerade or normal according to the number of commands in masquerade blocks. The proposed technique significantly improves the hit ratio but it still has high false positive rates and it does not update the user profile.
- **Naïve Bayes and Weighted Radial Basis Function (NB-WRBF):** Alok et al [122] integrates a Naïve Bayes approach with one based on a weighted radial basis function, WRBF, similarity. The Naïve Bayes algorithm includes information related to the probabilities of commands entered by one user over the other users. Instead, the WRBF similarity takes into account the similarity measure based on the frequency of commands, f, and the weight associated with the frequency vectors. Here, f is a similarity score between an input frequency vector and a frequency vector from the training dataset. The experiments confirm that NB-WRBF significantly improves the hit ratio but, as the previous approach, it suffers from the high false positive rates. Furthermore, it computes both the Naïve Bayes and the WRBF and integrates their results and this increases the overall overhead. Lastly, it does not update the user profile and neglects the low level representation of user commands.
- **Adaptive Naïve Bayes:** Dash et al [123] introduced an adaptive Naïve Bayes approach based on the premise that both the commands of a legitimate user and those of an attacker may differ from the trained signature but the deviation of the legitimate user is momentary, whereas the attacker one persists longer.
- **Sequence alignment:** The ability of sequence alignment algorithms to find areas of similarity can be used to differentiate legitimate usage from masquerade attacks. To do so, a signature of the normal behavior for a given user should be aligned with audit data from monitored sessions to find areas of similarity. Areas that do not align properly can be assumed to be anomalous, and the presence of several anomalous areas is a strong indicator of masquerade attacks [124]. Among possible algorithms such as global, local and semi-global alignments the most efficient is semi-

global alignment. The proposed approaches has several shortcomings and to address them. [125] modifies the Smith-Waterman alignment algorithm [133] to a semi-global alignment algorithm (SGA), along with a new scoring systems and signature updating scheme. SGA offers several advantages such as:

1) Better accurate and efficiency than current approaches. It achieves a low false positive rates and high hit ratio.
2) It can work with different kinds of audit data. This simplifies its adoption in heterogeneous environments such as grids and clouds.
3) The detection performance reduces the survival of the masquerader inside the system.
4) It can tolerate the few deviations in the legitimate user behaviors.

Figure 2.12 compares all the previous mentioned techniques against the SGA algorithm detailed in Section 2.2.4 in terms of the ROC curves based on SEA dataset. The SGA algorithm with its update scheme achieves a higher hit ratio with a corresponding lower false positive rate.

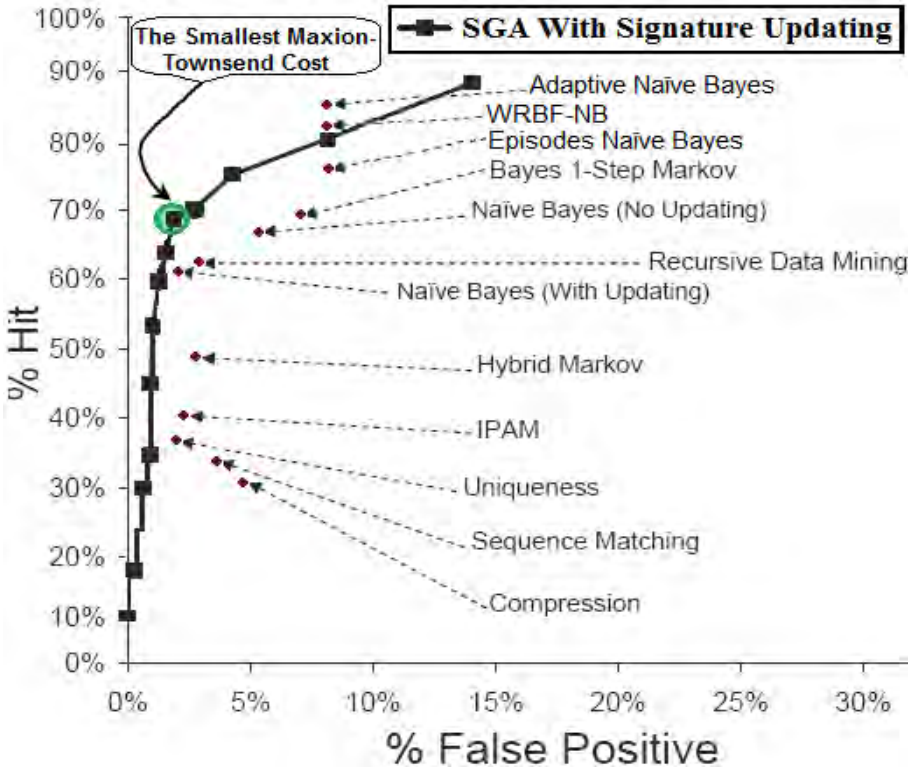


Figure 2.12: ROC curves for some detection techniques [124].

2) Masquerade detection in Windows Environments

In general, Windows security log files contain records of login/logout activity and/or other security-related events as specified by the system audit policy. As mentioned in Section 1.5, there are three log sources in Windows system namely: system, application and security. System and application log are used, respectively, by the operating system and by the applications. Only the Local Security Authority Subsystem Service (lsass.exe) can directly write to the Security log. Several categories of the events that can be logged [33]. Less research work has considered the Windows environments than the UNIX one.

[126] introduced a new framework to create a unique feature set for user behavior on GUI based systems. They collected real user behavior data from live systems and extracted parameters to construct feature vectors. These vectors contain information such as mouse speed, distance, angles and amount of clicks during a user session. They modeled their technique of user identification and masquerade detection as a binary classification problem and used a Support Vector Machine (SVM) to learn and classify the feature vectors.

[127] considers the interaction of the current user with the graphical user interface. Rather than mouse movements or keystroke dynamics, it profiles how the user manipulates windows, icons, menus, and pointers. The method shows potential for use in real-time systems, because it requires less data than other GUI interaction-based masquerade detection techniques while using a much simpler classification engine. Another user profiling method monitors system calls by analyzing the audit logs and program execution traces [128, 129].

3) Masquerade detection in Network Environments

The previous host profiling methods handle only user audits inside a host machine. Other approaches detect masquerade attacks through the user network behavior.

[130] uses basic network statistics. It does not consider host audits at all, because sometimes this data is not accessible or legal/ethical restrictions apply. The approach tags network events in the server log with the associated user and build user network profiles through anonymized summary data.

This limits the privacy impact and avoids the data accessibility issues of host-based approaches.

[131] adopts the well-known Interval Algebra network [132]. The underlying idea is that the signature captures detectable patterns in a user command sequence. A user is modeled as a binary constraint network where each node represents an episode of commands. A binary relationship between a pair of episodes is encoded as the disjunction of the Allens interval relations [132]. Any new subsequence of commands should be consistent with at least one user network.

2.2.4. Masquerade detection using SGA and Enhanced-SGA

The SGA [124] is more accurate and efficient than current approaches. It has low false positive and missing alarm rates and high hit ratio. It can be adopted in heterogeneous environment with distinct operating system because it can be applied to distinct audit data such as command line entries, mouse movements, system calls, registry events, file and folder names, sequence of opened windows titles and network access audit data. SGA aligns large sequence areas as in global alignments, while preserving the nature of local alignments. It can ignore both prefixes and suffixes and it only aligns the conserved area with the maximal similarity. Figure 2 shows an application of SGA and the influential parameters of an alignment namely: match score, mismatch score, test_gap penalty, signature_gap penalty, and detection threshold.

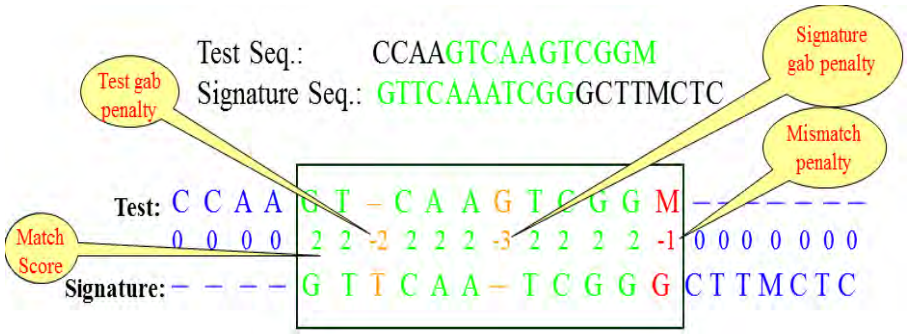


Figure 2.13: An alignment example using SGA algorithm

To discover the optimal alignment, SGA exploits dynamic programming. To this purpose, it initializes an m+1 by n+1 score matrix, M, and then determines the value of each position of M by one of three transitions, see Figure 2.14:

1. Diagonal transition: it aligns the i−1 symbol in the signature sequence with the j−1 symbol in the test sequence. The alignment score depends upon the lexical match of the symbols being aligned and it is added to *M*(i−1, j−1).
2. Vertical transition: A gap is inserted into the signature sequence and it is aligned with the j−1 symbol in the test sequence. The gap penalty is added to *M*(i, j−1).
3. Horizontal transition: A gap is inserted into the test sequence and aligned with the i−1 symbol in the signature sequence. The gap penalty is added to *M*(i−1, j).

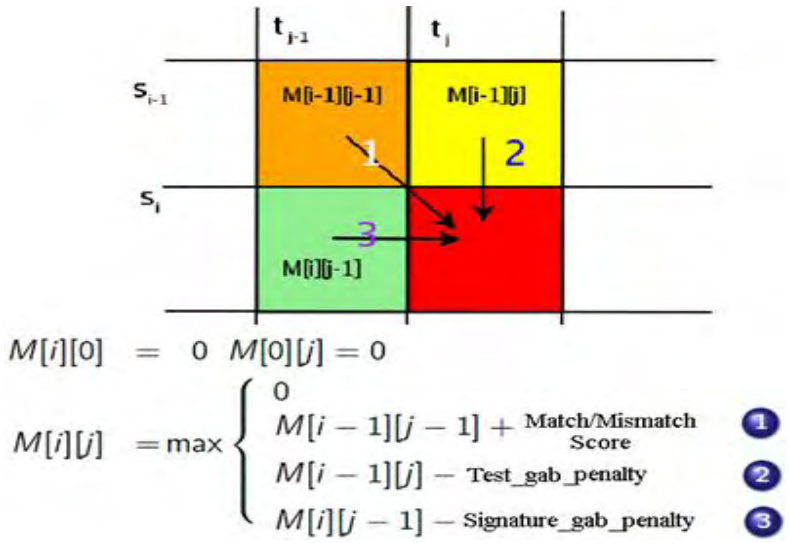


Figure 2.14: The three transitions to fill each cell in the transition-matrix

The SGA scoring system determines the gap penalties in transitions 2 and 3. The actual alignment depends upon the maximum value of the three transitions and its value is assigned to *M*(i, j). *M*(i, j) is the score of the optimal alignment of all symbols up to location i−1 in the signature sequence and j−1 in the test one. Hence, *M*(m, n) gives the score of the optimal alignment of the two sequences returned by the scoring system. We can rebuild this alignment by tracing back the transitions that have produced the score. The final score at *M*(m, n) measures the similarity of the two sequences according to the scoring system used and it is an indicator of masquerade attacks. The SGA algorithm is shown below [125]:

Algorithm 2.1: The SGA algorithm

```
Align (test_subseq of length n, sig_subseq of length m, match_score, mismatch_score,
      sig_gap_penalty, tes_gap_penalty)
01: Begin
02: for i=0 to m step 1 do
03:   for j=0 to n step 1 do
04:     if (i=0 or j=0) then
05:          M(i, j)=0
06:     else
07:          if ( i=m or j=n ) then
08:            top = M(i, j-1)
09:            left = M(i-1, j)
10:          else
11:           top = max (0, M(i, j-1) – sig_gap_penalty)
12:           left = max (0, M(i-1, j) – test_gap_penalty)
13:          end if
14:          if (sig_subseq(i-1) = test_subseq(j-1) ) then
15:            diagonal = M(i-1, j-1) + match_score
16:          else
17:            diagonal = M(i-1, j-1) + mismatch_score
18:          end if
19:          M(i, j)= max(top, left, diagonal)
20:     end if
21:   end for
22: end for
23: return (M(m, n))
24: End
```

The Enhanced-SGA:

Coull et al [125] modified the SGA algorithm to handle the problems of the traditional Smith-Waterman alignment algorithm from two perspectives. The first one considers that the usage patterns of legal users may change due to changes in their role or to new software. A static user signature is therefore prone to label as attacks some variations of legal users. To avoid these false positives, the signature is updated as new behaviour is encountered by exploiting the ability of SGA of discovering areas of similarity. Furthermore, as outlined in Section 5.2.3, they defined two scoring systems, the command grouping and binary scoring systems, to set the alignment scores and the gap insertion penalties.

The problem with Enhanced-SGA algorithm lies in determining the best scoring system. Till now, the penalties for gap insertion to the signature and test sequences (-3 and -2 respectively) are fixed and equal for all the users. Since distinct users behave in a different way, this reduces the efficiency of detection, because the alignment cannot tolerate slight changes in the user behaviour over time. Distinct scoring parameters improve the Enhanced-SGA algorithm and strongly reduce the number of false negatives and false positives. Instead of forcing the same scoring parameters for all users, Data Driven Semi-Global Alignment approach (DDSGA) [DDSGA], see Chapter 5, computes the best scoring parameters for each user separately based on user data and improves the computational performance and the security efficiency of the Enhanced-SGA algorithm.

The signature update scheme is applied with the binary scoring, their most efficient system. This scheme augments both the current signature sequence with information on the new behaviours and the user lexicon with the new commands the user invokes. The scheme also introduces a threshold for each user profile to ensure that both the signature sequence and user lexicon remain free of tainted commands from masquerade attacks. The threshold is used in both detection and update processes, and it is built through a snapshot of the user signatures.

The other perspective considers that the Smith-Waterman algorithm is computationally expensive and impractical to detect masquerade attacks on multi-user systems. By selectively aligning only the portions of the user signature with the highest success probability, the Heuristic Aligning approach [125] can significantly reduce the computational overhead with almost no loss of accuracy in detection. These modifications have been tested on the SEA dataset to simplify the comparison with other approaches.

2.3 Intrusion Detection Dataset

A dataset is a profile of training and testing signature patterns to train and evaluate a behaviour based IDS. In the following, we highlight the current masquerade datasets and the deficiencies arising when adopting them to evaluate cloud IDSs.

2.3.1 Existing Masquerade Datasets and Their Deficiencies

We briefly describe [134, 135] the four datasets that are currently used to evaluate masquerade detection techniques namely, SEA [136] Greenberg [137], Purdue [138], and RUU [139].

1) SEA dataset

Most papers about masquerader detection use the SEA dataset [136] with its associated configurations, SEA-I and 1v49. This dataset consists of commands collected from UNIX acct audit data. Only the username and the command were taken among all the fields of audit data. The data describe 50 users each issuing 15000 commands. The first 5000 commands are considered genuine. The remaining 10000 commands of each user are partitioned into 100 blocks of 100 commands each. These blocks are seeded with masquerade users, i.e. with data of further users. A block is a masquerader with a probability of 1%. If a block is a masquerader, then there is an 80% probability that the following one is a masquerader too. As a result, about 5% of the test data contain masquerades. One of most critical defects of this dataset is that commands have neither arguments nor any parameters. Due to the way acct collects audit data, it is impossible to tell commands typed by human users from those run from shell scripts. This can lead to pair some users with a very regular pattern of few commands.

1v49 Configuration: [112] propose an alternative configuration to the SEA dataset to address some methodological shortcomings in the original configuration. As an example, different masqueraders were injected into different users and some users did not even get any masquerader. This increases the complexity of evaluation and error analysis. In 1v49 configuration, for each user, the first 5000 commands of the other 49 users are used as masquerader data for testing purposes. Despite its methodological advantages, this configuration has not been widely used as it does not simulate masqueraders that are expected in real world.

SEA-I: It is a variation on the original SEA dataset. It was proposed in [140], where the masquerader blocks are replaced by synthetic blocks created according to the command frequency of each user. This is an attempt to model the behavior of an intruder who tries to act like the legitimate user. As a result, more complex techniques are required to detect masqueraders.

2) Greenberg dataset

This dataset [137] contains data from 168 UNIX users using csh (C shell) as command line shell. Users are classified into four groups: novice programmers, experienced programmers, computer scientists and non-programmers. Collected data is stored in plain text files that record : session start time, session end time, the command line entered by the user, the current working directory, the alias expansion of the previous command, an indication whether the line entered has a history expansion or not, and any error detected in the command line. This dataset was first used for masquerader detection purposes in [141]. In contrast with SEA, its main advantage is the availability of additional information for each command which may help to improve detection effectiveness.

3) Purdue University dataset

Purdue, or PU dataset [138] consists of the UNIX shell command histories of 4 users of the Purdue Millennium Lab, collected in a four month period. The number of collected commands per user goes from 7769 to 22530, with an average of about 16500 commands. Command names, arguments and options are preserved but filenames are omitted. This is due to the intuition that the behavior of a user is more significant than content for profiling. The very low number of works that use this dataset is probably due to the low number of users.

4) RUU dataset:

This dataset was collected by Columbia IDS group [139] from Windows environments. To this purpose, they built a Windows host sensor to audit process registry behavior and user window touches. Three types of records were created by the audit program: registry actions, process execution, and window touches. The registry actions that are recorded are open, close and update of specific registry keys by running programs. The records specifying user window touches include the actions of clicking on a window, of switching between two or more windows, and of updating the title of a window. The group has built and published only a Windows dataset even if they have also built a Linux sensor to collect information about the name of the process, path, command line parameters, and system level calls.

The dataset was collected from 34 normal volunteer users and 14 masquerade users who were paid to conduct a red team exercise. On average,

each normal user generated about 1 million records over about 4.5 days of computer use of Windows systems. They model how normal users typically search their computer file system and these models can be used to detect unusual searches that may be a warning of an illegal use of the machine. The masquerader data contains records from about 15 minutes of computer use by each masquerader. The red team users were asked to perform a specific task to find any data that could be used for financial gain on a target file system they had no prior access to.

2.3.2 Deficiencies of Using Current Datasets for Cloud Systems

The datasets previously described suffer partially or fully from several deficiencies which prevent their adoption to evaluate cloud IDSs. Their most significant weakness is the lack of real masquerade and attack data. No command sequences were issued by attackers, only the RUU dataset includes real masquerades but in a predefined scenario where masquerader users were asked to find any data useful for financial gain. Also the SEA dataset simulates masquerade attacks by randomly inserting excerpts of command sequence from one user into the command sequences issued by another user. Some other problems of the datasets are:

(1) They neglect the heterogeneity of clouds systems where user audits may be distributed among different VMs running distinct OSs. Furthermore, cloud users normally use a larger set of applications than those considered by the datasets. The existing masquerade datasets are based on host-based user profiling parameters, and lack important network parameters.

(2) The absence of command arguments and/or other useful audit details such as the time when the user commands were issued and the duration of each user's session.

(3) Their size is very small.

(4) They lack signature details. An efficient cloud dataset should include both behavior based and knowledge based audit data for better training and coverage for attacks in all cloud service models.

Chapter 3

CIDS and CIDS-VERT Frameworks and Their Correlation Models

This chapter introduces two frameworks for a Cloud based Intrusion Detection System, CIDS [142], and CIDS-VERT [143], its specialization version. The two frameworks deal with attacks like: masquerade, DDoS, host-based, and network-based attacks. This chapter details the architecture, testbed of both CIDS and CIDS-VERT frameworks. Furthermore, it describes some essential features of these frameworks to support the selection of the proper one for the cloud system of interest. Finally, it details the three correlation models, Audit Exchange, Independent, and Centralized-Backup.

3.1 CIDS Framework

CIDS is a framework for intrusion detection that provides a defense strategy that deals with attacks against the most widely used cloud services: SaaS, PaaS and IaaS. It is an active IDS that stops the malicious action and raises an alarm.

CIDS has a P2P architecture without a central coordinator to avoid a single point of failure. The architecture distributes the processing load at several cloud locations and executes the user tasks in a monitored VM to isolate them from the cloud. This helps in protecting CIDS components from threats that can control a task in the VM and that can modify CIDS components. To increase attack coverage, CIDS integrates knowledge techniques and behavior based ones. Furthermore, it collects events and audits from VMs to analyze them in the detector and correlator components. Each node also includes an audit system that monitors messages among nodes and the middleware logging system, and collects events and logs from the VMs.

By sharing both the knowledge and behavior databases in each node among the audit components, CIDS can detect the masqueraders that access from several nodes and both host-based and network-based attacks. Furthermore, to take into account the large volume of data in a cloud that

prevents administrators from observing any action, a further CIDS component parses and summarizes a large number of alerts from a NIDS component in a physical or virtual switch in the cloud virtual network. A report for the administrators collects alert messages from all the IDS detectors in the cloud system. CIDS resides inside the cloud middleware which provides a homogeneous environment for accessing all nodes. The middleware sets the access control policies and supports a service-oriented environment. Since the middleware can be install inside distinct grid and cloud systems, CIDS can be applied to several Grid and cloud systems.

3.1.1 CIDS Architecture

In the proposed architecture, each node runs two IDSs detectors, CIDS and HIDS and it cooperates to intrusion detection by identifying the local events that could represent security violations and by exchanging its audit data with other nodes. Figure 3.1 shows the sharing of information among the following CIDS components:

Cloud node: It is one cloud blade that hosts users VMs and resources homogeneously accessed through the cloud middleware.

Guest task: it is a sequence of actions and commands submitted by a user to an instance of VM.

Logs & audit collector: it acts as a sensor for both CIDS and HIDS detectors and collects logs, audit data, and sequence of user actions and commands.

VM: it encapsulates the system to be monitored. The detection mechanisms are implemented outside the VM, i.e. out of reach of intruders. A single instance of a VM monitors can observe several VMs.

Type II Virtual Machine Monitor (VMM): CIDS uses type II VMM implemented as a process of the OS of the host machine. Some properties of a VMM are useful in system security, among them: Isolation, Inspection, and Interposition as detailed in Section 1.3. VMM stores in the audit system the data collected by the logs and audit collector component and forwards them to both the CIDS and HIDS correlator components.

The audit system: this component implements three main functions. First of all, it monitors message exchanges among nodes and deduces the behavior of the cloud user. Then, it monitors the middleware logging system in the node

itself. CIDS can collect all audit data and middleware events such as user login or logout from the cloud system or tasks submissions. The third function collects and stores events and logs from the VM system. A log entry is created for each node action with the action type, (e.g. error, alert, or warning), the event that generated it, and the message.

CIDS correlator and detector: it correlates sequence of commands or actions, collected from several sources and analyzes them through our new Data Driven Semi-Global Alignment approach (DDSGA) detailed in Chapter 5.

HIDS correlator and detector: it correlates user logs and signatures collected from several sources. Then, it analyses them to detect known trails left by attacks or predefined sequences of user actions that might represent an attack. It is implemented by the OSSEC IDS tool detailed in Section 1.10 that receives user logs and signatures and determines whether a rule in the knowledge based database is being broken. After that, it computes the probability that a user action represents an attack, and it communicates this to the alert system that alerts the other nodes if the probability is sufficiently high.

Behavior-based database: it is a profile history database for the behavior of cloud users. It is important that all nodes share the same behaviour database of the same user because this helps in correlating the normal behaviors of a user to detect a suspected behavior distributed among user VMs in several nodes. Since a behavior deviation in one VM can be normal in another one, correlation reduces the false alarms rate and it is more suitable for the deployment and utilization of the cloud system, as a user task can be executed in several VMs. Access to all databases, including events collected by the VMM from the VMs, can be easily implemented by the middleware that transparently creates a virtual homogeneous environment and synchronizes the nodes. As an example, consider that the audit system can create a log entry such as: "User Roy only logs in for 2 to 3 hours and uses a specific sequence of UNIX commands", only if the nodes know the behavior of the user in all VMs in these nodes.

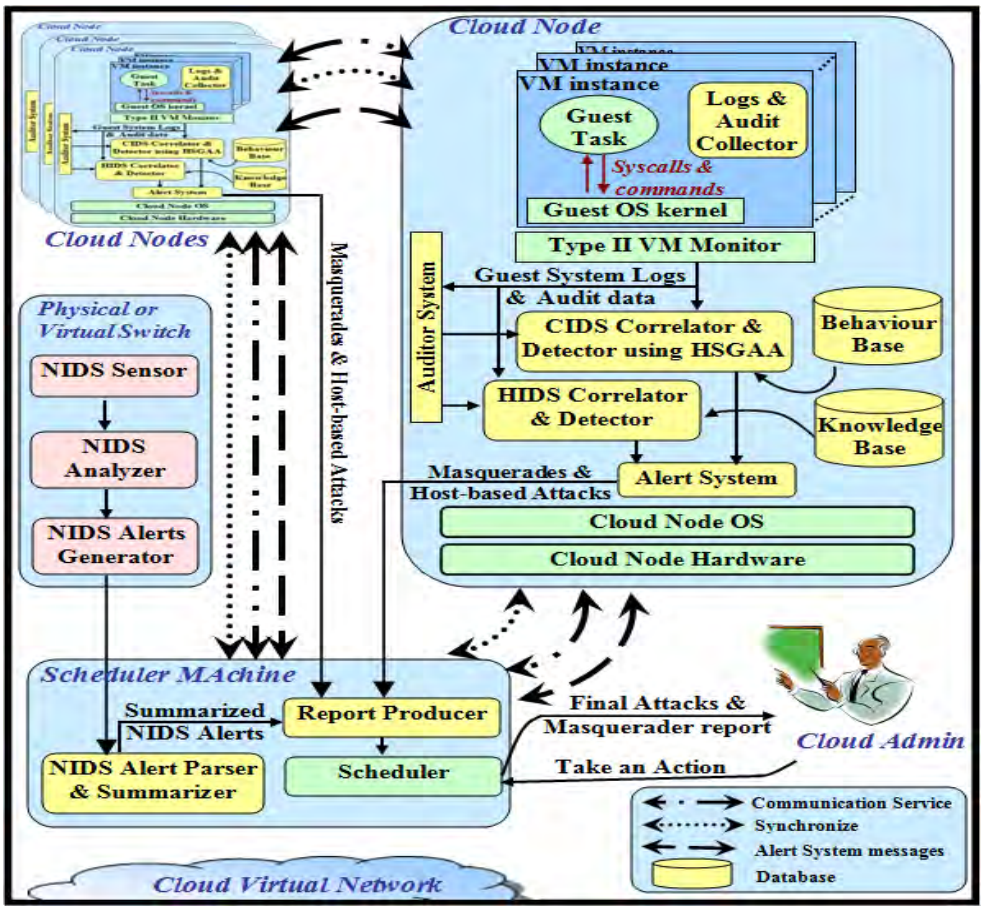
Knowledge-based database: it stores a set of rules and signatures for known attacks. It describes a malicious behavior with a rule to be matched

against those in the database. Like the behavior-based database, all nodes should share the same knowledge base.

Alert System: it uses the middleware's communication mechanisms to alert other nodes if the CIDS or HIDS correlator and detector components signal an attack. It also communicates its alerts to the report producer.

Parser and summarizer: it parses and summarizes the alerts fired by a component in the cloud virtual network. We will briefly explain later the adopted algorithm.

Report producer: it collects alerts from any IDS in the system and sends a report to the cloud scheduler. It helps service providers to discover if their infrastructure is exploited to penetrate other victims.



Yellow components are CIDS components, Green ones are cloud system components, and Pink ones are NIDS components

Figure 3.1: CIDS Architecture

3.1.2 CIDS-Testbed:

The testbed consists of 3 nodes connected by a Gigabit Ethernet. Each node is a quad core clocked at 2.8 GHz with 16 GB RAM and a 80 GB Hard drive. To provide a full heterogeneous testbed, each node hosts 3 VMs with a distinct OS, namely: Windows XP Professional SP3, UNIX (Solaris) and Linux (Centos)). Each VM is assigned one core of the physical node and 3 GB of RAM. Each node runs the VMware system that manages the communications among the VMs, and one 24 port Procurve Switch (10/100/1000 ports) for data networks and another 24 port Procurve Switch (10/100 ports) for console management.

3.1.3 CIDS Parser and Summarizer Approach

A clear, summarized, and readable alarm report is fundamental for the cloud administration. Since the high scalability of a cloud implies that a NIDS component produces an intensive number of alerts, this component reduces the number of alerts. Among the approaches to summarize and integrate NIDS alerts, we recall, [144, 145]. A more suitable and clear approach to store NIDS alerts is given in [146] that is based upon the alert parameters shown in Table 3.1.

Table.3.1: An example for the alert description table.

Alert ID	Destination IP	Signature ID	Alert Description						
			Signature Name	Signature Class ID	Signature Priority	Source IP	IP Protocol	Source Port	Destination Port
A1	1.2.3.4	11	DOS	1	3	1.1.2.3	TCP/IP	50	70
A2	1.2.3.8	13	Rootkit	3	5	1.2.3.3	TCP/IP	20	60
A3	1.2.3.8	13	Rootkit	3	5	1.2.3.7	TCP/IP	30	72
A4	1.2.3.4	11	DOS	1	3	1.1.2.3	TCP/IP	50	70
A5	1.2.3.12	13	Rootkit	3	5	1.2.3.6	TCP/IP	122	17
A6	1.2.3.4	11	DOS	1	3	1.1.2.3	TCP/IP	50	70
A7	1.2.3.12	14	Buffer Overflow	2	4	1.1.2.4	TCP/IP	152	23
A8	1.2.3.8	13	Rootkit	3	5	1.2.3.3	TCP/IP	30	82

To summarize the alerts, CIDS exploits the idea that one alert suffices if several hosts are attacking the same machine using the same attack signature.

For this purpose, it merges all the alerts with the same combination (destination IP, attack signature) into one alert. Our implementation uses SNORT with MySQL. The summarization approach neglects the source IP address because it can be spoofed. However, the final summarized table would contain all information that describes the attack including the source IP address. Table 3.2 shows the final alerts produced by our summarization approach.

Table.3.2: The final alerts summarization table.

Alert ID	Destination IP	Signature ID	Alert Description						
			Signature Name	Signature Class ID	Signature Priority	Source IP	IP Protocol	Source Port	Destination Port
S1	1.2.3.4	11	DOS	1	3	1.1.2.3	TCP/IP	50	70
S2	1.2.3.8	13	Rootkit	3	5	1.2.3.3, 1.2.3.7	TCP/IP	20 30	60, 72, 82
S3	1.2.3.12	13	Rootkit	3	5	1.2.3.6	TCP/IP	122	17
S4	1.2.3.12	14	Buffer Overflow	2	4	1.1.2.4	TCP/IP	152	23

We note that alerts A1, A4, and A6 are summarized by S1 because they refer to the same signature, their attacks target the same machine and the attacker uses the same method three times. The alerts A2, A3, and A8 have the same signature but with different signature details. The attackers fired these attacks from two different host machines. These alerts are summarized to alert S2 in Table 3.2. Finally, the attacks related to the alerts A5 and A7 have not been summarized because they target the same machine but their signatures differ. Algorithm 3.1 shows the parsing and summarization processes.

Algorithm 3.1: The parsing and summarization processes

01: Begin
02: Build Table *T with rows= n //This table is similar to table 3.1.*
03: Define*:*
dest-ip=1, sig-id=2,
i=1, // Index for rows of table T

alert-dscrp-strct = T(1)(signature-name, signature-class-id, signature-priority, score-ip, ip-protocol, source-port, destination-port) // Is a structure contains one record of table T with 7 columns of alert description (from 4 to 10 of Table 3.1),

summarized-T: // This table is similar to table 3.2.

```
04: While ( Length(T) >1  and   i < Length(T) )
05:     For j=i+1  to  Length(T) do
06:       If (( T(i, dest-ip) = T(j, dest-ip)) And (T(i, sig-id) =T(j, sig-id))
            And (T(i, alert-descrp-strct) = T(j, alert-descrp-strct)))Then
07:          Add      the i^th record to table summarized-T
08:          Delete   the i^th and the j^th records from table T,  set i=1
09:       Else
10:         If ((T(i, dest-ip)=T(j, dest-ip)) And (T(i, sig-id)=T(j, sig-id))
             And (T(i, alert-descrp-strct)!=T(j, alert-descrp-strct))) Then
11:                Merge  the i^th and the j^th records of table T and add the resultant
                   merged record to table summarized-T
12:          Delete  the i^th and the j^th records from table T,  set i=1
13:         End If
14:       End If
15:    End For
16:    i=i+1
17: End While
18: If (T is not Empty)
19:        Add   table T to table summarized-T
20: End IF
21: Return (summarized-T)
22: End
```

3.2 CIDS-VERT, the Full Virtualization Framework of CIDS

To define a defense strategy for several cloud deployment models i.e., private, public, and hybrid clouds, we have specialized the original framework. This results in two frameworks: the original CIDS and CIDS-VERT, its specialization. CIDS-VERT has been defined to improve the scalability of CIDS because the experiments reported in Chapters 6 and 7 show that the P2P architecture of CIDS may hinder both scalability and elasticity. Furthermore, while CIDS isolates task execution from the host OS, most of its components are exposed to attacks because they run in the host operating system. CIDS-VERT avoids all these shortcomings and achieves a reasonable performance even in large clouds. This may promote the adoption of CIDS in large systems such as hybrid or public clouds.

3.2.1 CIDS-VERT Architecture

While most CIDS-VERT components are similar to those of CIDS, See Figure 3.2, its architecture is centralized with full virtualization, backup, and

task scheduling facilities. We now briefly describe the main components and facilities of the framework.

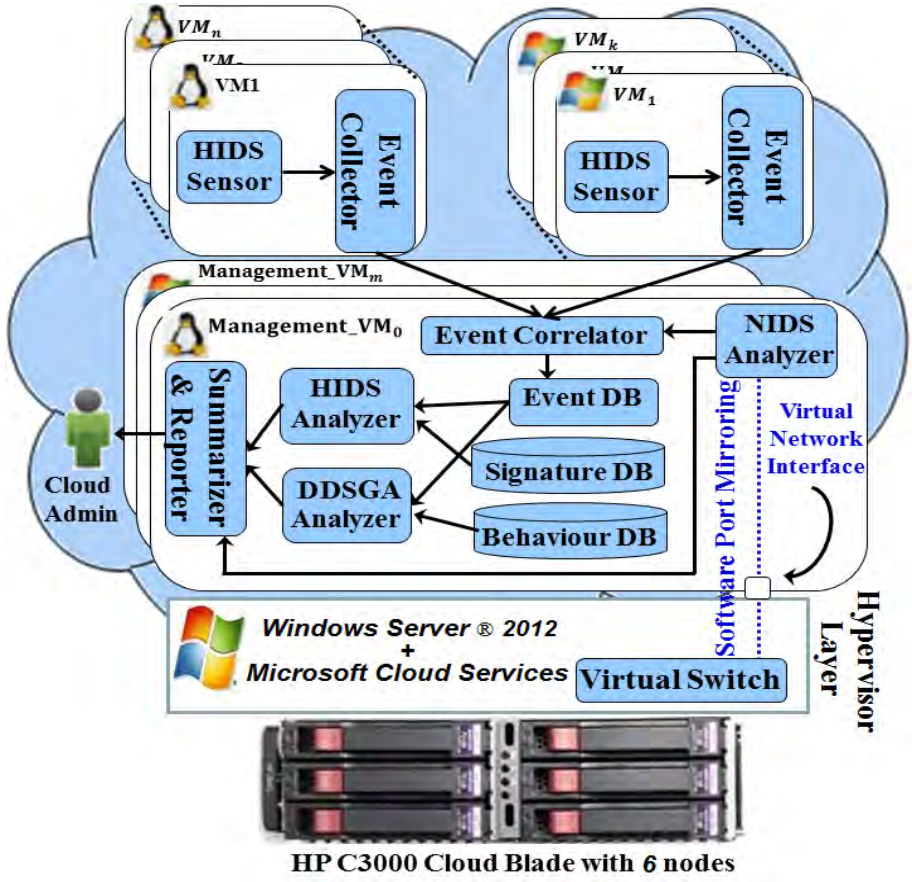


Figure.3.2: CIDS-VERT Architecture

Event collector: it collects logs, audit data, and sequence of user actions and commands from both HIDS sensor and the guest operating system. It also selects the most suitable management VM to analyze these audits and events.

Event Correlator: it correlates the user logs and signatures collected from several sources according to the start and end time of the session and the source IP address of the user. Then, it sends a final list of network and VMs environments events to the event DB. This helps in detecting a suspected behavior of a user that is distributed among several VMs. Since a behavior deviation in one VM can be normal in another one, this also reduces the false alarms rate.

Behavior-based database: it is a profile history database for the behavior of cloud users. It enables both CIDS and HIDS analyzers to compare the user behaviour in the current session against the stored profile.

Knowledge-based database: it stores a set of rules and signatures that describes known attacks. It describes a malicious behavior with a rule to be matched against those in the database.

DDSGA Analyzer: it analyzes the user behaviors, e.g. sequence of commands or actions, collected from several sources by applying the DDSGA approach. Whenever it detects a masquerade attack, it alerts the summarizer and reporter component.

HIDS Analyzer: it detects known trails left by attacks or predefined sequences of actions that might represent an attack in the user behavior. It is implemented by the OSSEC IDS tool that receives the user logs and signatures from all VMs in the cloud and determines whether a rule in the knowledge base is being broken. After that, it computes the probability that a user action represents an attack, and communicates it to the summarizer and reporter component.

NIDS Analyzer: it implemented by the SNORT IDS tool that analyses the VM network traffic to detect known trails that might represent an attack. The IDS can detect both network and DDoS attacks among Cloud zones and it receives the network traffic among the cloud VMs by mirroring it from the virtual switch. At first, SNORT determines whether a rule in the signature database is being broken. Then, it communicates to the summarizer and reporter component the probability that a user action represents an attack.

Summarizer & Reported component: It parses and summarizes the alerts fired by the HIDS and NIDS analyzers and correlates them. We use the IDMEF as a standard data format to summarize, integrate, and report the alerts about suspicious events. The most obvious solution is that the data channel from the intrusion detection analyzer to the manager that receives the alarms uses IDMEF. Chapter 8 details our integration and correlation approaches to integrate and summarize the alerts.

Management VMs: These VMs are reserved for all the components previously described and only be accessed by the cloud administrators that can manage all these components from one place. This also help to isolate and protect the components provided that the management VMs themselves

are protected. To improve scalability and avoid a single point of failure, the cloud runs several management VMs with distinct OS. These VMs are fully interconnected to provide backup sources and to mutually exchange the detection task to avoid overloading the active one, see Figure 3.3.

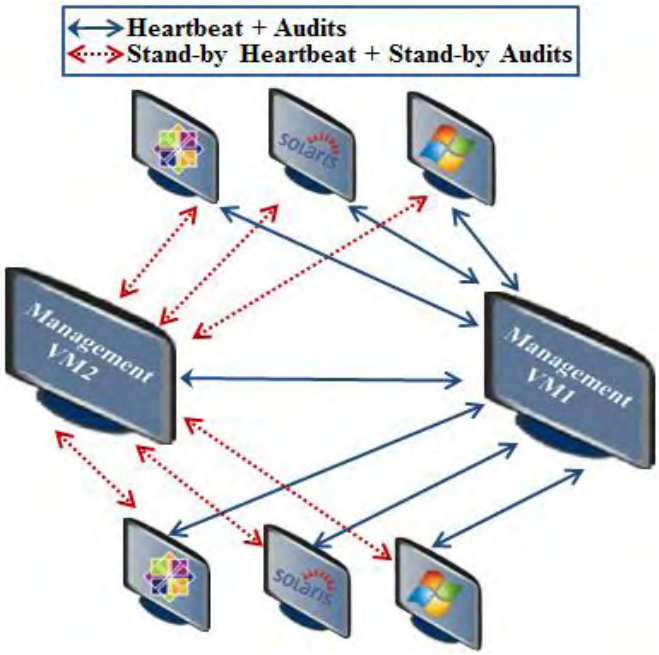


Figure.3.3: Data exchange among the management VMs.

The solid arrows in Figure 3.3 represent the audits sent to the management VMs and the heartbeat message with two fields. The first one defines the status of the active management VM to determine whether or not it is live. If the event collector does not receive the heartbeat message from the active management VM, it assumes that this VM is failed or is overloaded and switches to another management VM. The second field describes available resources in the active management VM e.g. processor speed, cache, and main memory, to enable the event collector to choose a management VM with proper resources to run the detection task. The dot lines in Figure 3.3 represent the interactions that occur if the active management VM fails or it is highly overloaded. All the idle management VMs are frequently updated with the user audits from the active management VM.

Hypervisor Layer: It manages the creation of virtual VMs, virtual networks, and virtual SAN driver. Furthermore, it provides system security

functions such as Isolation, Inspection, and Interposition. CIDS-VERT can work with different cloud environments, such as VMware cloud [56], Microsoft private cloud [13], Open Stack [147], Eucalyptus [12]. In CIDS-VERT deployment, we use the Microsoft windows server 2012 with its Hypervisor and the Microsoft Private Cloud system.

3.2.2 CIDS-VERT-Testbed:

The testbed consists of HP c3000 Cloud blade with six nodes. The first node is the head node that works as a front side interface for the cloud 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. The other five computing nodes are configured as the CIDS-Testbed nodes and their VMs are configured as those in the CIDS-Testbed. The only difference is that all nodes run a Microsoft core windows server 2012 instead than a VMware system. The head node runs a Microsoft GUI windows server 2012 with Microsoft cloud services and Microsoft Hypervisor manager 2012. The testbed also includes one 24 port Procurve Switch (10/100/1000 ports) for data networks and another 24 port Procurve Switch (10/100 ports) for console management.

3.3 Choosing the Proper Framework:

The size and the deployment model of the cloud system are the important issues that help us to select the proper intrusion detection framework.

The original CIDS framework is the ideal solution for a small cloud or private cloud behind the enterprise network firewall. Here, the management and deployment are taken care by the enterprise. The security of data in a private cloud is preserved by internal processes and data exchanged among the cloud nodes can be protected without violating the user security policies [148, 2]. This is actually what the CIDS framework needs. Private Cloud and the other deployment models use a distinct mechanism for data availability and service access. Most cloud deployment models leverage multiple copies of files on multiple nodes and consider each node as a failure domain so that server malfunctions do not crash the whole cloud nor result in data loss [148, 2]. The architecture of CIDS allows for backing up the data to avoid a single point of failure and to match the cloud robustness requirements.

The CIDS-VERT framework offers a security solution for large cloud systems e.g., public and hybrid clouds because its scalability is much better than that of CIDS. Furthermore, CIDS–VERT can be configured and managed in a simpler way than CIDS, because the administrators can access a central system backed up to other servers. This is important in public and hybrid cloud where the providers need to deploy and monitor the security solutions in a flexible way due to the large number of users. Even if CIDS-VERT works with any deployment model because of its scalability and controllability, it targets public and hybrid clouds while we use the original CIDS in private clouds. This strategy can achieve the performance and low network overhead of the Independent model that works with the original CIDS but not with CIDS-VERT that is centralized. Furthermore, CIDS-VERT may not be acceptable in a small or private cloud because it does not optimize resource utilization due to the adoption of several management VMs.

3.4 Attacks and Cloud Service Models Covered by CIDS

CIDS and CIDS-VERT satisfy the cloud IDS requirements mentioned in Section 2.1.4 and deal with attacks against SaaS, PaaS and IaaS clouds.

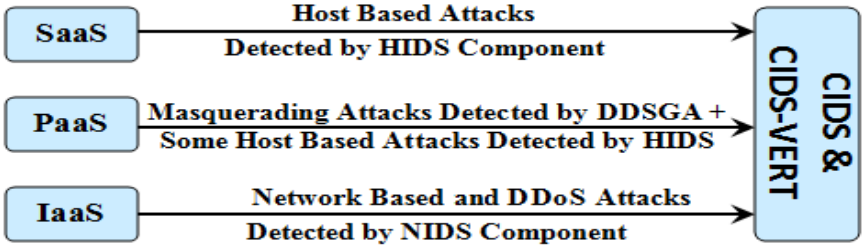


Figure.3.4: Attacks and cloud service models covered by CIDS.

As shown in Figure 3.4, the proposed frameworks can deal with the following attacks:

(1) Masquerading attacks:

This is a PaaS attack that impersonates a legitimate user to use service resources maliciously. This is by far the most critical attack as its exploitation is rather easy. CIDS and CIDS-VERT detect it through DDSGA.

(2) Host-based attacks:

Host based attacks may be a consequence of a masquerading attack. CIDS and CIDS-VERT detect several host based attacks using the current HIDS tools.

(3) Network-based attacks:

CIDS and CIDS-VERT detect network attacks by analyzing network packets using NIDS tools.

(4) DDoS attacks:

We have built two deployments for CIDS-VERT, the Centralized and Distributed, to detect the DDoS attacks. Chapter 8 describes the two deployments and their experimental results.

3.5 The Correlation Models

In the following, we describe the three alternative correlation models to correlate and exchange the audit and alerts between the IDS components in the cloud system. These models are [142, 143]:

(A) Audit exchange.

(B) Independent.

(C) The Centralized-Backup.

The first two models work with CIDS framework, while the third one with CIDS-VERT.

(A) Audit exchange model

In this model, nodes exchange their audit data so that each one has any audit data for its current users. The detection phase depends on two parameters:

(1) The alignment score computed in the CIDS detector component,

(2) Alerts fired by the HIDS component.

In this way, the detection overhead is balanced among nodes with no single point of failure. The detection efficiency is high because the user audit is concentrated in one place and the masquerader surviving is very short. As a counterpart, this model introduces some overhead in the cloud network due to the periodic exchange of audit data. The processing steps are:

If *user HIDS* ***or*** *CIDS instance fired* ***Then*** *// If this condition is satisfied, this denotes that an attack has been detected (Host-based or masquerade attack).*

1. *Alert all nodes that have VM instance(s) for that user to stop exchanging his audit data.*
2. *Send alerts to the scheduler node to do the following tasks:*
   a) *Stop the current tasks related to this user from all his VMs. If the alert is coming from HIDS detector then, stop only this malicious VM.*
   b) *Prepare a summarized report to the cloud administrator contains some information about the masqueraded user, the malicious VMs, and the detected attack.*
   c) *Apply the administrator action against this user by re-initializing his malicious VM(s) or by Blocking or suspending his account.*

End if

(B) The Independent Model

Each cloud node evaluates its own user audits without interacting with other nodes. The detection phase depends upon the same two parameters of model *A*. Login usage patterns for a user are evaluated using both CIDS and HIDS detectors inside a cloud node *CN* and by using the behavior-based and signature-based of *CN* only. If the HIDS detector of *CN* fires an alert, the algorithm will behave according to step 2 of model *A* for each user HIDS instance firing. If the CIDS detector of *CN* fires an alert, the algorithm checks the current login usage patterns against the audit data of the current user in the other nodes. The user is marked as a masquerader unless one node accepts the current pattern. Then, this model will behave according to step 2 of model *A* for each user CIDS instance firing. Algorithm 3.2 shows the steps of model *B*.

The model advantages are:

(1) A very low overhead for the cloud network, as there is a data exchange only if the score DS_i is less than the previous define threshold θ_{DS}. The nodes exchange the test audit data (*test_d*) produced by the user during the login session.

(2) A lower processing overhead for each cloud node than models *A* and *C*, because each node executes the DDSGA alignment of the current login session, only if DS_i is less than θ_{DS}.

The disadvantages are:

(1) A longer masquerader surviving than both models *A* and *C* because the analysis requires a long time to check the audit data (test_d) in all nodes.

(2) A lower hit rate and a higher false alarm rate than model *C*. Instead, its hit and false alarm rates are similar to those of model *A*.

Algorithm 3.2: The analysis algorithm for the Independent model

01: Begin
02: Inputs*: test audit data (test_d) produced by user (i) during the current login session, behavior-base(behavr_d) stored for user (i) during the training phase inside the current login cloud node,* DS_i *is the DDSGA alignment Score for user i,* θ_{DS} *is the alignment threshold defined for user i, Not-Masq-flag = False.*
03: *Use DDSGA to compute* DS_i *by aligning (test_d) against (behavr_d).*
04: If $DS_i < \theta_{DS}$ ***Then***
05: ***For*** *each cloud node (CN) contains (behavr_d) of user i,* ***do****:*
06: *Use DDSGA to compute* DS_i *for the* i^{th} *user in CN*
07: ***If*** $DS_i \geq \theta_{DS}$ ***Then***
08: *Not-Masq-flag =* ***True***
09: *Exit the loop;*
10: ***End if***
11: ***End for***
12: End If
13: If *Not-Masq-flag = false* ***or*** *HIDS instance is fired* ***Then***
14: *Run step 2 of model A for each user i IDS instance firing.*
15: End If
16: End

(C) The Centralized-Backup Correlation Model

This model works with the CIDS-VERT framework where, users VMs send their audit data to a reserved management VM that has a complete view of audit data for all users to analyze and report the final alerts. The management VM is backed up to some other VMs as explained in Section 3.2.1 to balance the detection overhead among the management VMs with no single point of failure. This model achieves the best detection efficiency because the user audit is concentrated in one place and there is not loss of the audit data. The masquerader surviving is shorter than that in both model *A* and *B*. The detection time is inversely proportional to the number of

management VMs because it reduces the processing overhead in the active management VM. This speeds up the detection phase and protects the IDS components from tampering by any attackers. On the other hand, the network overhead increases with the number of management VMs. Furthermore, the model requires several resources as it reserves some management VMs for detection.

The experimental evaluation of the two frameworks and their corresponding correlation models are detailed in Chapters 6 and 7.

Chapter 4

Cloud Intrusion Detection Dataset (CIDD)

This chapter introduces the Cloud Intrusion Detection Dataset, CIDD [149], the dataset we have defined to test and train an IDS and that will be used in the thesis. The chapter discusses the major challenges to build a cloud intrusion dataset. Furthermore, it introduces the Log Analyzer and Correlator System (LACS) that has supported the building of CIDD by parsing and analyzing user's binary log files and correlating user audits data. Finally, the chapter describes the distribution of attacks and masquerades in CIDD and compares CIDD against other publicly available datasets.

4.1. Challenges to build a cloud dataset

We have detailed in Chapter 2, the deficiencies of current publically available datasets which hinder their adoption to evaluate cloud IDSs. Building a real intrusion dataset for the cloud systems is a complex task, because it takes a long time to collect the training audits and to prepare the scenarios for both training and testing phases. Furthermore, data collection requires special tools to access and monitor the cloud infrastructure and system that require proper authorization to preserve privacy and confidentiality. These are major challenges in cloud systems for several reasons:

(1) Lack of real data to study available solutions and models. Data are out of reach and controlled under the rules of evidence, rather than being a source of valuable information for research purposes. Most cloud systems are commercial and the control of their infrastructures is very difficult if it is not impossible. Private cloud systems cannot be accessed by external users and this hinders the building and the analysis of complete attack scenarios.

(2) It is difficult to collect real data about a malicious or a legal user if audits are distributed across different environments. The heterogeneity of the audit parameters increases the complexity of audit correlation. The complexity is even larger for low level formats. It is also difficult

to build a summarized statistical profile for each user, because categorizing a set of UNIX commands differs from categorizing Windows events and applications.

(3) The huge size of the audit data for cloud systems (more than 20GB for CIDD dataset) and the high number of users require huge computing resources.

4.2 Cloud Intrusion Detection Dataset (CIDD)

To overcome the problems previously outlined, we have developed a Log Analyzer and Correlator System (LACS) to parse and correlate user audits from low level log files. We have applied LACS to logs from the DARPA Intrusion Detection Evaluation Group of MIT Lincoln Laboratory [150]. The logs and the TCP dump data are from the Eyrie Air Force Base network that consists of two segments, representing the inside and outside of a government installation. The outside segment consists of 10 machines running Solaris, Linux, and SunOS, 1 SNMP monitor, 1 gateway, and 1 web server. The inside segment consists of 35 machines with Windows NT, Solaris, Linux, and SunOS, 1 inside gateway, and 1 inside sniffer. The network has also 2 routers and 2 hubs. The log files focus mainly on the network audit data. However, we have extracted the host and network audits by parsing the log files collected from, respectively, one Windows NT machine, one Unix Solaris machine, and the raw packet data collected through TCP-dump so that CIDD considers both network and host audit data. These data are correlated according to user IP address and audit times. The following section describes the architecture of LACS.

4.2.1 LACS System

LACS parses the binary log files collected by Unix Basic Security Module (BSM), the security, application and service log files of the Windows event log system, and data in raw packets. In the following, we briefly describe the component of LACS in Figure 4.1:

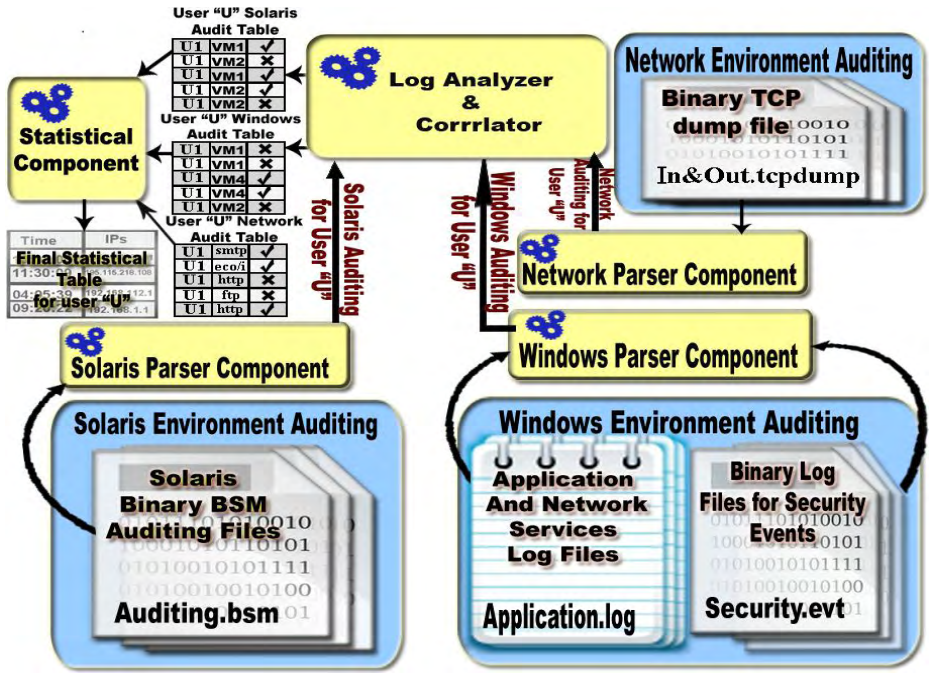


Figure 4.1: The architecture of LACS system

A. Parsers components: LACS has 3 parser components for each independent environment:

1- Solaris parser: The Solaris C2 audit daemon, e.g. the auditing capability of BSM, writes binary event data to the local file system. Our parser converts the audit events of this file into a readable text format and stores its output in a local file while preserving the ordering of events. This file can be analyzed by the log analyzer and correlator component. The parser extracts the parameters shown in Figure 4.2: user id, user name, day, time, system calls, path (for processes or files), login source (IP address or URL), session id, login period, audit part, VM name, and return value (success or fail).

Order	Day	User ID	User Name	Time	MSEC	Date	System Call	Path	return	Source IP	Sess ID	Login Period	Audit Part	VM Name
2	D1W1	2049	lucyj	09:30:48	26992275	06/01	setaudit	-	success	falcon.eyrie	1118	Morning	VM-part2	VM3
14	D1W1	2053	reynaldv	09:27:00	23500108	06/01	open-rea	/etc/sec	fail	duck.eyrie.	1087	Morning	VM-part2	VM4
24	D1W1	2054	selmam	08:09:35	56938041	06/01	login - teln	-	success	pigeon.eyri	463	Morning	VM-part1	VM3
31	D1W1	2055	yuvalt	13:56:33	28408787	06/01	exit	-	fail	swan.eyrie	2968	AfterNoon	VM-part2	VM4
36	D1W1	2056	kiara	08:07:50	21775685	06/01	setaudit	-	success	finch.eyrie.	439	Morning	VM-part2	VM2
43	D1W1	2058	jyacintl	09:22:37	66175489	06/01	login - teln	-	success	eagle.eyrie	1035	Morning	VM-part1	VM1
58	D1W1	2059	gwendolv	08:31:04	24870200	06/01	exit	-	success	falcon.eyri	631	Morning	VM-part1	VM3

Figure 4.2: An example of CIDD Solaris auditing data

2- Windows parser: It converts into a human readable format the primary binary/encoded Windows security event, the application, and the service log files. It stores its output in a local file to be analyzed by the log analyzer and correlator component. The parser extracts from the security event log files the parameters in Figure 4.3: type (audit success or fail), date, time, event id, source (security log in this case), audit category (e.g., system event, object access, detailed tracking, privilege use, logon/logoff, account management), user id, user name, VM name, audit action, and audit parameters (e.g., object name, handle id, privileges). The parser extracts form the application and service log files the information in Figure 4.4: source machine (IP address or URL), user name, date, time, service or application name, source and destination port, target.

Order	Type	Date	Time	Event	Source	Category	Domain\User	User Name	VM Name	Audit Action	Audit Parameter
63	Audit Success	29-Mar	14:56:29	538	Security	Logon/Logoff	\S-1-5-21-742865521-1025978620-313593124-500	Administrator	vm1	User Logoff	User Name: administrator
305	Audit Success	29-Mar	15:34:56	528	Security	Logon/Logoff	\S-1-5-21-742865521-1025978620-313593124-500	Administrator	vm1	Successful Logon	Logon Process: User32
29360	Audit Success	30-Mar	03:03:06	538	Security	Logon/Logoff	\S-1-5-21-742865521-1025978620-313593124-	alie	vm1	User Logoff	User Name: alie
22506	Audit Success	29-Mar	16:14:25	528	Security	Logon/Logoff	NT AUTHORITY\ANONYMOUS	ANONYMOUS	vm1	Successful Logon	Logon Process: KSecDD

Figure 4.3: An example of CIDD Windows auditing data

Order	Source Machine	Users	Date	Time	Service nan	Src Prt	Dest Prt	Target
1	197.218.177.69	anonymous	3/2	8:47:37	MSFTPSVC	0	16	anonymous
2	197.218.177.69	dietmars@pluto.plum.net	3/2	8:47:38	MSFTPSVC	484	30	dietmars@pluto.plum.net
3	197.218.177.69	dietmars@pluto.plum.net	3/2	8:47:47	MSFTPSVC	31	169	/usr/bin7/gnudoit
4	197.218.177.69	dietmars@pluto.plum.net	3/2	8:47:58	MSFTPSVC	0	6	-
5	172.16.113.84	anonymous	3/2	8:48:37	MSFTPSVC	0	16	anonymous

Figure 4.4: Examples of training data (sequences of mails and web services)

3- Network parser: It extracts user audits from the raw packets data files collected by TCP-dump that contains information on the activities of the user source machine. The parser extracts from the TCP-dump files the values in Figure 4.5: date, time, duration, service/protocol name, source port, destination port, source IP, destination IP. If the packet is contaminated by an attack, then also the attack name is extracted.

Order	Date	Time	Duration	Service Name	Source Port	Dest Port	Source IP	Destination IP	Attack?
46	27-Jul	8:02:59	0:00:01	snmp/u	161	1860	192.168.001.001	194.027.251.021	-
47	27-Jul	8:03:04	0:00:01	snmp/u	1862	161	194.027.251.021	192.168.001.001	-
48	27-Jul	8:03:04	0:00:01	snmp/u	161	1862	192.168.001.001	194.027.251.021	-
49	27-Jul	8:03:07	0:00:01	snmp/u	1864	161	207.230.054.203	192.168.001.001	snmpgetattack,40,*,y,r2l
50	27-Jul	8:03:09	0:00:01	snmp/u	1866	161	194.027.251.021	192.168.001.001	-
74	27-Jul	8:03:35	0:00:01	ntp/u	123	123	172.016.112.020	192.168.001.010	-
75	27-Jul	8:03:39	0:00:01	snmp/u	1888	161	194.027.251.021	192.168.001.001	-
130	27-Jul	8:04:45	0:00:01	snmp/u	161	1950	192.168.001.001	207.230.054.203	snmpgetattack,40,*,y,r2l
131	27-Jul	8:04:47	0:00:01	http	8466	80	172.016.114.168	207.077.090.015	-

Figure 4.5: A snapshot of TCPdump data with labeled attacks

B. Log analyzer and correlator component: This is the core component and its analysis includes the following steps :

(1) It correlates the user audits in host and network environments using user IP and audit time. Then it links each audit to the corresponding user.

(2) It pairs user audits with a set of VMs according to their login sessions time and the characteristic of the user task. During audit collection, each user logs into the network in one or two different time shifts, one in the morning and the other in afternoon or evening and sometimes both. It also assigns user morning sessions to one VM and the other sessions to another VM. Section.4.2.2 describes the distribution of users to the VMs.

(3) It marks malicious audit records according to attacks and masquerades tables given by MIT group [150]. The marking is done according to attack time, date, destination IP/URL and the name of victim user. It also marks some audit records in a session with different time and/or different source IP than the training audit data stored for the user.

(4) It produces the final tables with the marked audits for each individual user with its assigned VMs. This step produces three tables namely, Solaris, Windows, and network audit tables. Both the Solaris table and the Windows one contain a sequence of user actions. The Network table records the sequence of machines, network services and protocols accessed by the user, and normal times and dates of accesses. These tables enable any IDS to deduce the sequence of user audits in different environments. Equation 4.1 correlates the audits of the three tables:

$$P_{Cmasq}(U_i) = \sum_{j=1}^{m} \left(\frac{P(U_i) * P(IP_j)}{\sum_{k=1}^{n} P(U_k)} \right) + P(U_i) \qquad (4.1)$$

Where:

- $P_{Cmasq}(U_i)$*:* Probability that U_i is a masquerader according to U_i behaviors in any cloud node. It considers the probability that the masquerader can be detected by the login IP(s).
- $P(U_i)$: Probability that U_i is a masquerader according to U_i behaviors in any cloud node. It does not include user IP behaviors.
- *m:* Number of IP(s) that U_i uses to login to the cloud.
- *n:* Number of cloud users who share the same IP_j of U_i
- $P(IP_j)$: Probability that IP_j reveals to be a masquerader.

Consider, as an example, a simple case where U_1, U_2 and U_3 share IPs, IP_1 *and* IP_2. Also suppose that the probabilities that IP_1, *and* IP_2 could be used by a masquerader are: $P(IP_1) = 0.4$, and $P(IP_2) = 0.5$, and U_1, U_2, and U_3 reveal to be masqueraders according to their behaviors in all the cloud nodes with the following probabilities: $P(U_1)=0.4$, $P(U_2)=0.3$, and $P(U_3)= 0.6$, and the detection threshold $\theta = 0.75$. We apply the previous equation for each U_i. We have that only U_3 is a masquerader because:

$P_{Cmasq}(U_1) = 0.6769 < \theta$ (not masquerader)

$P_{Cmasq}(U_2) = 0.5076 < \theta$ (not masquerader)

$P_{Cmasq}(U_3) = 1.0153 > \theta$ (masquerader)

C. The statistical component: It uses the previous tables to build host and network based statistics. Host based statistics include: number of login failures, logging times (morning, afternoon, evening, and nights), logging source address(es), a list with:

(a) common commands and system calls used by the user (in case of Unix Solaris system),

(b) a list of common services, applications, and security actions(in case of Windows NT),

(c) VMs names used by each user.

Network based statistics are based on the IP address and include: network services and protocols used, machines accessed, hours and days when the IP becomes active, and list of failures.

4.2.2 CIDD Architecture

CIDD audit data consists of two parts. The first one is a collection of Unix Solaris audits and their corresponding TCP dump data. The second part includes Windows NT audits and their corresponding TCP dump data. As any intrusion dataset, CIDD includes training and testing data for both parts 1 and 2. In training data of part 1, CIDD has 7 weeks (35 days) of Unix Solaris audits and TCP dump data with labeled attacks which can be used to train any IDS with a set of attack signatures. Figure 4.6 shows the distribution of these labeled attacks. The UNIX audits of week 6 contains 21 real masquerade attacks that can be used to test any anomaly detection based IDS. Figure 4.7 shows the distribution of these masquerade attacks in week 6 data.

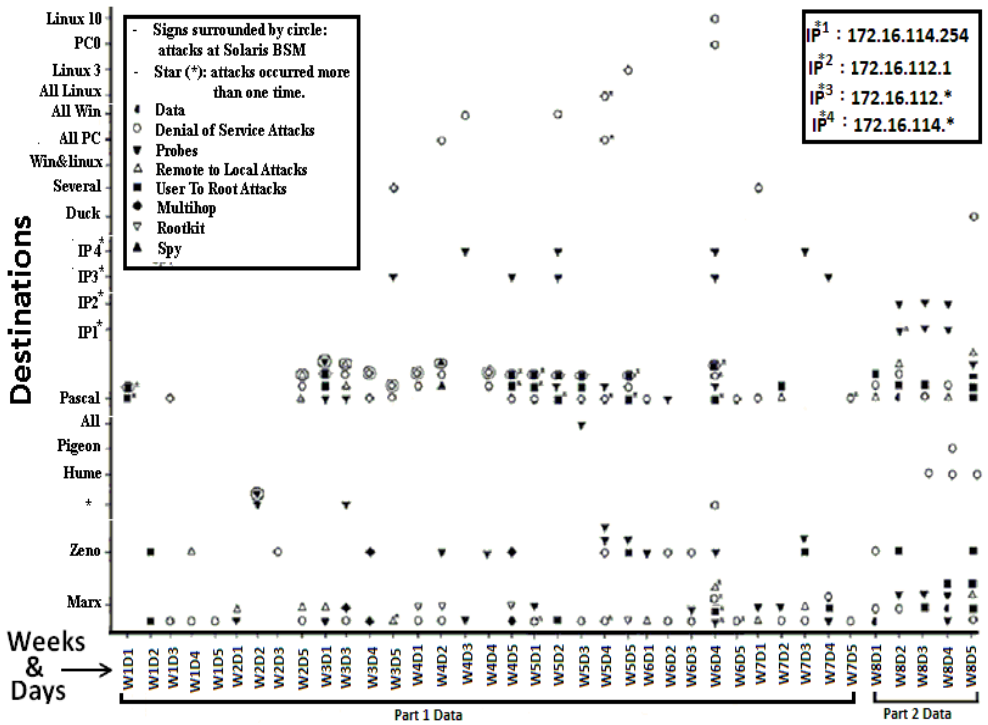


Figure.4.6: Attacks distribution in the training data (Solaris BSM, Windows Audits and TCP-dump data)

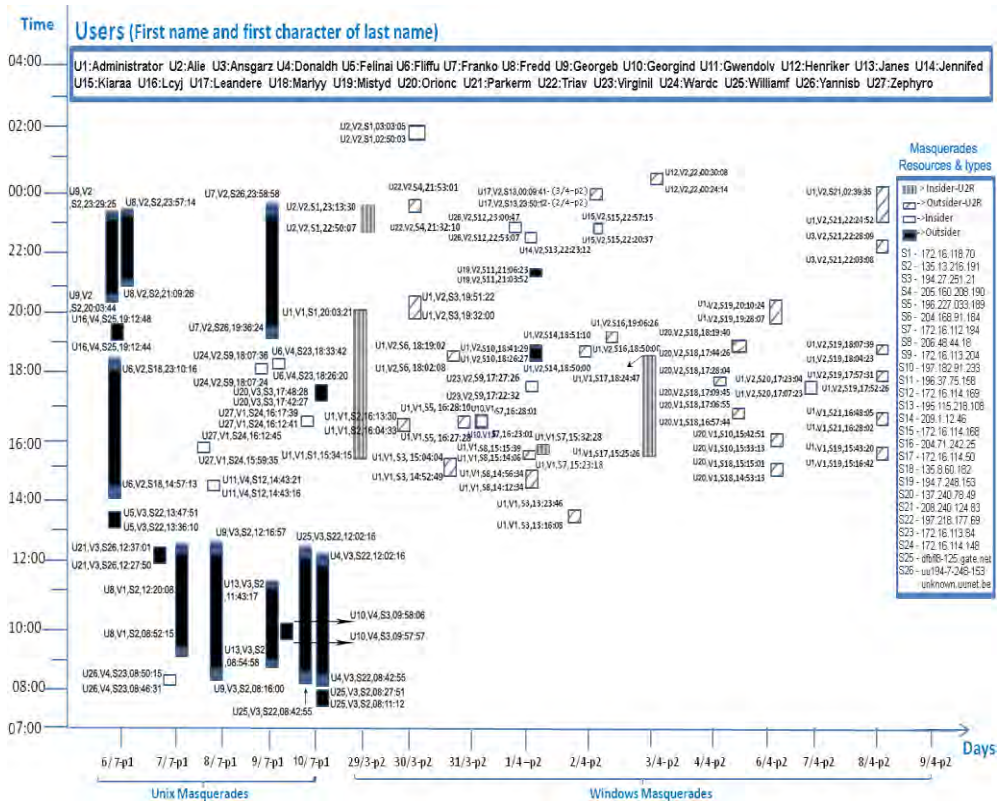


Figure 4.7: Masquerade attacks in week 6 of Part1 and the two testing weeks of part2

Most of audits of CIDD users are distributed across several VMs. Users with less than 5 login sessions have been deleted. CIDD has 84 Solaris users distributed into 4 VMs. Users are categorized according to the applications of their host machines, see Figure 4.8, into:

- 2 programmers sharing VM1 and VM2,
- 1 secretary using VM3 and VM4,
- 1 system administrator using VM3 and VM4,
- 56 normal users using VM3 and VM4 to issue UNIX commands, exchange mails, and internet navigation,
- 22 advanced users that access VM1 and VM2 to run advanced applications for some special tasks.

The testing data of part 1 includes 2 weeks (10 days) of Unix Solaris audits and their corresponding TCP dump data for testing purpose. The data

include more than one hundred instances of attacks that are classified into distinct categories such as Denial of Service (DoS), User to Root (U2R), remote to user, surveillance probing and anomaly attacks, see Figure 4.9. Figure 4.10 shows the distribution of these attacks in part1 testing data.

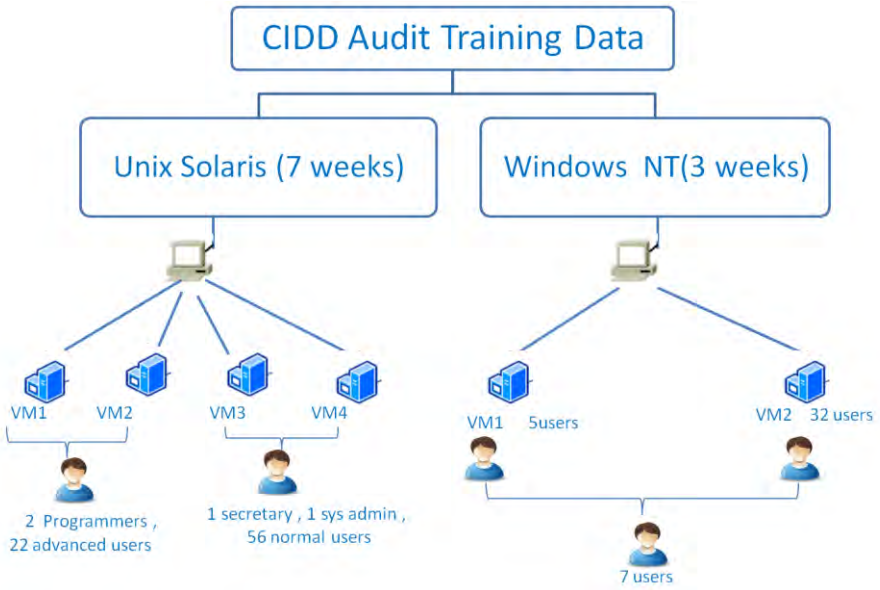


Figure 4.8: Users distribution in CIDD training part

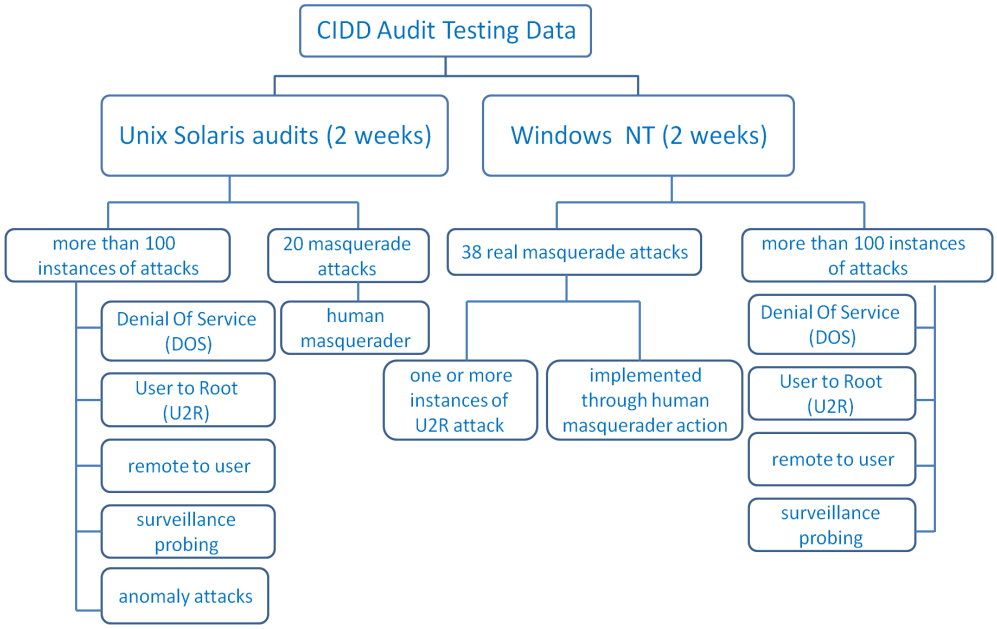


Figure 4.9: Attacks distribution in CIDD testing part

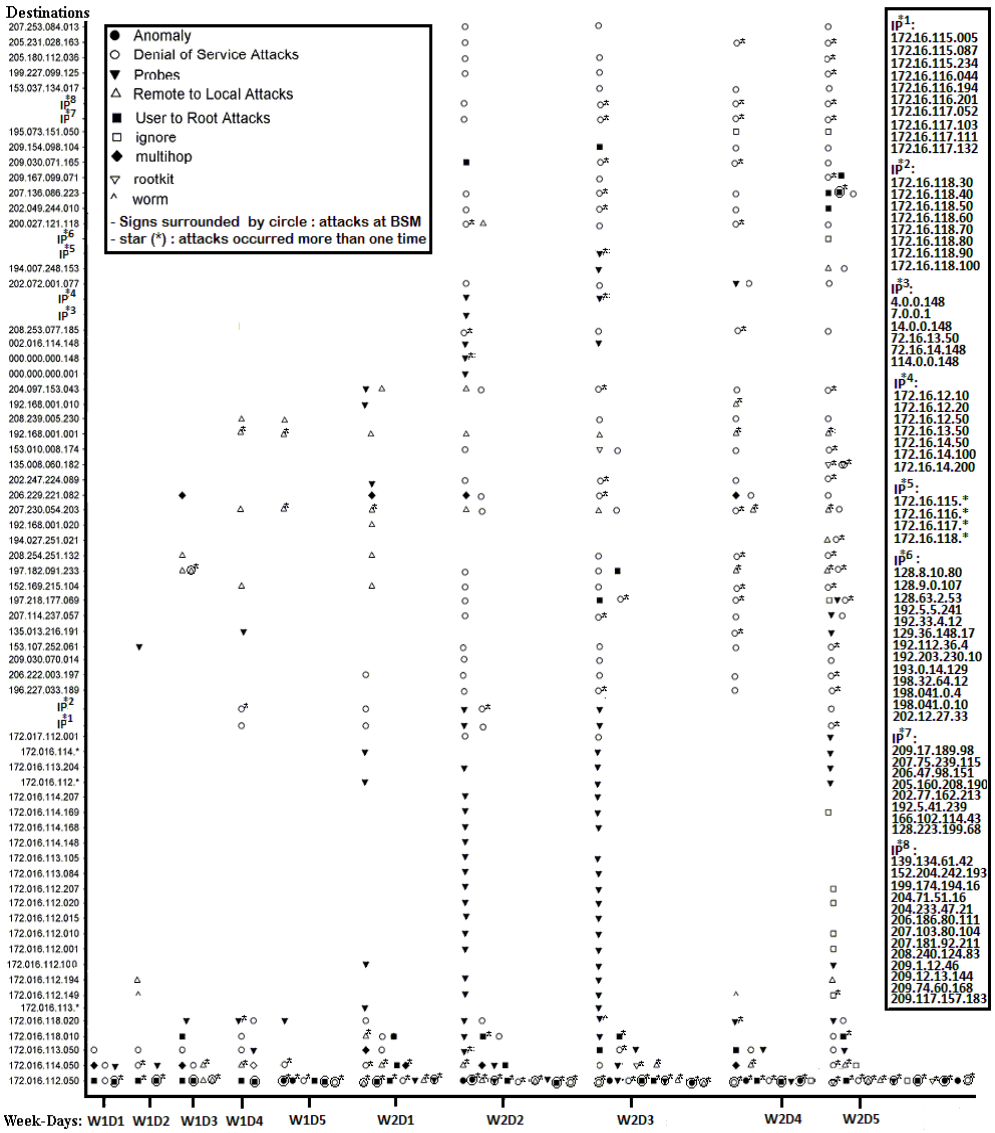


Figure 4.10: Attacks distribution in testing data of part1 (Solaris BSM and TCP-dump data)

The training data of part 2 includes 3 weeks (15 days) of Windows NT audits and their corresponding TCP dump data with labeled attacks only in the second week. Figure 4.6 shows the distribution of these labeled attacks.

CIDD describes 44 Windows NT users with a complete windows audit data. Some of these users exist in part1 audits with the same names. Users are distributed among VMs as in Figure 4.8: 5 in VM1, 32 in VM2, and 7 that share both VM1 and VM2.

The testing data of part 2, describes 2 weeks (10 days) of Windows NT audits and their corresponding TCP dump data for testing purpose. Part 2 testing data contains 38 real masquerade attacks in Windows NT audits. Some of these attacks result from one or several U2R attacks, while others are implemented through human masquerader action, see Figure 4.9. One user of the inside network segment implements masquerade attacks, while "outsider" are due to either users of the outside network or outside the network that is considered. Figure 4.7 shows the distribution of these masquerade attacks. Part 2 testing data has the same attack categories of part 1 and a further category, data attacks. Figure 4.11 shows the distribution of these attacks in part2 testing data.

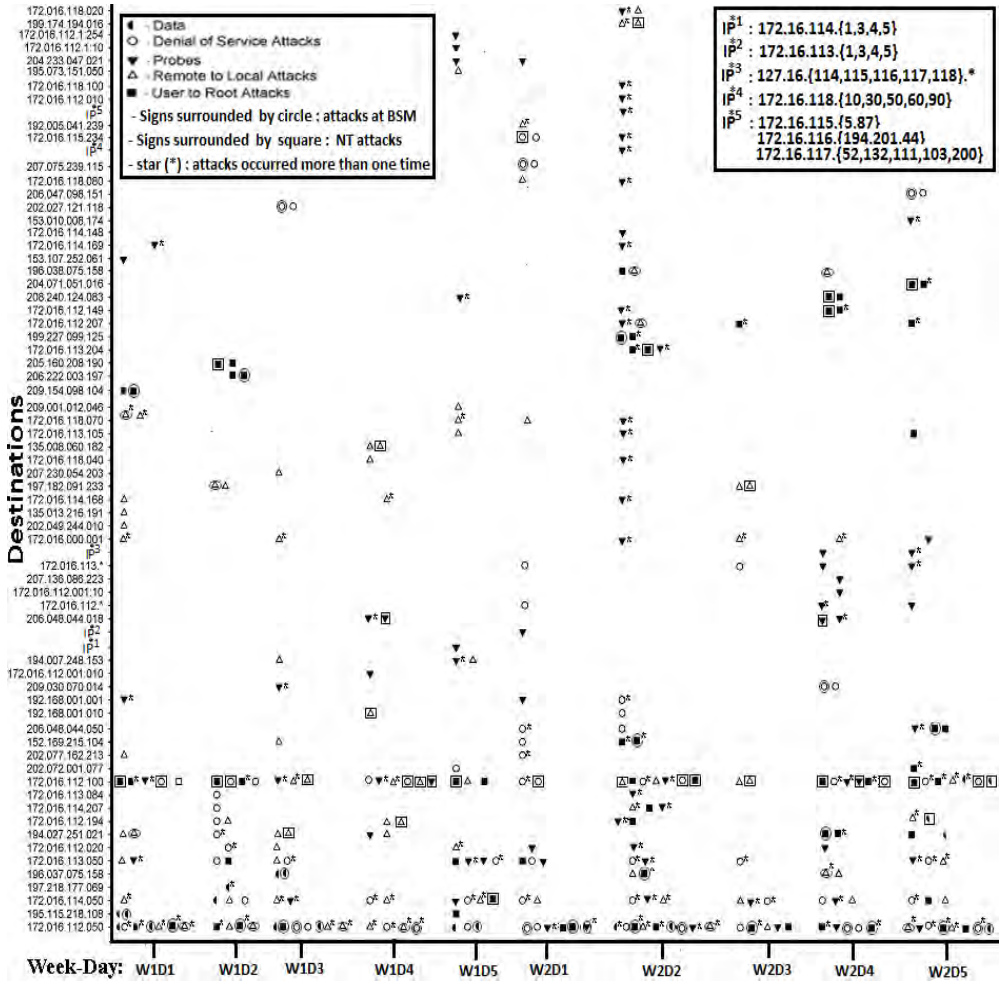


Figure 4.11: Attacks distribution in testing data of part2 (Windows audits and TCP-dump data)

The CIDD webpage [149] describes further details such as user statistic tables, masquerade distributions, the simulated network of the auditing experiments, attacks database and their categories, and snapshots for both training and testing data. Table 4.1 compares CIDD against publicly available datasets.

Table 4.1: Comparison of publicly available datasets

Dataset	Normal Users	Audit Format	Audit Parameters	Real Masquerades?	Audit Environment	Sessions	Attack Signatures
SEA	50	Unix commands	No	Simulated	Unix hosts	No	No
Greenberg	168	Unix commands	yes	Simulated	Unix hosts	Yes	No
PU	4	Unix commands	yes	Simulated	Unix hosts	Yes	No
RUU	34	Windows audits	yes	Real but with predefined scenarios	Windows hosts	Yes	No
CIDD	128 (84 in UNIX + 44 in Windows)	Complete Unix audits, Windows audits, TCP dumps	yes	Real time masquerades and more than 100 instances of real attacks	Hosts(Unix, Windows), Network, Grid and Cloud	Yes and assigned to VMs	Yes

Chapter 5

Data-Driven Semi Global Alignment (DDSGA)

A masquerade attacker authenticates as a legal user after stealing or cracking the user credentials or attacking the authentication service. Even it does not leave trails in the target system, this attack may be discovered through a masquerade detection process that matches the user active session against a profile of the previous behaviour of the same user and that signals any mismatching as an attack. Current profiling methods consider several features such as command line commands, system calls, security events, mouse movements, opened files names, opened windows title, and network actions. As mentioned in Chapter 2, masquerade detection has not yet achieved the level of accuracy and performance for practical deployment. Accuracy may be even lower in systems with a massive amount of resources, like grids and cloud systems, where a profile can be built only by correlating several user activities in distinct VMs.

This chapter introduces the Data-Driven Semi Global Alignment, DDSGA, the approach we adopt to efficiently detect masquerade attacks and anomalous actions. It also describes the three main phases of DDSGA namely, configuration, detection, and update. Then, it explains the implementation and the experimental results of each phase. Lastly, it compares DDSGA against other approaches, and highlights its computational performance and detection accuracy.

5.1 DDSGA Approach Overview

DDSGA is a masquerade detection approach based upon the Enhanced-SGA algorithm [125] described in Section 2.2.4. It aligns the user active session sequence to the previous ones of the same user and it labels the misalignment areas as anomalous. A masquerade attack is signaled if the percentage of anomalous areas is larger than a dynamic, user dependent threshold. DDSGA can tolerate small mutations in the user sequences with small changes in the low level representation of user commands and it is decomposed into a configuration phase, a detection phase and an update one. The configuration phase, computes, for each user, the alignment parameters

to be used by both the detection and update phases. The detection phase aligns the user current session to the signature sequence. The computational performance of this phase is improved by two approaches namely the Top-Matching Based Overlapping (TMBO) and the parallelized approach. In the update phase, DDSGA extends both the user signatures and user lexicon list with the new patterns to reconfigure the system parameters. Figure 5.1 shows these phases and the modules that implement them that we discuss in the following.

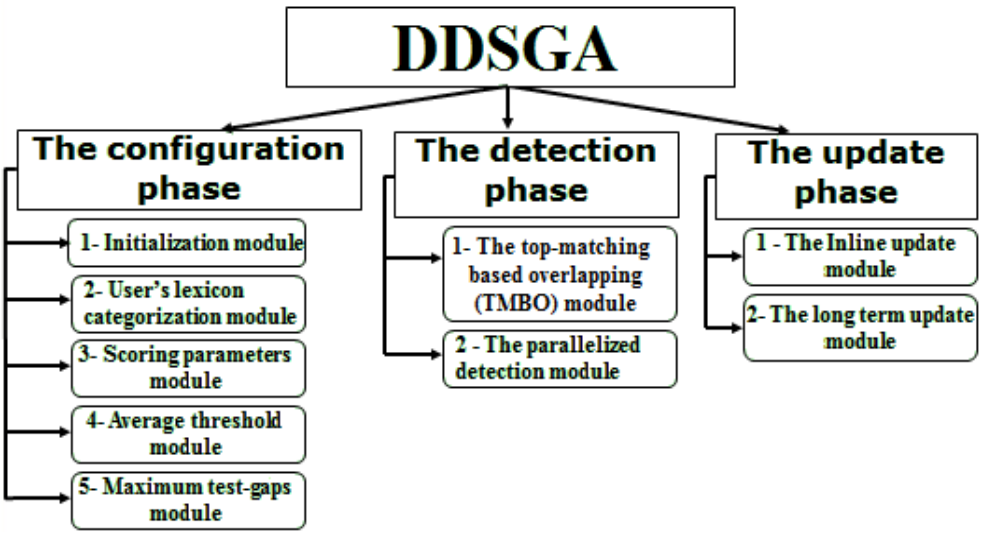


Figure 5.1: DDSGA Phases and modules

DDSGA Main Features and Improvements:

DDSGA improves both the computational and the security efficiency of the Enhanced-SGA.

From a computational perspective, DDSGA improves the performance of both the detection and the update through a parallel multithreading scheme and a new Top-Matching Based Overlapping (TMBO) approach that improves the Heuristic Aligning and saves computational resources. While the Heuristic Aligning splits the signature sequence into a fixed overlapped subsequences of size 2n, where n is the size of the test sequence, TMBO simplifies the alignment through shorter overlapped subsequences. Besides saving computational resources, this speeds up the detection and update phases and consequently reduces the masquerader live time inside the system.

With respect to the accuracy of masquerade detection, DDSA introduces distinct scoring parameters for each user, namely the gap penalties and the overlapping length. The adoption of distinct scoring parameters for each user improves the detection accuracy, the false positive and false negative rates and increase the detection hit ratio with respect to the traditional and enhanced SGA that use the same parameters for any user. This neglects differences among the behaviours of distinct users and reduces the accuracy of detection, because the alignment cannot tolerate even slight changes in the user behaviour over time. Starting from the data of each user, the configuration phase of DDSGA computes the scoring parameters that result in the maximum alignment score for the considered user.

Furthermore, to improve the accuracy of the alignment, DDSGA integrates binary and command group, two scoring systems suggested by Coull et al., into two other scoring systems, restricted and free permutation. The resulting systems tolerate permutations of previously observed patterns with a low reduction of the score. To tolerate changes in the low-level representation of commands with the same functionality, the scoring systems classify user commands into several groups and align two commands in the same group without reducing the alignment score. The DDSGA configuration phase also creates a dynamic threshold for each user to be used by both the detection phase and the update one. While Enhanced-SGA builds this threshold through a snapshot of a user profile, DDSGA builds a more sensitive and dynamic threshold by considering any data in the profile.

Furthermore, DDSGA runs two update modules: the inline and long term modules. The inline module updates the user signature patterns, the user lexicon list, and their corresponding command categories in a reconfiguration phase. The long-term module updates the system with the latest changes in the alignment parameters. It also updates the dynamic threshold values, scoring parameters, and the overlapping length i.e., the length of the overlapped signature subsequence according to maximum number of inserted test gaps. The dynamic threshold, the scoring systems, and the two update modules enable DDSGA to tolerate slight changes in the user behaviour overtime. Table 5.1 briefly compares DDSGA and Enhanced-SGA.

Table 5.1: A comparison between DDSGA and Enhanced-SGA

	Enhanced-SGA	DDSGA
Accuracy	1. Command grouping and binary scoring systems 2. Fixed and equal gab insertion penalties for all the users. 3. Compute the threshold for each user using snapshots of user date. 4. Signature Update.	1. Free and restricted permutation scoring systems. 2. Define gab insertion penalties for each user independently, based on user previous data. 3. Compute an optimal threshold for each user. 4. Signature Update (inline and long term).
Computational Performance	Heuristic Aligning.	1. TMBO Approach. 2. Parallel computation.

In the following sections we detail the current implementation of DDSGA.

5.2 The Configuration Phase:

We briefly define the parameters that this phase computes for each user and that are used by the following phases. The detailed calculation of each parameter is described in the remainder of this section.

- **Optimal gap penalties:**

The optimal test gap penalty and the optimal signature gap penalty are paid when inserting a gap into the test sequence and the signature one respectively. While in Enhanced-SGA all the users share the same fixed penalties, DDSGA computes two distinct penalties for each user according to distinct user behaviours. In this way, DDSGA determines the smallest penalties corresponding to the maximum alignment score.

- **Mismatch Score:**

DDSGA evaluates the mismatch score through the restricted permutation scoring system and the free permutation one. These systems integrate the command grouping and binary scoring systems defined in [125].

- **Average optimal threshold :**

DDSGA computes a distinct threshold value for each user according to the

changes in the user behavior. The threshold is used in both the detection and update phases and its sensitivity affects the accuracy of both phases, as discussed in the average threshold module.

- **Maximum factor of test gaps (mftg):**

This parameter relates the largest number of gaps inserted into the user test sequences to the length of these sequences. DDSGA computes a distinct parameter for each user and updates it in the update phase. The detection phase uses the parameter to evaluate the maximum length of the overlapped signature subsequences in the TMBO.

The configuration phase is implemented using the following five modules:

5.2.1 DDSGA Initialization Module:

To provide an independent set of test and signature sequences for the configuration phase of each user, we split the user signatures into *nt* non-overlapped-blocks each of length *n* and use them as test sequences to the user. These sequences represent all given combinations of users signature sequences and all the modules in the configuration phase use them to compute the user alignment parameters. To define the signature sequences, we divide the user signature sequence into a set of overlapped groups of length $m = 2n$. In this way, the last *n* symbols of a block also appear as the first *n* of the next one. *ns*, the number of signature subsequences is equal to *nt-1* groups to consider all possible adjacent pairs of the signature sequences of size *n*. We have chosen a length *2n* to overlap the signature sequence because any particular alignment uses subsequences with a length that is, at most, *2n*. Any longer subsequence necessary scores poorly, because of the number of gaps to be inserted. In fact, since the scoring alignment depends upon the match between the test and the signature subsequences, the former should be shorter than the latter. As a consequence, the signature sequence for this phase consists of 2n command produced by overlapping the signature sequence. Hence, there are *nt-1* groups that are created as in Figure 5.2.

In the case of SEA, ns = 49 subsequences and a tested block consists of 100 commands because SEA marks each 100 command block as an intrusion or a non-intrusion. Since SEA does not supply any information on which commands in a block correspond to the intrusion, the correctness of larger or smaller blocks cannot be checked. The dynamic average threshold for both the detection and update phases is the average score of all the alignments

between each test sequence of length 100-command and the 49 overlapped 2n signature subsequences. A test sequence is not aligned with the signature subsequence that contains the test sequence itself because this returns a 100% alignment score. We will detail this step in the average threshold module.

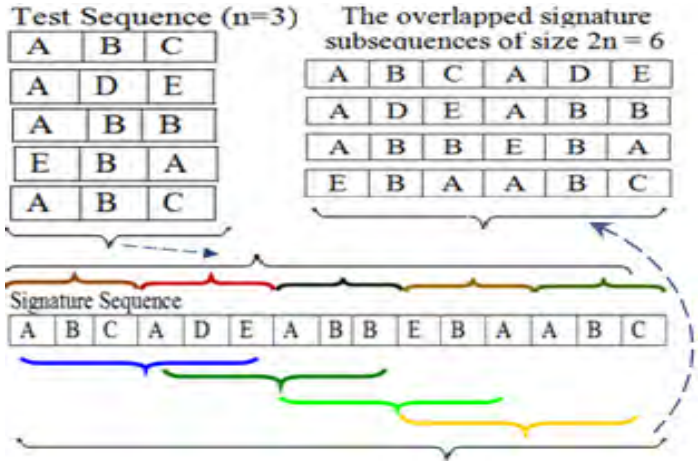


Figure 5.2: The non-overlapped test sequences and the overlapped signature subsequences

5.2.2 User's Lexicon Categorization Module:

This module builds a lexicon for each user, i.e. list of lexical patterns classified according to their functionality and that is used to tolerate changes in the low level representation of a pattern. In the SEA dataset, these patterns are UNIX commands. This module combines the user lexicon list and command grouping approach introduced in [125].

Table 5.2: User 1 Lexicon List

User'sLexiconList	Functional Group
cat	File system,Text processing
vi	Text processing
sh	Shell programming
env	User environment
kill	Process Management
reaper	Networking
mail	Internet Applications
hostname	System Information
nawk	Development , Shell Programming
getopt	Development , Shell Programming
grep	Searching , Text processing
find	Searching

Table 5.2 shows an example of a classification of user commands into several categories according to their functionalities e.g., since a user can use either *cat* or *vi* to write a file, the two commands can be aligned because both belong to the same group, “Text processing". In the same way, *grep* can be aligned with *find* because they both belong to "searching".

5.2.3 Scoring parameters module.

Starting from the test and signature subsequences of each user, this module returns three parameters: optimal test gap penalty, optimal signature gap penalty, and mismatch score. At first, the module inserts into the list *top_match_list* all the test sequences with the top match score. This list enables DDSGA to align the top match test sequences only rather than all the *nt* sequences. To build the *top_match_list*, we select the highest match scores for all the *nt* sequences. The match score *MS* of a test sequence is computed as in Equation 5.1. Then, the *top_match_list* sequences are aligned to the *ns* overlapped signature subsequences using any possible gap penalty, i.e. the test gap penalty ranges from 1 to *n*, while the signature gap penalty ranges from 1 to *n*. The mismatch score is 0 and the match score is +2.

$$MS = \sum_{i=1}^{n} \sum_{k=1}^{nt} Min(Noccur\_Itself(p_i), Noccur\_Seq_k(p_i)) \quad (5.1)$$

Where:

- *n* is the length of the test sequence.
- *nt* is number of test sequences.
- *Noccur_Itself(* p_i *)* is number of occurrence of pattern *i* in the current evaluated sequence.
- *Noccur_Seq* $_k$ (p_i) is number of occurrence of pattern *i* in test sequence *k*.

By computing each alignment separately, we produce several alignment scores for each combination of the scoring parameters and select as the optimal parameters those resulting in the maximum score. If several alignments result in this score, we select the one with the smallest penalties of inserting a gap into the signature subsequence and test one, respectively. To justify this solution, consider that the highest alignment score denotes a high level of alignment between the test and the signature subsequences and

SGA normally subtracts these penalties from the score. We have applied the previous steps to each user in SEA dataset to define the corresponding optimal penalties. Figure 5.3 and Table 5.3 show the penalties for user 1.

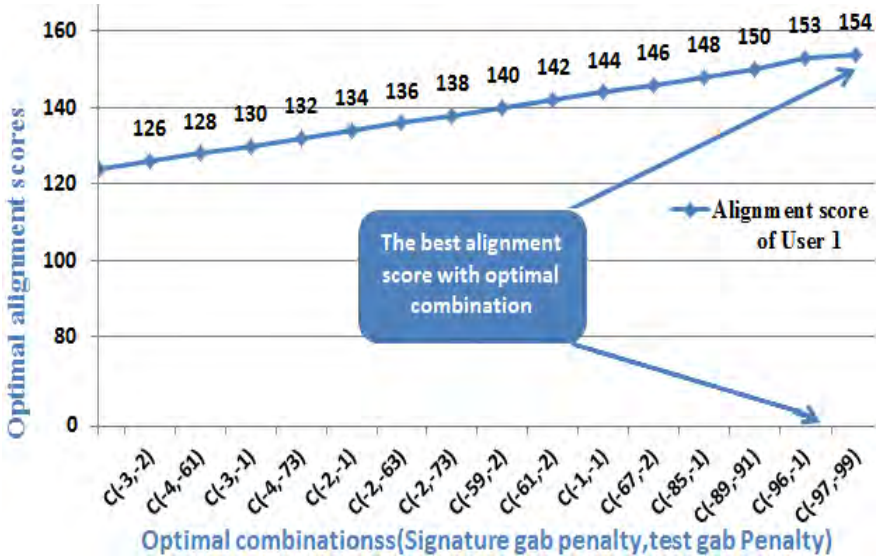


Figure 5.3: The best alignment score that corresponds to the optimal combinations of gap penalties for user 1 in SEA Dataset

Table 5.3: Example of the Top Match Scores of User 1

Max alignment score	Test sequence index (i)	Signature gap penalty	Test gap penalty	Optimal combination
154	2	99	95	-
154	5	100	95	-
154	8	97	100	-
154	**13**	**97**	**99**	**ok**

The Mismatch-Score Evaluation Algorithm:

This algorithm computes the mismatch score parameter using the restricted and the free permutation scoring systems as shown in Figures 5.4 and 5.5. Both systems integrate command grouping and binary scoring [125] and are based on distinct assumption about mutations in the audit data. Command grouping assumes that sequences of audit data mutate by replacing some original symbols with functionally similar ones. Command grouping assigns a static reward of +2 to exact matches and scores a

mismatch through the functional groups of the two commands. If a command in the signature aligns with a mismatched command in the test sequence but that belongs to the same group, the mismatch score is set to +1 rather than to -1. The assumption on mutation of the binary scoring system follows the results in [151] where mutation does not replace base symbols in the user lexicon. In fact, these symbols are a strong indicator of a legal use, but the original base symbols can be permuted in some fashion [125]. The binary scoring system rewards exact matches by adding +2 to the alignment score. A mismatch is scored to +1 if the mismatched command has previously occurred in the user lexicon and to -1 otherwise. Both scoring systems penalize the insertion of a gap into the signature sequence and into the test one by, respectively, -3 and -2.

(A) Restricted Permutation Scoring System:

It rewards a mismatched command in the test sequence if the two commands belong to the same group. These groups are manually created from a set of common UNIX commands in the signature sequences. Furthermore, the mismatched command of the test sequence should have previously occurred in the user lexicon. This tolerates various permutations of previously observed patterns without reducing the score significantly.

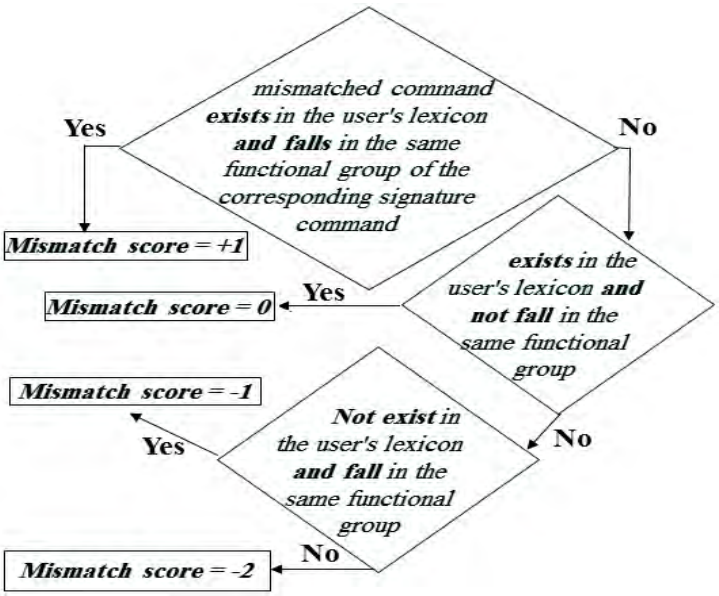


Figure 5.4: The restricted permutation scoring system.

(B) Free Permutation Scoring System:

This system is more tolerant than the previous one because it does not require that the mismatched commands belong to the same group and it even rewards a mismatched command provided that it belongs to the user lexicon. This tolerates a larger number of permutations of the signature patterns without reducing the score significantly.

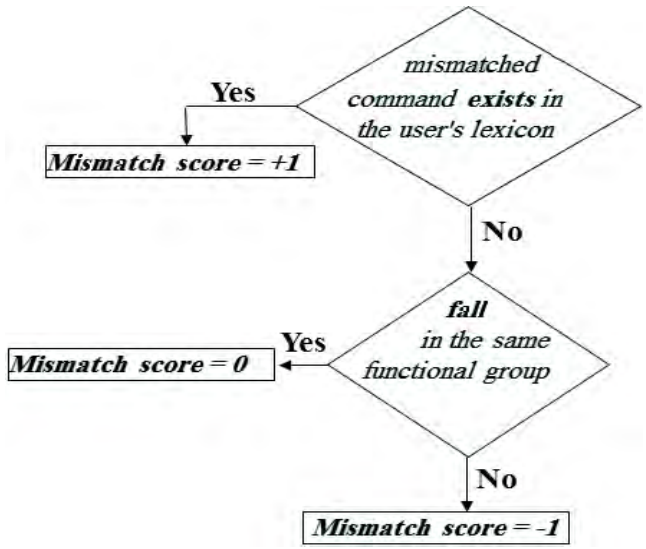


Figure 5.5: The Free Permutation Scoring System.

5.2.4 Average Threshold Module.

This module computes a dynamic average threshold for each user to be used in the detection phase and that may be updated in the update phase. In the detection phase, if the alignment score is lower than the threshold, then the behavior is classified as a masquerade attack. With respect to Enhanced-SGA, this module considers all the user data to improve the sensitivity of the threshold that, in turns, determines the one of the detection. The module uses the same test and signature subsequences of the initialization module and it can works with a test sequence of any length because, in practical deployment user test session can be of any length.

At first, the module builds a trace-matrix, to record all alignment details of the current user. We align each test sequence to all *ns* overlapped signature subsequences and run the module twice, one for each scoring systems to select the one to be used. The detection phase uses the output of

this module to compare the two scoring systems. These alignments use the optimal scoring parameters for the current user. The module applies Equation 5.2 to compute the average alignment of test sequence *i*, *avg_align_i*, and the sub_average score for all previous alignment scores, *score_align_i*, of test sequence *i*. In the equation, *max_score_align_i* is the largest alignment score resulting from the alignment of sequence *i* to all *ns* signature subsequences. Then, the module applies Equation 5.3 to compute the *detection_update_threshold*, the overall average of the *nt* sub average scores.

$$avg\_align\_i = ((\sum_{j=1}^{ns} score\_align\_i_j / ns) / max\_score\_align\_i) * 100 \quad (5.2)$$

$$detection\_update\_threshold = (\sum_{k=1}^{nt} avg\_align\_i_k) / nt \quad (5.3)$$

To compute the optimal alignment path and number of test gaps (*ntg*) to be inserted into test sequence *i*, we apply the trace backward algorithm (TBA) to trace back the transitions to derive each optimal score. An example is shown in Table 5.4.

Table 5.4: An Example for the Trace-Matrix

Test Seq. ID	Length of Test Seq. (*lts*)	Signature Subseq. ID	Optimal alignment	Number of Test Gaps (*ntg*)	Avg-align of test Seq. *i* (%)
1	100	1	27	65	
1	100	2	33	60	
⋮	⋮	⋮	⋮	⋮	
1	100	49	77	22	70.31
2	100	1	37	57	
2	100	2	53	43	
⋮	⋮	⋮	⋮	⋮	
2	100	49	67	31	69.72
⋮	⋮	⋮	⋮	⋮	⋮
50	100	1	122	9	
50	100	2	134	7	
⋮	⋮	⋮	⋮	⋮	
50	100	49	102	11	73.75

The Trace Backward Algorithm (TBA):

Any dynamic programming algorithm can select the optimal alignment path by tracing back through the optimal scores computed by SGA. We implemented the TBA to build the backward-transition-matrix, see Figure 5.6, in a way that helps in filling the trace-matrix in Table 5.4 as previously discussed. The alignment process also applies the TBA to extract the final alignment path. The TBA traces back the transition-matrix according to the labels that the alignment has inserted into this matrix. One of four labels may be inserted: "M" if a match has occurred, "!M" if a mismatch has occurred, "GS" or "GT" if a gap has been inserted into a signature subsequence or into the test one. Figure 5.6 outlines the TBA and shows the transition-matrix and the corresponding backward-transition-matrix to align the test sequence "AWGHE" to the signature sequence "PAWHE".

Test sequence: A W G H E
Signature sequence: A W - H E

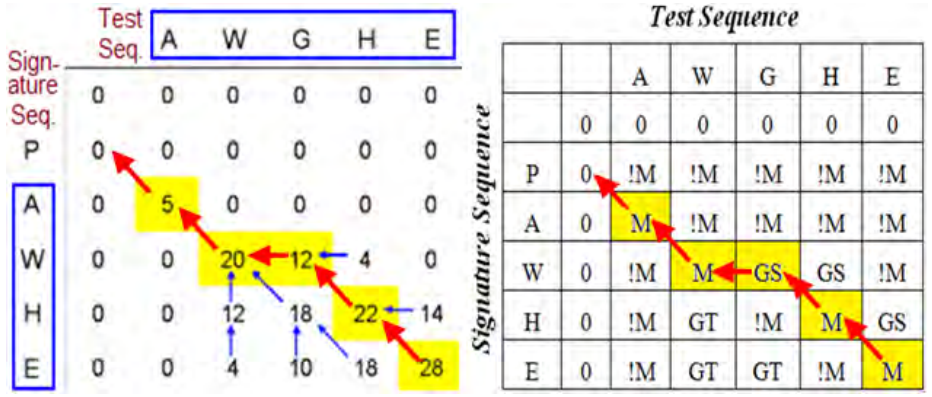


Figure 5.6: The Transition and Backward-Transition matrices respectively
The thick arrows show the optimal path that leads to the maximum alignment score, the thin ones show other proper paths that do not lead to the maximum alignment score.

5.2.5 Maximum Test Gap Module

We recall that the Enhanced-SGA Heuristic Aligning decomposes the signature subsequence into *2n* overlapped subsequences because if subsequences of length *n* are aligned, the maximum number of gaps that can be inserted into the test sequence is *n* for all users. By tracing the SGA

algorithm, we have noticed that the maximum number of gaps is much lower than *n*, the length of the test sequence, and it differs for each user according to the level of similarity among the subsequences in the user signature and to the length of the test sequence. Even if the test sequence is long enough, the number of gaps is at most half of the sequence length. This means that by partitioning the signature sequence into *2n* overlapped subsequences, the Maximum Test Gap module can divide it as in Equation 5.4 to compute *mftg*, the largest number of test gaps inserted into the user test sequences by the average threshold module.

The Computational Enhancement (CE) for the session alignment of a user can be computed according to Equation 5.5. We refer to the detection phase for an example that explains this section.

$$L = n + \left\lceil \left\{ \max_{k}^{nt} \left(\frac{ntg_k}{lts_k} \right) \right\} * n \right\rceil \tag{5.4}$$

Where:

- *ntg is number of test gaps inserted to each test sequence,*
- *lts is the length of the test sequence,*
- *nt is number of the test sequences of the user, fifty in case of SEA,*

$$CE = \left(n - \lceil mftg \rceil\right) * \left(100/(2*n)\right)\ \% \tag{5.5}$$

Where, Maximum Factor of Test Gaps inserted into all user test sequences

$$(mftg) = \left\{ \max_{k}^{nt} \left(\frac{ntg_k}{lts_k} \right) \right\}$$

5.3 The Detection Phase

We have run a complete alignment experiment based upon the test and signature blocks of the SEA dataset to evaluate the alignment parameters and the two scoring systems. The test blocks are the actual SEA testing data and they differ from those described in the initialization module. To simplify a

comparison with other approaches, we use the ROC curve and the Maxion-Townsend cost function [112] defined in Section 2.2.2.

Our experimentation focuses on the effects of the alignment parameters on the false positive and false negative rates and on the hit ratio. This experiment did not apply the maximum test gap module. False positives, false negatives and hits are computed for each user, transformed into the corresponding rates that are then summed and averaged over all 50 users. Equations 5.6, 5.7, and 5.8 show the DDSGA metrics.

$$TotalFalsePositive = \left(\left(\sum_{k=1}^{nu} fp_k / n_k \right) / nu \right) * 100 \quad (5.6)$$

Where:

- *fp = No. of false positive alarms,*
- *n = No. of non-intrusion command sequence blocks,*
- *nu = No. of users (50 in our case)*

$$TotalFalseNegative = \left(\left(\sum_{k=1}^{nui} fn_k / ni_k \right) / nui \right) * 100 \quad (5.7)$$

Where:

- *fn = No. of false negatives,*
- *ni = No. of intrusion command sequence blocks,*
- *nui = No. of users who have at least one intrusion block*

$$TotalHitRatio = 100 - TotalFalseNegative \quad (5.8)$$

To plot the ROC curve, the experiment with the traditional SGA algorithm have used distinct values of the alignment parameters to obtain different false positive rates in the x-axis and the corresponding hit ratios in y-axis. We also repeat the experiment with distinct values of some alignment parameters such as reward for matches and rewards or penalties for mismatches computed by the two scoring systems. Figure 5.7 shows the ROC curve for the two scoring systems, the Enhanced-SGA scoring systems, and other detection approaches, based upon the previous metrics.

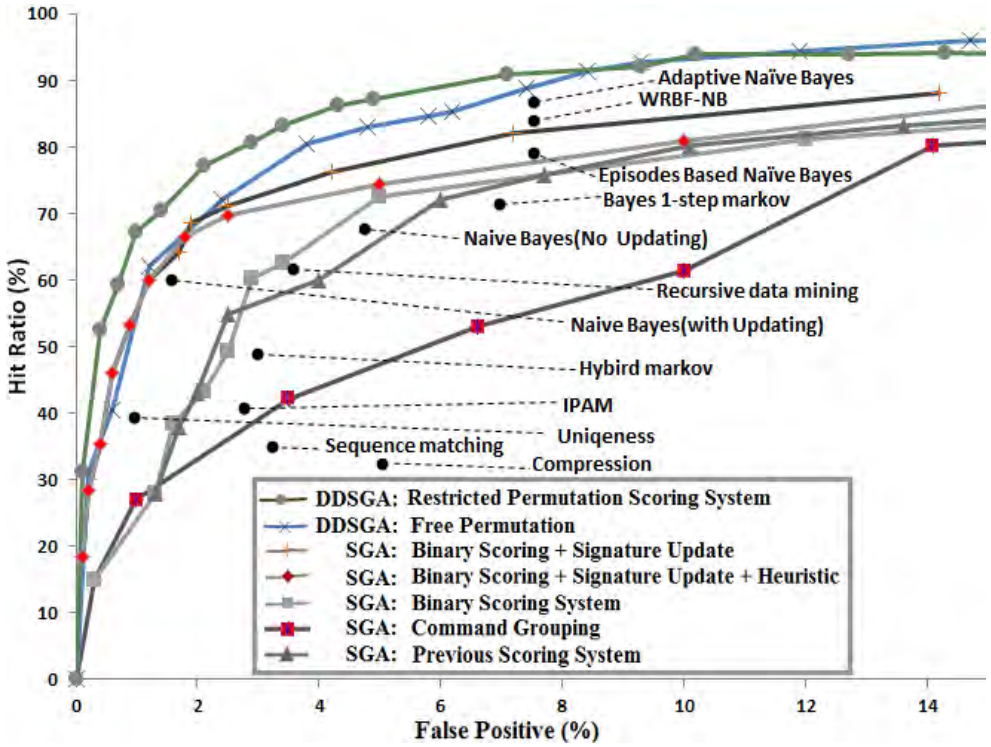


Figure 5.7: ROC curve for our two scoring systems, SGA ones, and other detection approaches

According to this figure, the restricted permutation system results in higher hit ratio with corresponding low false positive rates. As the false positives rate increases, the free permutation system achieves a higher hit ratio than the restricted one because it can tolerate a large number of mutations and deviations in user behaviours. According to this experiment, we have adopted the restricted permutation system as a suitable scoring system for all the phases of DDSGA.

Table 5.5 compares DDSGA using the restricted and the free permutation scoring systems against the current masquerade detection approaches sorted by Maxion-Townsend cost. The results for the various detection approaches, including all ROC curve values, are published in the references shown in Table 5.5 and are sorted by "Maxion Townsend cost" that is used to simplify the comparison by considering both the false positive and hit ratio. We re-implemented all Coull et all works in [124, 125] to compare it against DDSGA.

Table 5.5: A Comparison between Our Two Scoring Systems and the Current Detection Approaches

Approach Name	Hit Ratio %	False Positive %	Maxion T. Cost
DDSGA (Restricted Permutation)	**83.3**	**3.4**	**37.1**
DDSGA (Free Permutation)	**80.5**	**3.8**	**42.3**
SGA (Signature updating) [125]	68.6	1.9	42.8
SGA (Signature updating + Heuristic Aligning) [125]	66.5	1.8	44.3
Naïve Bayes (With Update) [112]	61.5	1.3	46.3
SGA (Binary Scoring)[125]	60.3	2.9	57.1
Adaptive Naïve Bayes [123]	87.8	7.7	58.4
Recursive Data Mining [120]	62.3	3.7	59.9
Naïve Bayes (No Update) [112]	66.2	4.6	61.4
WRBF-NB [122]	83.1	7.7	63.1
Episode based Naïve Bayes [121]	77.6	7.7	68.6
Uniqueness [138]	39.4	1.4	69.0
Hybrid Markov [116]	49.3	3.2	69.9
SGA (Previous Scoring) [124]	75.8	7.7	70.4
Bayes 1-Step Markov [115]	69.3	6.7	70.9
IPAM [117]	41.1	2.7	75.1
SGA (Command Grouping) [125]	42.2	3.5	78.8
Sequence Matching [118]	36.8	3.7	85.4
Compression [114]	34.2	5.0	95.8

The Computational Enhancement Modules:

The computational complexity of SGA is rather large. As an example, it requires about 500,000 operations to test one user session for masquerade attacks in the SEA dataset, because the length of the signature sequence is 5000 while that of the test sequence is 100. The resulting overhead is unacceptable in multiuser environments like cloud systems or in computationally limited devices like wireless systems. To reduce this overhead, we introduce two computational enhancements that concern, respectively, the Top-Matching Based Overlapping (TMBO) module and the

parallelized detection module that are executed in each alignment session. In the following, we outline in details these two modules.

5.3.1 The Top-Matching Based Overlapping Module

To align the session patterns to a set of overlapped subsequences of the user signatures, this module uses the restricted permutation scoring system, Maximum Factor of Test Gaps (mftg) in Equation 5.9, and the scoring parameters of each user. As explained in Section 5.1, the TMBO improves the Heuristic Aligning of the Enhanced-SGA. After splitting the signature sequence into a set of overlapped blocks of length L, see Equation 5.4, it chooses the subsequence with the highest match to be used in the alignment process. In the worst case, all overlapped subsequences should be aligned as they have the same highest match value. We have verified that on average, the number of alignments is rather smaller because of the variation between the overlapped signature subsequences. The evaluation of the proposed TMBO approach mainly depends upon two parameters: (a) Number of average alignments for the detection process, (b) The effect of the TMBO on false alarm rates and hit ratio.

To evaluate our approach with respect to (a) we show through an example how it reduces the alignment computations. As far as concerns (b), we use the ROC curve and Maxion-Townsend. To evaluate TMBO, Figure 5.8 shows the user session patterns to be aligned to the signature patterns. The configuration phase returns these values: signature_gap_penalty = 9, test_gap_penalty = 5, optimal_score_sys = "Restricted Permutation", detection_update_threshold = 82.2%, and mftg (maximum factor of test_gaps) = 33%.

The first step of TMBO computes the following length of the overlapped subsequences according to Equation 5.9:

$$L = \left(n + \lceil mftg \ * \ n \rceil\right) = \left(10 + \lceil 33/100 \ * \ 10 \rceil\right) = 10 + \lceil 3.3 \rceil = 10 + 4 = 14 \qquad (5.9)$$

With respect to the one in the initialization module, the current overlapping runs with length L rather than $2n$. Figure 5.8 shows the resulting overlapped signature subsequences of size L = 14. The last subsequence, i.e. subsequence 15, may be shorter than L, but it is still longer than the test sequence.

The second step computes the match corresponding to each subsequence as shown in the front of each subsequence in Figure 5.8. We only consider the matching of subsequence 1 because those for other subsequences are similar. If we denote by Match(X, s), the minimum between the occurrences of X in, respectively, a user session and in a subsequence, then Match (subseq.1) =

$\sum_{X \text{ in } \{A, B, C, D, E, F\}}$ (Match(X, subseq.1)) = $\sum$(Min(3,1), Min(2,2), Min(1,2), Min(1,2), Min(2,2), Min(1,2))= $\sum$(1,2,1,1,2,1)=8

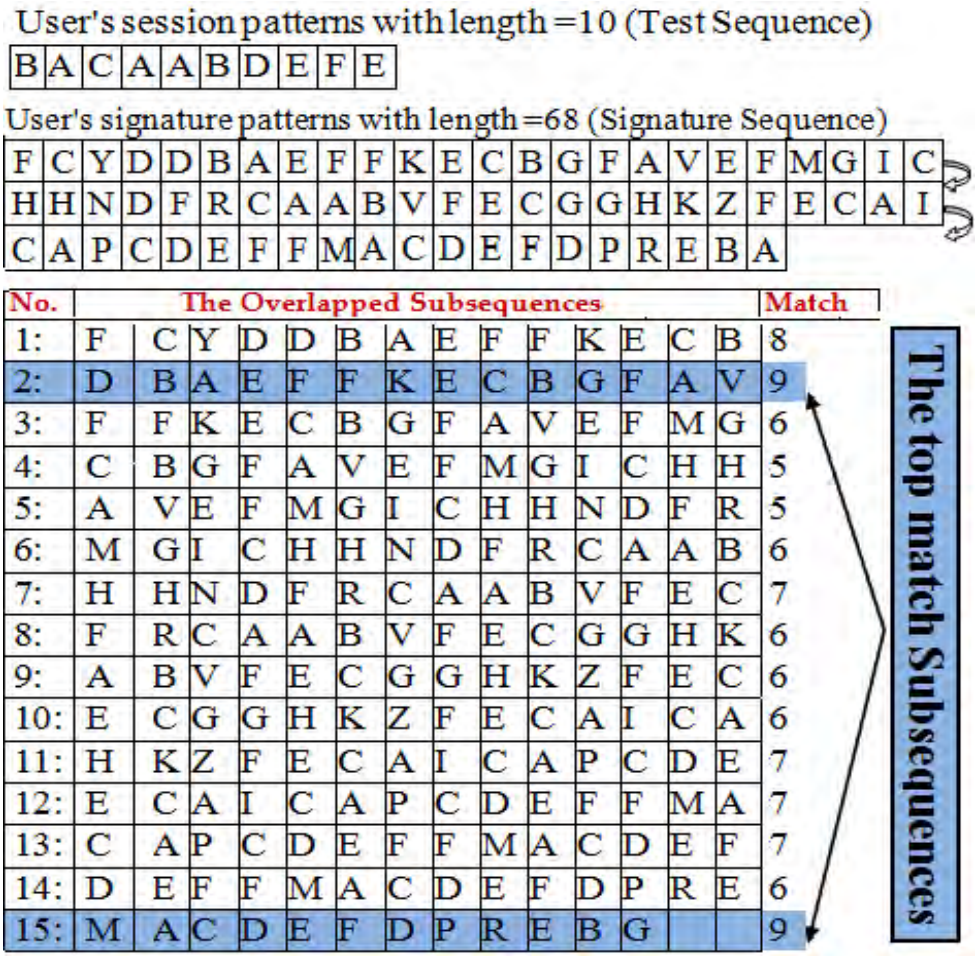


Figure 5.8: Overlapped Signature Subsequences of Size 14

The third step chooses the top match subsequences, e.g. subsequences 2 and 15 in the example, as the best signature subsequences to be aligned

against the test session patterns of the user. To evaluate the reduction of the workload due to TMBO, consider the Number of Asymptotic Computations (NAC) computed by Equation 5.10. In the previous example, TMBO reduces the number of alignments from 3000 to 280 with a saving of 90.66%.

$$NAC = Avg\_n\_align * Sig\_len * Test\_len \quad (5.10)$$

Where:

- *Avg_n_align is the average number of alignments required for one detection session over all existing users.*
- *Sig_len is the length of the overlapped signature subsequence.*
- *Test_len is the length of the test sequence).*

To determine if the session patterns of the current user contain a masquerader, the final step compares the highest scores of the previous two alignments against a *detection_update_threshold.* As explained in the update phase, if at least one of the previous eight alignments has a score larger than or equal to the *detection_update_threshold*, then an inline update process should be executed for the signature subsequence and the user lexicon.

The evaluation using the SEA dataset shows that TMBO reduces the maximum number of alignments from 49 to an average of 5.13 alignments per detection, a substantial improvement in detection scenarios. Table 5.6 shows the asymptotic computations for three detection approaches. The first is our TMBO without the inline update module. The second one is the Heuristic Aligning with signature update [125]. Finally, the third one is the traditional SGA algorithm without the Heuristic Aligning or update feature. The *NAC* per one detection session can be computed as in Equation 5.10. If we considered that each of the fifty users in SEA dataset has one active session in a multi users system, then *Total_NAC = NAC * 50.*

Table 5.6: TMBO approach in three detection approaches

Approach Name	Avg-n-align	NAC per 1 user	NAC per 50 user
DDSGA with L = 145.73	5. 13	5. 13 * 145.73 * 100 = 74759.49	74759.49 * 50 = 3737974.5
SGA with Signature length = 200	4.5	4.5 * 200 * 100 = 90000	90000 * 50 = 4500000
Traditional SGA with Signature length = 200	49	49 * 200 * 100 = 980000	980000 * 50 = 49000000

To evaluate false alarm rates and hit ratios, we have tested TMBO using the ROC curve and Maxion-Townsend score. Figure 5.9 and Table 5.7 show that TMBO has a lower impact on the overall accuracy than other approaches.

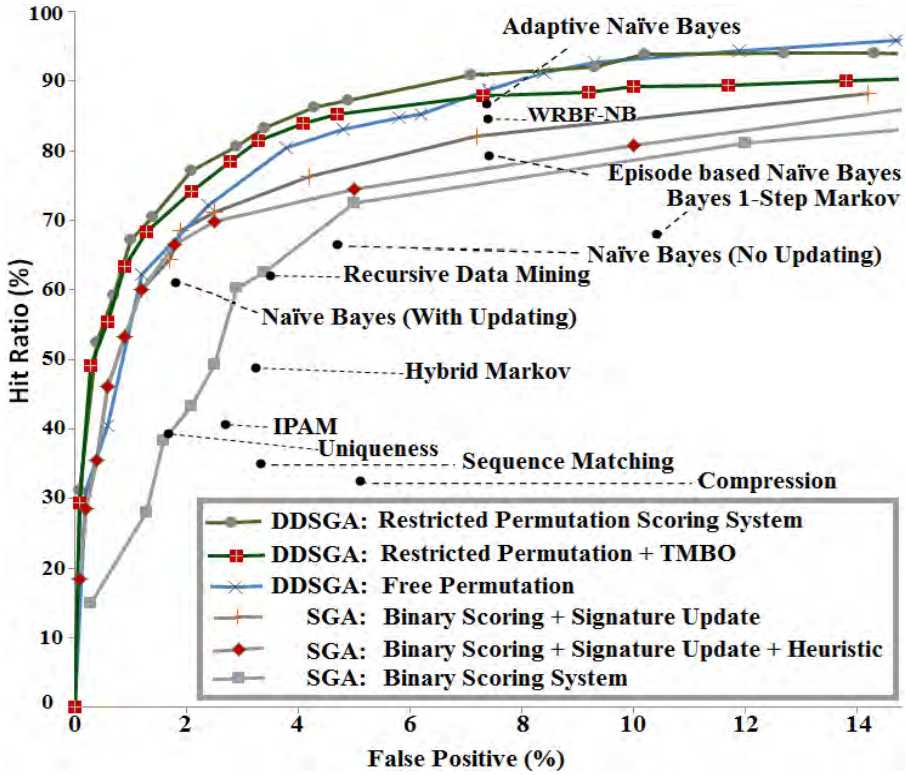


Figure 5.9: The impact of our TMBO approach on the system accuracy

Table 5.7: The Masquerade Detection Approaches against DDSGA with its Two Scoring Systems

Approach Name	Hit Ratio %	False Positive %	Maxion-T Cost
DDSGA **(Restricted Permutation, Without Updating)**	**83.3**	**3.4**	**37.1**
DDSGA **(Restricted Permutation, Without Updating)** + TMBO	**81.5**	**3.3**	**38.3**
DDSGA (Free Permutation)	80.5	3.8	42.3
SGA (Signature updating)	68.6	1.9	42.8
SGA (Signature updating) + Heuristic	66.5	1.8	44.3
Naïve Bayes (With Updating)	61.5	1.3	46.3
SGA (Binary Scoring, No Updating)	60.3	2.9	57.1

5.3.2 The parallelized Detection Module

Since TMBO partitions the user signature into a set of overlapped subsequences, we can parallelize the detection algorithm because it can align the commands in the user test session to each top match signature subsequence separately. In the example of Section 5.3.1, we can run in parallel the threads to align subsequences 2 and 15. If a thread returns a computed alignment score at least equal to *detection_update_threshold*, then it sends a "No Masquerader" message and then runs an inline update of both its signature subsequence and the lexicon of the current user. Instead, if the alignment score is lower than the *detection_update_threshold*, the thread raises a "Masquerader Detected" alert. Figure 5.10 shows the parallelized detection module processes.

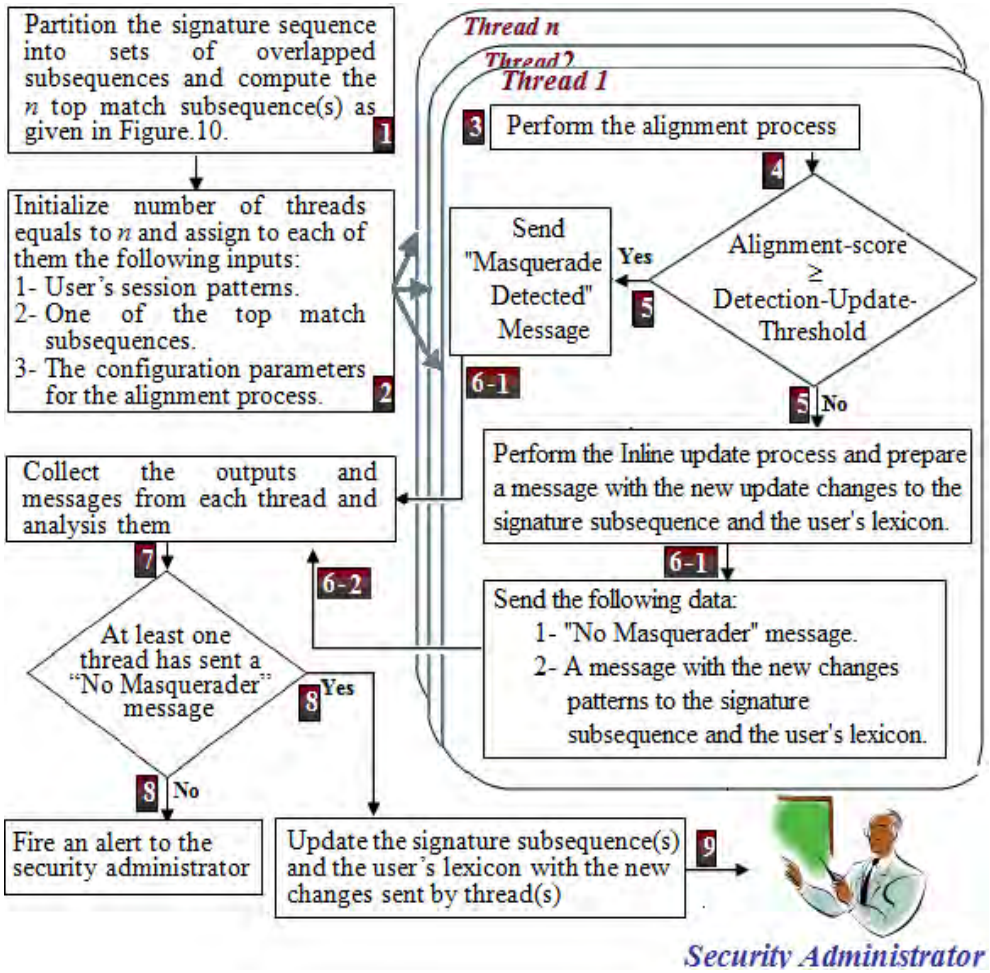


Figure 5.10: The processes of the parallelized detection module

We have run several experiments using the SEA dataset to evaluate how the parallel version of the module influences the overall performance of detection. The experiments have used an Intel Core 2 Duo Processor E4500 (2M Cache, 2.20 GHz) with 4 GB of memory and running Windows 7 SP1, Machine "A". Machine "B" is Intel Core i3-2330M (3M Cache, 2.30 GHz) with 6 GB of memory and running Windows 7 SP1.

The results show that there are two operational modes of the module: Full Parallelization Mode (FPM) and Semi Parallelization (SPM) one. The module selects the most appropriate one according to the capabilities of the underlying machine and to *n_aligns*, the number of alignments per detection. It selects the FPM mode if the machine capabilities match *n_aligns* so that the module achieves the best performance. This is the most common mode in our experiments. An example is the detection sessions of user 7 where *n_aligns* = 5, Figure 5.11 shows that if five thread are used then each thread runs a distinct alignment and this minimizes the detection time.

The SPM mode is selected if the machine capabilities do not match the required *n_aligns*. This results in a small performance degradation due to inactive threads. The SPM mode is rarely selected in our experiments because the fifty users of the SEA dataset results in a value of *Avg_n_align*, equals to 5.13. Hence, on average, the parallelized detection module uses 6 threads to run a detection session. A SPM mode example is the detection sessions of user 23 where *n_aligns* =9. Figure 5.12 shows that the shortest detection time is reached when running 6 threads in machine "A" and 8 threads in machine "B". In this case, 3 threads are idle in machine "A" and 1 in machine "B".

5.4 The Update Phase

The update of the user signature patterns is critical because any IDS should be automatically updated to the new legal behaviours of a user. This update is implemented by two modules: the inline update module and the long term update one.

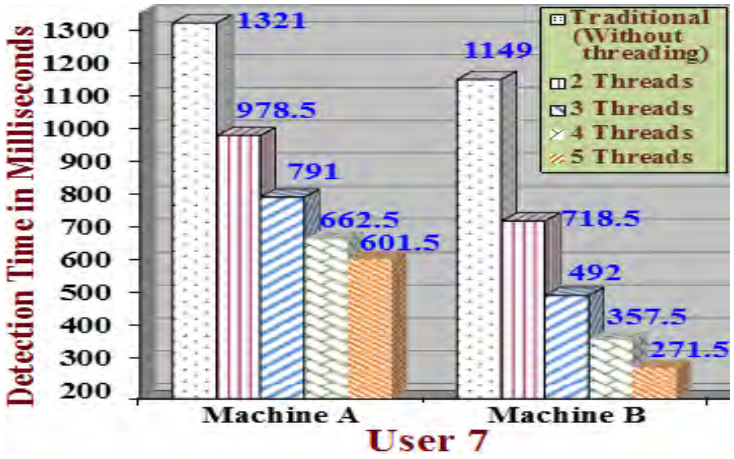


Figure 5.11: FPM and SPM modes for user 7 in Machines "A" and "B"

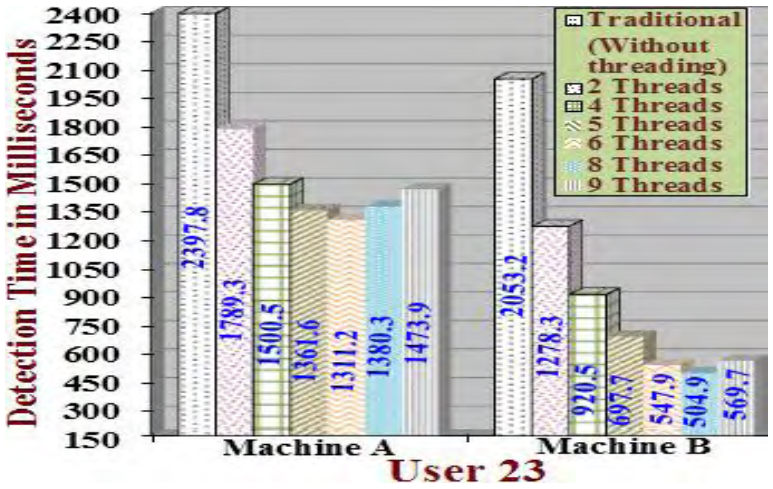


Figure 5.12: FPM and SPM modes for user 23 in Machines "A" and "B"

5.4.1 The inline update module

This module has two main tasks:

1) Finding areas in user signature subsequences to be updated and augmented with the new user behavior patterns.
2) Update the user lexicon by inserting new commands.

In the detection phase, after each alignment, each parallel thread may update both the user signature subsequences and the user lexicon. Three cases are possible in the TBA, see Figure 5.13:

a) The test sequence pattern matches the corresponding signature subsequence pattern,

b) A gap is inserted into either or both sequences
c) There is at least a mismatch between the patterns in the two sequences.

In case (a), no update is required because the alignment has properly used the symbol in the proper subsequence to find the optimal alignment. Also case (b) does not require an update because symbols that are aligned with gaps are not similar and should be neglected. In case (c), we consider all the mismatches within the current test sequence. Then, both the signature subsequence and the user lexicon are updated under two conditions.

The first one states that we can insert into the user signatures only those patterns that are free of masquerading records. This happens anytime the overall_alignment_score for the current test sequence is larger than or equal to the detection_update_threshold.

The second condition states that the current test pattern should have previously appeared in the user lexicon or belongs to the same functional group of the corresponding signature pattern.

The two conditions imply that the inline module updates the user lexicon with the new pattern if it does not belong to the lexicon. It also extends the pattern with the current signature subsequence and adds the resulting subsequence to the user signatures without changing the original one. In other words, if a pattern in the user lexicon or in the same functional group of its corresponding signature pattern has participated in a conserved alignment, then a new permutation of the behavior of the user has been created. For instance, if the alignment score of the test sequence in Figure 5.13 is larger than or equal to the detection_update_threshold, then the pattern 'E' at the end of this test sequence has a mismatch with 'A' at the end of the signature subsequence. If ‘E’ exists in the user lexicon or belongs to the same functional group of 'A', the signature subsequence is augmented so that the position with 'A' matches with both 'A' or 'E'. If ’E’ does not belong to the lexicon, it is also inserted into it. This simply embeds observed variations in the signature sequence without destroying any information it encodes. In this case, a new augmented subsequence (HEE) is inserted into the user signature subsequences. If only the first condition is satisfied, only the user lexicon is updated so that the following checks use the new pattern. In fact, it is highly likely that a pattern that has appeared within a conserved, high scoring alignment has been created by the legitimate user.

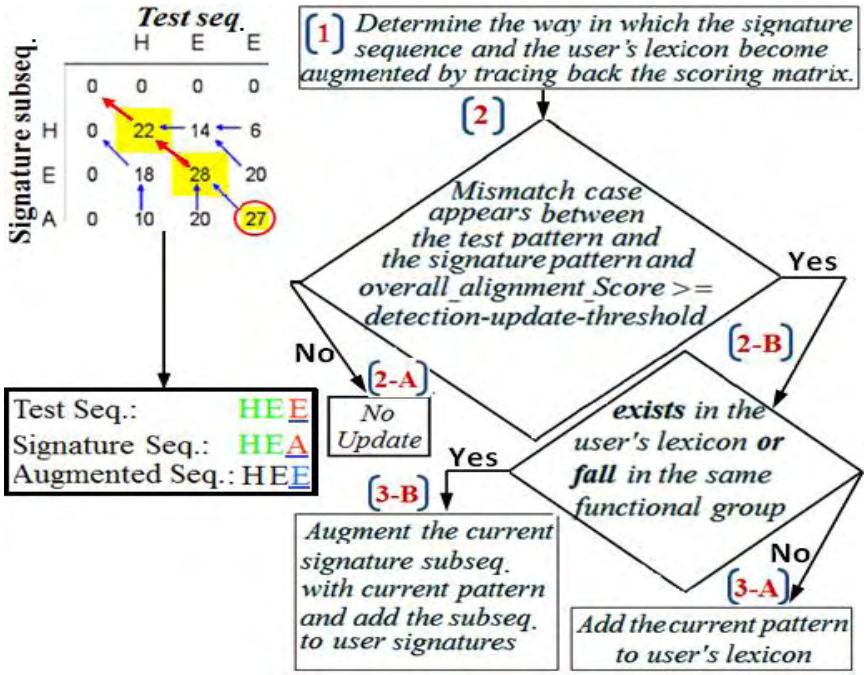


Figure 5.13: The inline update steps

Besides improving the computational performance of system update, the module also improves the signature update scheme of the Enhanced-SGA [125] as following:

1) It uses the parameters, the threshold, and the scoring system returned by the configuration phase.
2) It runs in parallel with the detection phase and starts as soon as the alignment score is computed. Instead, the signature update scheme runs independently after each detection process and it repeats the backward tracing step.
3) It improves flexibility in the signature update by considering any occurrence of commands permutations or functionality matching.

To evaluate how the inline update module reduces the false alarm rates and improves the hit ratio, we have used the ROC curve and the Maxion-Townsend score after applying the inline update module. Figure 5.14 and Table 5.8 show that the inline update module reduces the false alarm rates and increases the hit ratio. Therefore, it significantly improves the accuracy with respect to other approaches.

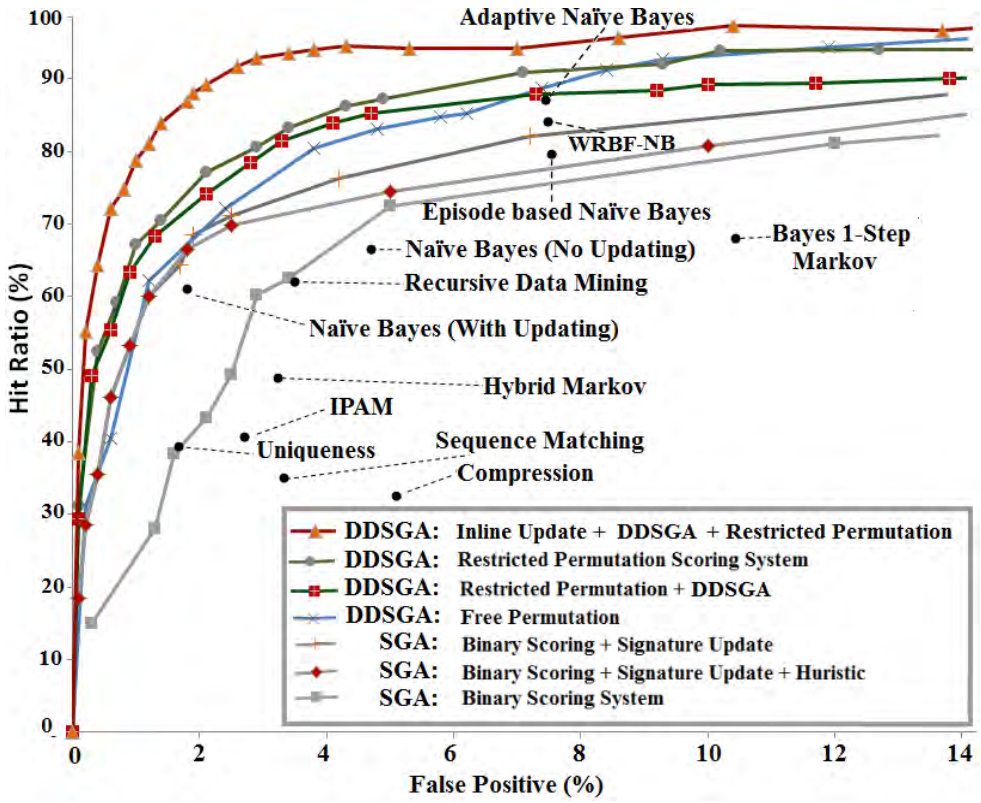


Figure 5.14: The impact of the inline update on the system accuracy

Table 5.8: Masquerade Detection Approaches against DDSGA with Its Inline Update Module.

Approach Name	Hit Ratio %	False Positive %	Maxion-T Cost
DDSGA **(Inline Update + TMBO + Restricted Permutation)**	**88.4**	**1.7**	**21.8**
DDSGA **(Restricted Permutation, Without Updating)**	**83.3**	**3.4**	**37.1**
DDSGA **(Restricted Permutation, Without Updating) + TMBO**	**81.5**	**3.3**	**38.3**
DDSGA (Free Permutation)	80.5	3.8	42.3
SGA (Signature updating)	68.6	1.9	42.8
SGA (Signature updating) + Heuristic	66.5	1.8	44.3
Naïve Bayes (With Updating)	61.5	1.3	46.3
SGA (Binary Scoring, No Updating)	60.3	2.9	57.1

5.4.2 The long term update module

This module reconfigures the system parameters through the outputs of the inline update module. There are three strategies to run the module: periodic, idle time, and threshold. The proper one is selected according to the characteristic and requirements of the monitored system.

The periodic strategy runs the reconfiguration step with a fixed frequency, i.e. 3 days or 1 week. To reduce the overhead, the idle time strategy runs the reconfiguration step anytime the system is idle. This solution is appropriate in highly overloaded systems that require an efficient use of the network and computational resources. The threshold strategy runs the reconfiguration step as soon as the number of test patterns inserted into the signature sequences reaches a threshold that is distinct for each user and frequently updated. This approach is highly efficient because it runs the module only if the signature sequence is changed.

The DDSGA webpage [152] describes further details on, among others, analysis results for each user in SEA dataset, output charts, and pseudo algorithms. Chapter 6 and 7 describe an implementation of DDSGA to detect masquerade attacks in the cloud systems using different auditing profiles.

Chapter 6

Detecting Masqueraders through System Calls and NetFlow Data

Masquerade attacks become a big challenge in cloud systems because, the massive amount of system resources, alternative deployment models, and the distribution of user audits and activities across several VMs with distinct environments, strongly increase the complexity of their detection. Most of the current approaches to detect these attacks suffer from severe limitations when applied to cloud systems. As an example, they analyze user behaviors just in one environment without correlating all activities of the same user in host and network environments. Furthermore, they are not applicable to alternative deployment models such as private, hybrid, and public clouds. This chapter discusses three detection strategies [143]. The first strategy analyzes sequences of correlated system calls audits from the operating systems of the VMs, the second considers data from the network environment and the last one integrates the first two strategies. To simplify the testing and the evaluation of the three strategies, we used our CIDD dataset introduced in Chapter 4 as a source for cloud audits data. Finally, the chapter details our experiments to determine the optimal parameters for the various strategies and describe the experiments we have implemented to evaluate the computational performance and the detection accuracy of these strategies.

6.1 Overview

All the three detection methods described in the following analyze the audit data through DDSGA. The first method applies DDSGA to sequences of user system calls from the host environment. The second one uses NetFlow audits collected from the network environment. The third method integrates the outputs of the first two methods through a neural network and uses statistical information associated with active session e.g., the login period, user's source IP address, and login failure actions. To tune and evaluate the three methods we have used the system calls and NetFlow data in the CIDD dataset. DDSGA is applied to the user audits in CIDD in a fully functional cloud system according to the distributed architecture of CIDD.

Any approach that analyses the system calls result in a high degree of information assurance because these calls reflect all system activities and their monitoring is tightly integrated with the OS. This makes the monitoring process more tamper-resistant to malicious updates of the information of interest. As a counterpart, an analysis that considers all the system call categories may result in a slow detection process and a high false alarm rates with a low hit ratio. This is the reason why our analysis extracts specific features from the system calls through our "Behaviours Triangle Model" that is focused on calls related to file access and to process activities because they are essential and unavoidable for any user. This simplifies the labeling of abnormal behavior because these calls reflect any regularity of the user behaviour in audit data more than other calls.

In the network environment, a NetFlow profile is built not for each user but for each source IP address. This profile is based upon sequences of network actions captured by sniffing tools. For each action, we record the sequence of destination IPs that have been accessed successfully together with the protocols used.

The three alternative implementation models described in Chapter 3, Audit Exchange, Independent, and Centralized-Backup, are evaluated using their corresponding CIDS and CIDS-VERT frameworks. These models help in analyzing the behavior of the same user in distinct cloud nodes. Furthermore, they improve the functionality of their frameworks to efficiently cover attacks in distinct cloud deployment models.

6.2. Detecting Masquerades in Host Environment

This section describes masquerade detection based on the anomalous analysis of system calls sequences and evaluates this strategy through the UNIX audits of the CIDD dataset.

6.2.1 System Calls Feature Extraction

Several monitoring and audit strategies of OS activities are focused on system calls. These calls can be roughly grouped into five major categories: process control, file management, device management, information maintenance, and communication [153].

While current IDSs analyze calls in all categories, our solution only monitors two categories: file access and process activities. To explain this

decision, we briefly discuss the disadvantage of considering all categories of system calls. First, an analysis that considers any call with its full parameters and features often results in a slow detection process that produces very high false alarm rates with low hit ratio. This is due to the large number of system calls parameters and the large number of possible permutations of the calls. Another problem is that the training patterns for most calls are specific to the program versions so that the accuracy of detection changes anytime the version changes. Another important reason is that the basic premise for anomaly detection is the intrinsic regularity in the audit data that is consistent with the normal behavior and distinct from the abnormal one. In other words, the more regular the data, the better the performance of anomaly detection [154]. We have focused on file access and process execution because they are essential and unavoidable for any user and can strongly reflect the user behaviour with more intrinsic regularity than other activities. To analyze file access and process execution, our Behaviours Triangle Model, builds a profile of system activity of each user and reflects user behaviors in terms of three relationships, see Figure 6.1:

(a) User access a file,
(b) Process accesses a file,
(c) User invokes a process.

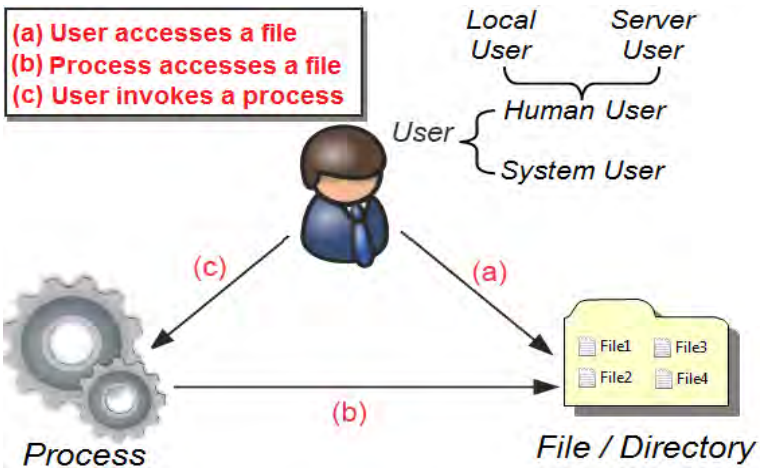


Figure 6.1: The Behaviours Triangle Model

The Behaviours Triangle Model classifies users into human and system users. In turns, it partitions human users into local and server users according to their nature. Local user activities are more complex and dynamic than

those of server users and include database administration, word processing, web browsing, and miscellaneous activities such as command-prompt and window manager interaction. The activities of server users are more related to email services, file transfer, and web browsing. Unlike human users, system users are dedicated to a few tasks and have specific privileges and are likely to behave statically. These limited activities and privileges simplify the detection of attempts to masquerade as a system user. DDSGA detects these attempts through a lexical list with common files and processes accessed or invoked by system users. Any pattern outside this list is a strong indication of a masquerader. We separate human user activities from system user processes and then check each user separately.

In multi user systems like cloud systems, activities of distinct users are recognized in traces through the user ID paired with each system call. Using these IDs, we can easily define the activities of each human user. Instead, system users activities may interact with local and server users, but they usually do not refer to a specific user. Hence, we can detect a masquerader that is misusing the system user but without any information about the legal user that originated the misuse. To this purpose a further detection process should be run for all human users.

Figure 6.1 shows three training sessions for the three kinds of users, according to the features extracted from the CIDD dataset. The uucp system user can access to the /uccp directory, whereas the root user can access the /ufs and /etc files. As soon as a masquerader compromises them, the program tries to access files in other system directories such as /lib or /user. By exploiting the interactions between system users and human processes, a masquerader can escalate his/her access privileges to acquire some human user privileges. Both users uccp and root execute the last two masquerades in the sessions in Figure 6.2 but it is not known if they affected one or more of the human users, e.g., users 2060 or 2140. DDSGA detects these masquerades by checking the system users' profiles. If the detection rate is smaller than the detection threshold, DDSGA checks all the human users to discover if any of them were impersonated by that masquerader to use the privileges of the system users. The next section details how DDSGA detects masquerade attacks in UNIX system call traces.

Local User

= = = = = =

Session Head: Session-ID, User-ID, SourceIP, Period, LoginFailure?, Real-Masquerade?

368-VM2 , 2140, 194.007.248.153, Evening, 0, 1

Session Contents: (Path, Return-Value)

(/export/home/janes/.hushlogin, 0), (/opt/local/bin/tcsh, 0), (/usr/lib/fs/ufs/quota, 0), (/usr/bin/cat, 0), (/usr/bin/rm, 1), (/usr/bin/vi, 0), (/usr/bin/su, 1), (/usr/ucb/whoami, 0), (/usr/bin/hostname, 0), (/opt/local/lib/solaris/specs, 1) ……

Server user

= = = = = =

102-VM4, 2060, 172.016.112.207, Afternoon, 0, 0

(/opt/local/bin/tcsh, 0), (/usr/lib/fs/ufs/quota, 0), (/usr/bin/cat, 0, 0), (/var/mail/lucyj, 0), (/usr/bin/ftp, 0), (/usr/bin/lynx, 0), (/usr/lib/sendmail, 0) ……..

System Users

= = = = = = = =

68-VM1, uucp, 127.0.0.1, Afternoon, 0, 1

(/usr/bin/sh, 0), (/usr/bin/date, 0), (/usr/lib/uucp/uusched, 0), (/usr/lib/uuxqt, 1), (/usr/bin/touch, 1), (/usr/bin/date, 0), (/usr/lib/uucp/uusched, 0), (/usr/lib/uucp/uuxqt, 0) ……

73-VM1, root, 127.0.0.1, Morning, 0, 1

(/usr/lib/fs/ufs/ufsdump, 0), (/etc/dumpdates, 0), (/usr/ucb/whoami, 1)……..

Figure 6.2: Three training sessions with the extracted features for three types of users

Figures 6.1 and 6.2 show that we build a user profile based on the sequence of patterns of the files the user has accessed and of the process being invoked. The arcs of the triangle define three relationships:

(a) A user accesses a file: it includes the file access patterns that correspond to a given task. These access patterns represent a profile of the normal user behavior. Therefore, any deviation in the current pattern with respect to this profile will be considered as an anomalous.

(b) A process accesses a file: this relationship defines how the process accesses a file, a masquerader may use the user process privileges to access important files. The analysis of the sequences of the files that the process accesses may result in a highly accurate detection with respect to the files the user accesses, because this relationship involves a fixed and well defined list of the files each process can access.

(c) A user invokes a process: while the previous analyses of system calls [sys-call-model, Behav-Monitor, and Detect-Buffer] consider the list of all the processes that each program forks or executes, our analysis focuses on the sequence of processes the user has invoked and the

programs being executed. The main idea is that each user has a characteristic working set of accessed files and a list of favorite programs. We record the sequences of these programs and consider as anomalous the case where either the process does not follow the normal sequence or a test process or program does not appear in these sequences. The analysis neglects forked processes, because they are highly predictable and the HIDS component can easily detect a process that should not have been forked.

Figures 6.3, 6.4, and 6.5 show, respectively, the distributions of the access patterns and of the executed programs for the three user categories.

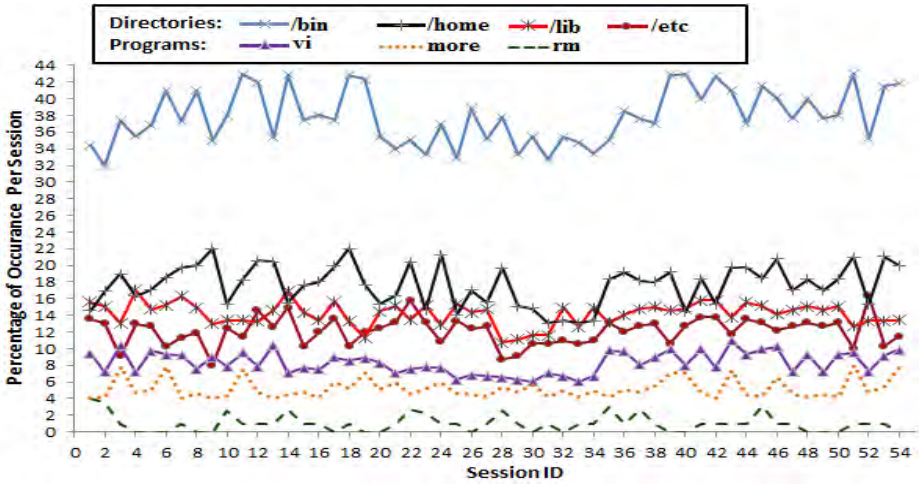


Figure 6.3: The Local User with ID "2140" in CIDD.

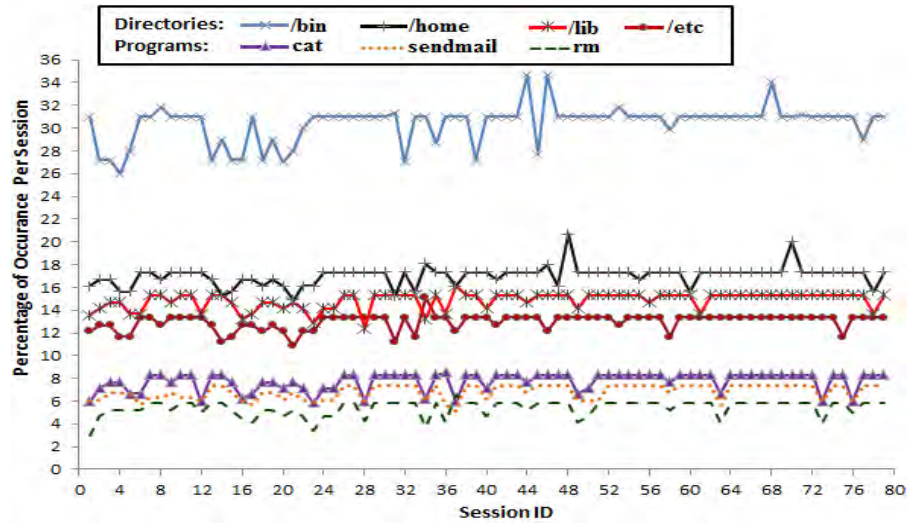


Figure 6.4: The Server User with ID "2060" in CIDD.

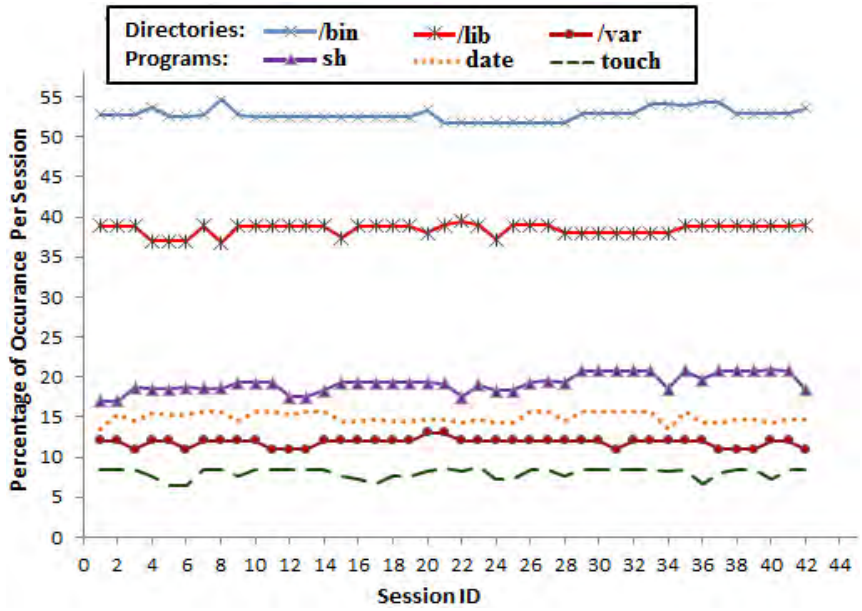


Figure 6.5: The System User with ID "UUCP" in CIDD.

The previous figures show that both the behaviours of server and system users are highly predictable. Hence, any deviation in these behaviours may be more accurately detected than those for local users. We also notice that the programs that are executed can be more reliably predicted and the corresponding access patterns improve the detection process.

6.2.2 Applying DDSGA to Correlated System Calls

DDSGA [DDSGA] can work efficiently with any sequence of patterns regardless of the running environment. To detect masquerades in UNIX audits, DDSGA computes the best alignment score by aligning the active session sequence e.g., the access patterns and executed processes as in Figure 6.2, to the previous sequences of the same user. The implementation of DDSGA is updated because originally it was applied to the simulated SEA dataset [136] with overlapped sessions of fixed length. Instead, the CIDD dataset is distributed among the cloud VMs and has non-overlapped non-fixed length real time sessions. To this purpose, we have to discover the best Sliding Window Size (SWS) for each test session and to build a reasonable scoring system according to the extracted features of UNIX audits. Both the detection and update phases of DDSGA do not change if CIDD is considered, because they use the new parameters returned by the configuration phase. In the detection phase, we test three correlation models,

Independent, Audit Exchange and Centralized-Backup to correlate the behaviour of the same user in distinct cloud nodes. We detail these issues and discuss the effect of correlation in the following subsections.

6.2.2.1 Choosing the Best Sliding Window Size (SWS)

The dynamic or Sliding Window Size (SWS) determines the length of the test sequence in the active session or, in other words, it determines when the detection phase should start the detection process. The SWS affects this process and helps in improving the system call modeling methods. Previous work [154] has chosen the optimal size with reference to an information theoretic framework that applies one of two approaches based on the training data. The first one uses entropy modeling and it considers data regularity. The second approach fully exploits the context dependency of the optimal size and estimates it according to the specific system calls in the training subsequences. The latter approach models the complete set of system call traces and it is not suitable in our framework that is focused on a subset of call traces (access patterns and the invoked processes). Furthermore, it uses the sparse Markov transducers [155] while DDSGA uses a more flexible alignment technique. We have estimated the SWS factor using four approaches:

(a) Minimum Conditional Entropy (MCE),
(b) Test Session Length (TSL),
(c) Test Session Length with Sensitive Action (TSLSA),
(d) Average Signature Session Length (ASSL).

We have evaluated each approach through two main measures, the detection accuracy using the Receiver Operator Characteristic (ROC) curves [113] described in Chapter 2 and the masquerader live time length. After highlighting the four approaches, we detail their evaluation.

(a) The Minimum Conditional Entropy (MCE):

Conditional entropy, see Equation 6.1, measures the regularity of the training data for each user using alternative *SWS* values and choose the one with the lowest entropy that also corresponds to the most regular data. The definition of conditional entropy is recalled in Equation 6.1.

$$H(X|Y) = -\sum_{x,y \in X,Y} P(x,y)\ log_2\ P(x|y) \qquad (6.1)$$

Where *P(x, y)* is the joint probability of *x* and *y*, and *P(x|y)* is the conditional probability of *x* given y.

To apply the entropy to our model, we introduce the following definitions:

- *X= S:* the set of system call patterns (access patterns and invoked processes).
- *Y=* S_{SWS-1} : the set of system calls patterns sequences of length *SWS-1*.
- S_{sws}: A sequence of length *SWS.*
- *A:* the set of all sequences in the training data.
- $N(S_{sws})$: The number of times the sequence S_{sws} appears in *A.*
- *N(A)*: The total number of sequences in the training data.

If we define the joint probability *P(x,y)* as following:

$$P(x,y) = P(s_{SWS}) = \frac{N(s_{SWS})}{N(A)}$$

The conditional entropy in Equation 6.1 for a window size *SWS* is:

$$H_{SWS}(X|Y) = -\sum_{x \in X, y \in Y} \frac{N(s_{SWS})}{N(A)} \log_2 P(x|y) \quad (6.2)$$

The conditional probability *P(x|y)* is the prediction of this entropy model and it means that the probability of system call *'x'* at a position (*SWS)* in the training sequences is estimated from *y*, the previous (*SWS-1)* system calls. To compute the prediction for each user training data, we consider each (*SWS-1*) sequence of system call patterns in the training data of user *U* and keep counts of the following system calls.

Then, *P(x|y)* the prediction for system call *'x'* given a sequence *'y'* that includes the (*SWS-1*) preceding system calls is simply *p/t* where, *'p' is* the count of system call *'x'* in the sequences such as *'y'* and *t* is the total count of system call *'x'* in the sequences that consider even the calls after the *SWS-1* position.

If a sequence S_{sws} does not occur in the system call patterns sequences, $P(S_{sws})=0$. Therefore, we can set as in Equation 6.3 the conditional entropy of Equation 6.2 for a window size *SWS* to denote that at least one S_{sws} sequence occurs in *'A'*, the set of all sequences in the training data.

$$H_{SWS}(X|Y) = -\sum_{s_{SWS} \in A} \frac{1}{N(A)} \log_2 P(x|y) \qquad (6.3)$$

Equation 6.3 computes the conditional entropy $H_{SWS}(x|y)$ for the system call patterns sequences of each user using a window size *SWS*. The most suitable windows size for each user is the one that corresponds to the minimum entropy and the more regular data.

To compute the conditional entropy for each user data, we use the cross validation in [154] for the training sequences of each user extracted from the CIDD dataset. We train the prediction models with one part of the training data and apply Equation 6.3 to compute the entropy over the second part of this data. Then, we repeat the computation after swapping the two parts. The total entropy is the sum of both entropies. Figure 6.6 shows the conditional entropy for three users, each of a distinct kind.

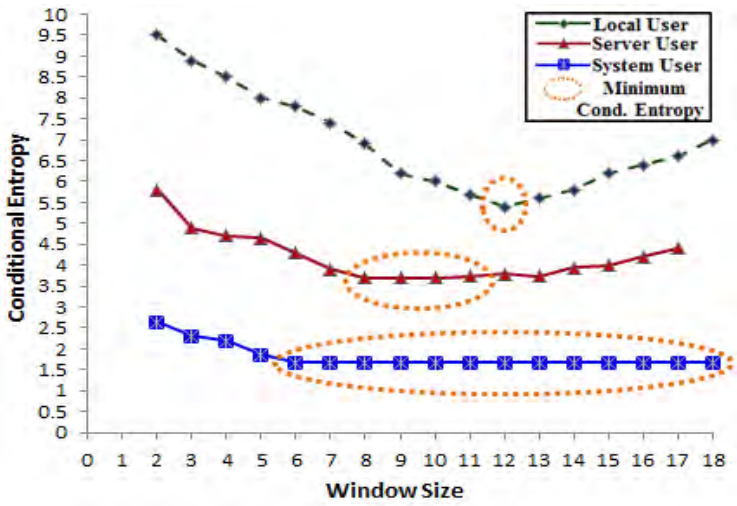


Figure 6.6: The conditional entropy under different *SWS* values for local user "2140", server user "2060", and system user "UUCP".

Notice that the curves for server and system users do not have a specific minimum due to the high data regularity for these users. We have noticed that the lower the entropy, the more regular the data and the better the performance of detection. The accuracy and masquerader live time resulting when adopting the conditional entropy model are acceptable for server and system users because of their highly regular data. This solution does not achieve the same performance for local users whose data are less regular.

(b) Test Session Length (TSL):

It sets SWS to the length of the active test session. This approach achieves the best accuracy among all the approaches because the resulting size makes it possible to compare the active test session against the previous training sequences. However, the masquerader live time is longer than in the other approaches.

(c) Test Session Length with Sensitive Action (TSLSA):

The *SWS* is set to the length of the active test session but the detection process can start at any time a sensitive action occurs. Sensitive actions are a set of sensitive file access patterns or of invocation to programs predefined in the DDSGA database. An attacker exploits these patterns or these invocations to misuse cloud resources. Table 6.1 shows some sensitive files and programs in the DDSGA database. The approach achieves suitable detection accuracy and a shortest masquerader live time for all user categories except the system users. This may due to the fact that usually these users access sensitive files and execute sensitive commands to support operating system functionalities. As an example, when a user logs in, the operating system checks the */etc/passwd* file.

Table 6.1: Examples for sensitive files and programs

File/Program	Pattern	Pattern task
File	*/etc/passwd*	Records users encrypted password
File	*/usr/adm/saveacct*	Records accounting information
File	*/usr/adm/wtmp*	Records all logins and logouts
File	*/etc/hosts*	List of IP hosts and host names
Program	*/bin/passwd*	Changes user password
Program	*/bin/yppasswd*	Changes NIS password
Program	*/etc/ttymon*	Monitors terminal ports
Program	*/sbin/*fdisk	Formats hard disk
Program	*/bin/chmod*	Changes file permissions

(d) Average Signature Session Length (ASSL):

The *SWS* is set to the average length of training sessions or to the length of the current session if it is shorter than the average one. ASSL increases both the accuracy and the masquerader live time for users with close training session lengths.

We evaluate and compare the detection accuracy of each of the four approaches for local, server, and system users through the ROC curves in Figures 6.7, 6.8, and 6.9 respectively. Each curve graphs the false positive rate versus the detection rate. We have also built the chart of the masquerader live time that shows some masqueraded sessions from the training data for the three users. To get distinct false positive rates in the ROC curve, we have changed some of the DDSGA parameters such as the detection threshold and the scoring parameters.

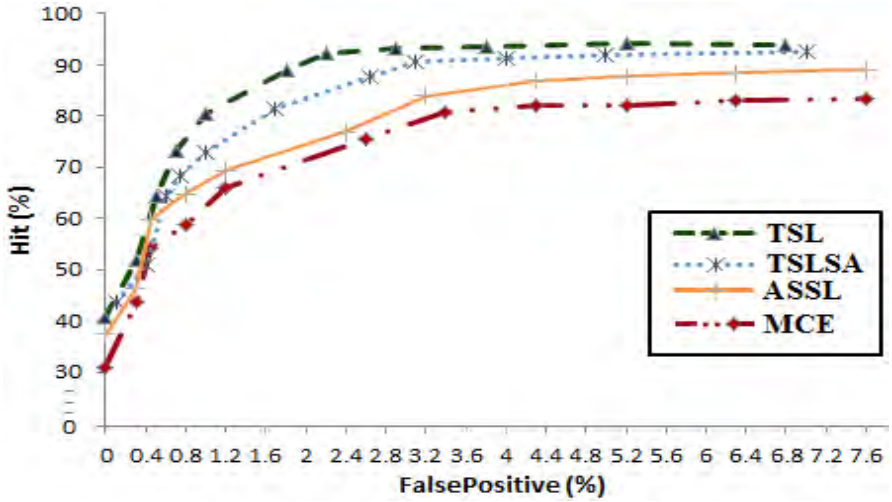


Figure 6.7: The ROC for the three sliding window selection methods for local user "2140" in CIDD.

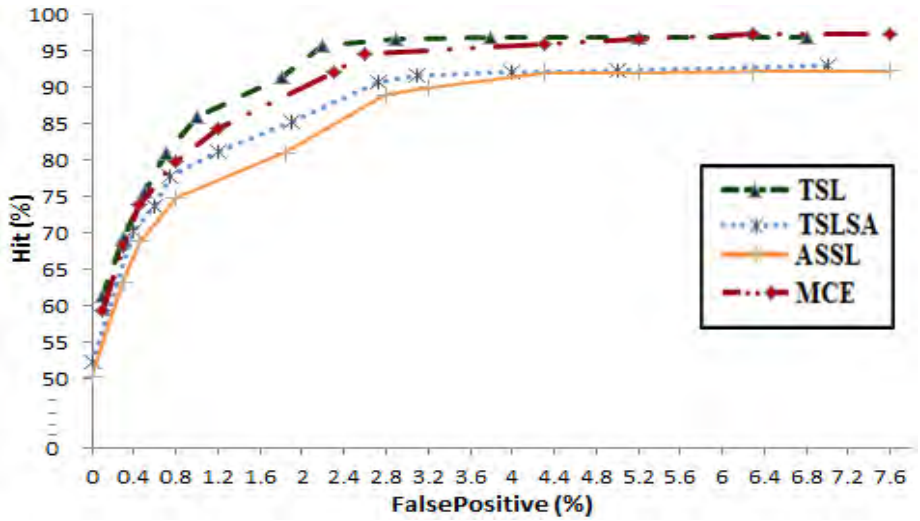


Figure 6.8: The ROC for the three sliding window selection methods for server user "2060" in CIDD.

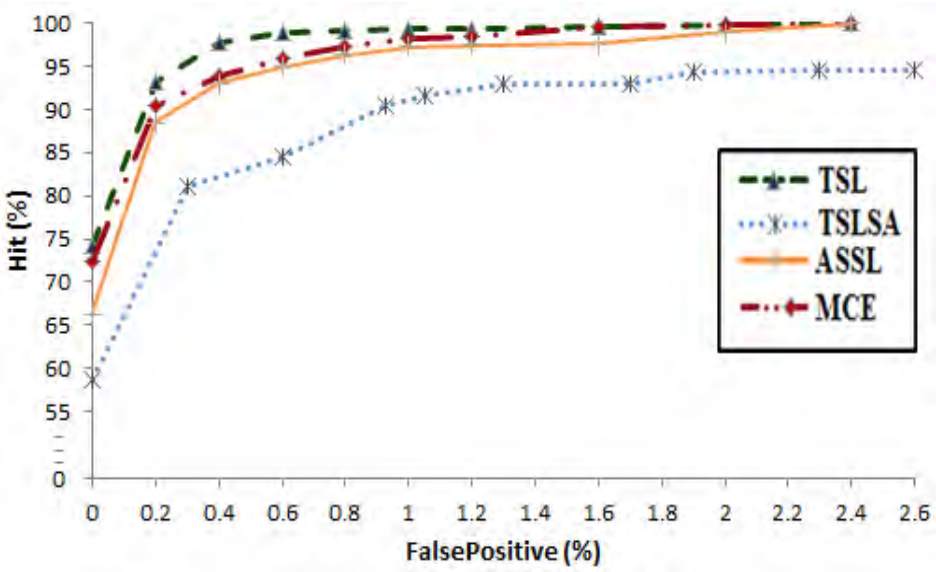


Figure 6.9: The ROC curve for the three SWS approaches for system user "UUCP" in CIDD.

We have applied DDSGA using the Centralized-Backup model with the lexical and return checks as detailed in the next section. Other correlation models result in the same conclusions. In the ROC curves, the best detection achieves the highest detection rate, minimum false positive rate, and smallest Maxion-Townsend cost [112]. Table 6.2 summarizes the comparison.

Table 6.2: A comparison between the best detection outputs for the previous four SWS approaches sorted by user category and Maxion-Townsend cost

SWS Approach	User Category	False Positive %	Hit %	Maxion-Town Cost
TSL	Local	2.2	92.24	20.96
TSLSA	Local	3.1	90.49	28.11
ASSL	Local	3.2	84.01	35.19
MCE	Local	3.4	80.83	39.57
TSL	Server	2.2	95.74	17.46
MCE	Server	2.6	94.61	20.99
TSLSA	Server	2.73	90.71	25.67
ASSL	Server	2.8	88.98	27.82
TSL	System	0.6	98.94	4.66
MCE	System	0.8	97.31	7.49
ASSL	System	0.8	96.27	8.53
TSLSA	System	1.05	91.62	14.68

As shown in Figure 6.10 shows that both the *SWS* and the characteristic of the user activities affect the masquerader live time. As an example, system user sessions are the longest one, because they reflect some operating system activities. Instead, server user sessions are among the shortest ones, because each reflects a small set of activities e.g., sending an email or transferring a file. The length of a local user session changes according to the user behaviour.

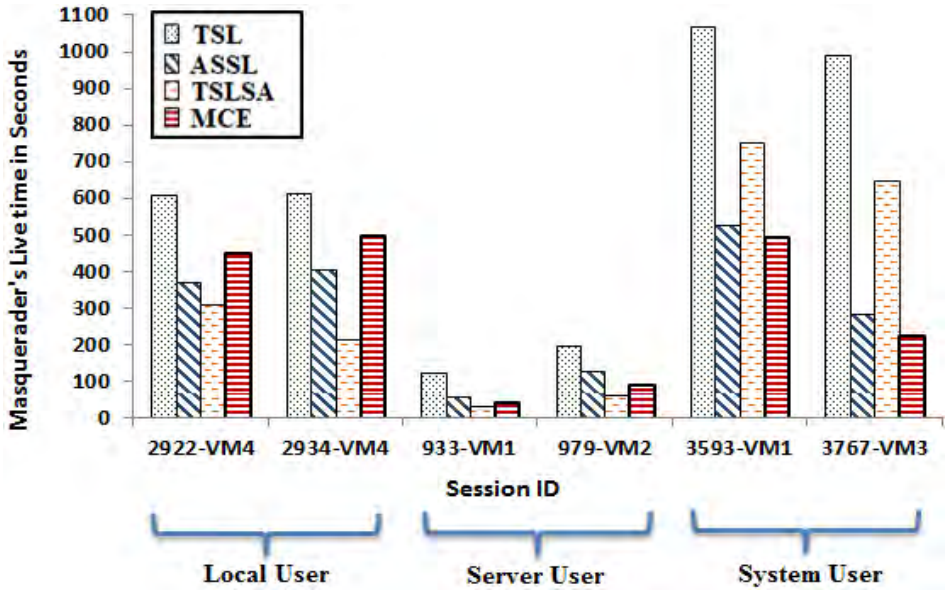


Figure 6.10: The masquerader live time in seconds for local, server, and system users in some attached sessions

According to the output of our comparison, we use TSLSA for local users, because of the low regularity of their data and MCE for both server and system users that have highly regular data.

6.2.2.2 Scoring System

The input of the DDSGA is a tuple with n+1 fields where "n" is the length of the session

$$\{X\ (U, S, P, F, M, T), y_1(r_1), y_2(r_2)\ldots y_n(r_n)\}$$

Where,

- y_i is the input parameter, a file or process name.
  - r_i: the return parameter of the executed pattern, is zero if the pattern was successfully and one otherwise

- *X* is the session ID followed by its input parameters,
  - *U*: the current user id.
  - *S*: the login source IP address.
  - *P*: the login period,
  - *F*: a Boolean value to define whether there is a login failure in the session.
  - *M*: a Boolean value to define if the session has a masquerade or not.
  - *T*: the detection threshold for the session user.

S, P, and F are the input of the neural network described in Section 6.4

For example, a valid tuple is:

{713-VM1 (2143, 135.013.216.191,"Afternoon", 0, 0, 0.72), /usr/lib/fs/ufs/(0), /usr/ucb/whoami(1), ...}.

DDSGA computes the detection score for the session and compares it against the threshold to determine whether the session is a masquerade. Then, it compares this score against "*M*" to compute the false alarms and hit ratio.

We have modified the DDSGA scoring system to work with the extracted features of the "Behaviours Triangle Model", see Figure 6.11. The system rewards a mismatched pattern in the test sequence in two cases: If the test pattern has previously appeared in the user lexicon or if the access or executed pattern was successful.

The first case tolerates various permutations of previously observed patterns without reducing the detection score significantly. In the second case the scoring system rewards successful action and penalizes failed one, because we noticed that most of masqueraded patterns include failure attempts since a masquerader usually lacks some knowledge of the victim file system. The next section highlights the evaluation of the scoring system in terms of security and complexity for the three correlation models.

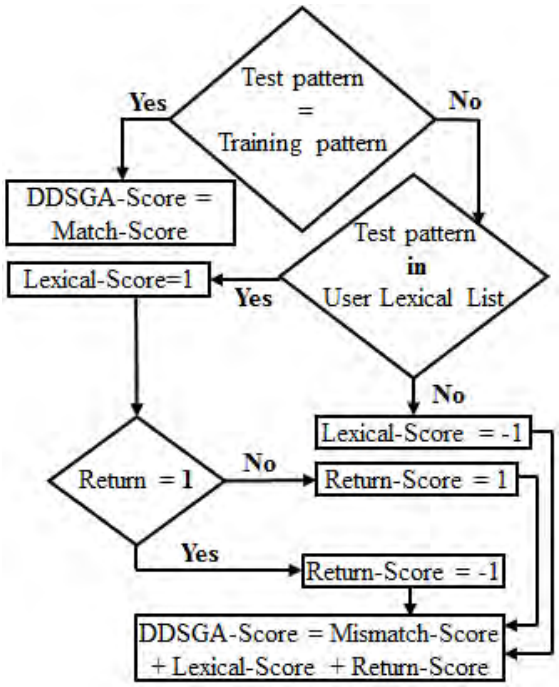


Figure 6.11: A flowchart for the modified DDSGA scoring system.

6.2.3 The Independent, Audit Exchange, and Centralized-Backup Models

We have developed three correlation models of a user behavior for different deployment models namely, Audit Exchange, Independent, and Centralized-Backup. The first two models work with the original CIDS framework and both are detailed in [142]. The third one works with the improved framework, CIDS-VERT. Section 6.2.4 discusses the experimental results of the first two models using the CIDS-Testbed and of the third model using the CIDS-VERT-Testbed. In the following, we outline the three models and refer to [142] for further details.

(A) Audit Exchange Model

Here nodes exchange their audit data so that each one stores any audit data for any of its current users. Nodes also exchange:

(1) The alignment score computed by the CIDS detector component.
(2) The alerts fired by the HIDS component.

This balances the detection overhead among nodes with no single point of failure. The detection efficiency is high because the user audit is

concentrated in one node and the masquerader surviving is much shorter than in model B, see Figures 6.13 and 6.14. As a counterpart, the model needs a fast periodic update and the exchange of the audit data that increases the cloud network overhead and hinders scalability. Furthermore, in highly overloaded networks some audit data may be lost in the exchange see Figure 6.15.

(B) The Independent Model: The detection phase of this model uses the same two parameters of model A. A cloud node *CN* evaluates login usage patterns of a user *U* using both CIDS and HIDS detectors and by using the behavior-based and signature-based of *CN* without interacting with other nodes. If the CIDS detector of *CN* fires an alert, the current login usage patterns are checked against the audit data of *U* in the other nodes until one of them accepts the current pattern. If no node related to U accepts the pattern, the current login session is marked as a masquerade attack. The model advantages are:

(1) It does not require a periodic update of user audit data in each node. The regular periodic backup for VMs data is similar to that of other models,
(2) A very low overhead for the cloud network, as data is exchanged only if the detection score is less than the threshold. In this case, nodes exchange the test audit data of the login session, see Figure 6.15,
(3) High detection efficiency in terms of hit and the false alarm rates close to that of model A, see Figure 6.13.
(4) A lower processing overhead than the other models, because each node executes the DDSGA alignment only if the detection score is less than the threshold. The detection time is directly proportional to *NN*, the number of nodes that have audits for user *U*. See Figure 6.16.

As a counterpart, the surviving of a masquerader is longer than in models A and C because as *CN* increases it also increases the time to analyze the audit data in all nodes, see Figure 6.14. Hence, this model does not scale to large clouds.

(C) The Centralized-Backup Correlation Model

In this model, users VMs send their audit data to a reserved management VM that has a complete view of any user audit data to analyze and report the final alerts. The management VM is backed up to some other VMs as explained in Section 3.2.1 to balance the detection overhead among the

management VMs with no single point of failure. This model achieves the best detection efficiency because the user audit is concentrated in one place and no audit data may be lost. The masquerader surviving is very short comparing to model A and B, see Figures 6.13 and 6.14 with low network overhead. The detection time is inversely proportional to the number of management VMs that reduce the processing overhead in the active management VM. This speeds up the detection phase and protects the IDS components from tampering by any attacker. On the other hand, the network overhead increases with the number of management VMs, see Figures 6.15 and 6.16. Furthermore, the model requires several resources as it reserves some management VMs for detection.

6.2.4 A Comparison of the Three Models

We have applied DDSGA to all users in the CIDD dataset and focused our evaluation on local users with a large deviation in their behaviours. The experiments compare the three models in terms of four values:

A. Accuracy and efficiency using both the ROC curve and Maxion-Townsend cost.
B. Average masquerader live time per session,
C. Average transmitted data per session during the detection time,
D. Average detection time per session.

(A) The accuracy and efficiency

To evaluate the accuracy and efficiency of the models, we focus on the effects of the two DDSGA alignment parameters computed for each user i.e., the detection threshold and the scoring system rewards and penalties, on the detection accuracy. The false positives, false negatives and hit ratios are computed for each user and then transformed into the corresponding rates that are summed and averaged over all users. Equations 6.4, 6.5, and 6.6 show the metrics used by DDSGA.

$$TotalFalsePositive = \left[\left(\sum_{k=1}^{nu}\left(\frac{fp_k}{n_k}\right)\right)/nu\right] * 100 \quad (6.4)$$

Where:

- *fp = No. of false positive alarms,*
- *n = No. of non-intrusion sessions,*
- *nu = No. of users in CIDD dataset (84 in our case)*

$$TotalFalseNegative = \left[\left(\sum_{k=1}^{nui}\left(\frac{fn_k}{ni_k}\right)\right)/nui\right] * 100 \quad (6.5)$$

Where:

- *fn = No. of false negatives,*
- *ni = No. of intrusion command sequence blocks,*
- *nui = No. of users who have at least one intrusion block*

$$TotalHitRatio = 100 - TotalFalseNegative \quad (6.6)$$

Figure 6.12 shows the masquerades distribution in the test sessions for some CIDD users and the detection threshold that DDSGA computes for each user in the training phase.

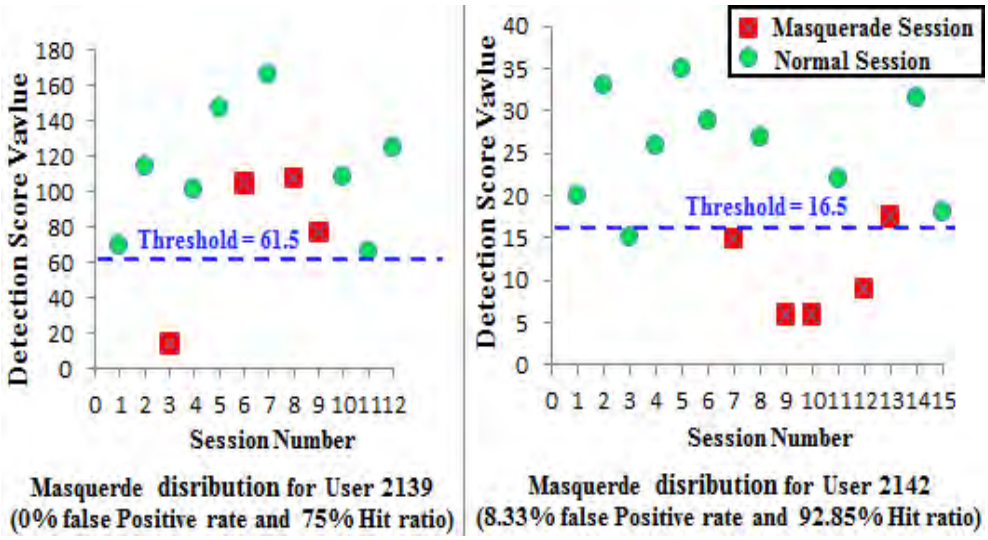


Figure 6.12: DDSGA threshold and masquerades distribution in test sessions of CIDD users 2139 and 2142

To plot the ROC curve, we use distinct values of the alignment parameters that result in false positive rates in the x-axis and the corresponding hit ratios in y-axis. Figure 6.13 shows the ROC curves for each model with the scoring system, the Centralized-Backup model without the scoring system, and the No-Correlation model with the scoring system.

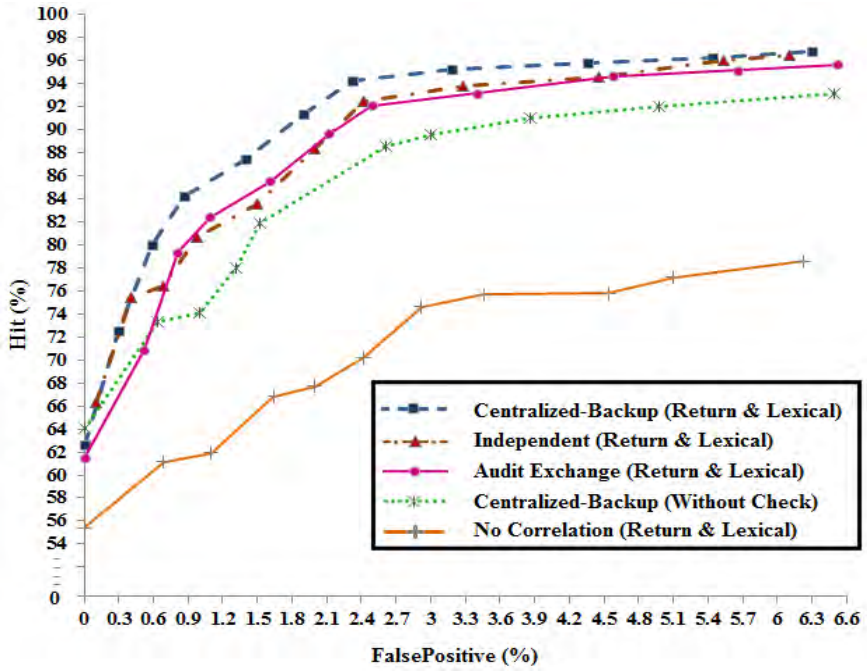


Figure 6.13: The ROC curve for the three correlation models with and without the scoring system.

Table 6.3: The best accuracy of the three Correlation Models sorted by Maxion-Townsend cost

Correlation Model	False Positive %	Hit %	Maxion-Town Cost
Centralized-Backup model with return and Lexical Check	**2.32**	**94.24**	**19.68**
Independent model with return and lexical check	2.42	92.44	22.08
Audit Exchange model with return and lexical check	2.49	92.09	22.85
Centralized-Backup model without the scoring system (without check)	2.61	88.99	26.67
No Correlation Centralized-Backup model with return and lexical check	2.92	74. 60	42.92

Figure 6.13 and Table 6.3 show that the Centralized-Backup model with the scoring system results in the highest hit ratio with the corresponding lowest false positive rates. The ability of the scoring system in tolerating a large number of mutations and deviations in user behaviours increases the hit

ratio by about 5.25% and reduces Maxion-Townsend cost by 6.99. The correlation of the user audits in all the cloud VMs is mandatory to build a consistent profile and it improves the hit ratio by about 19.64% and reduces Maxion-Townsend cost by 23.24%.

(B) Average Masquerader live Time

We compute the average masquerader live time over all sessions for the three correlation models. In the Centralized-Backup model, a larger number of management VMs reduces the computational overhead and, consequently the live time as well. As a counterpart, it increases the cloud network overhead. Therefore, we experimentally determined the optimal number of management VMs. Figure 6.14 shows that the shortest live time is achieved if two management VMs are used.

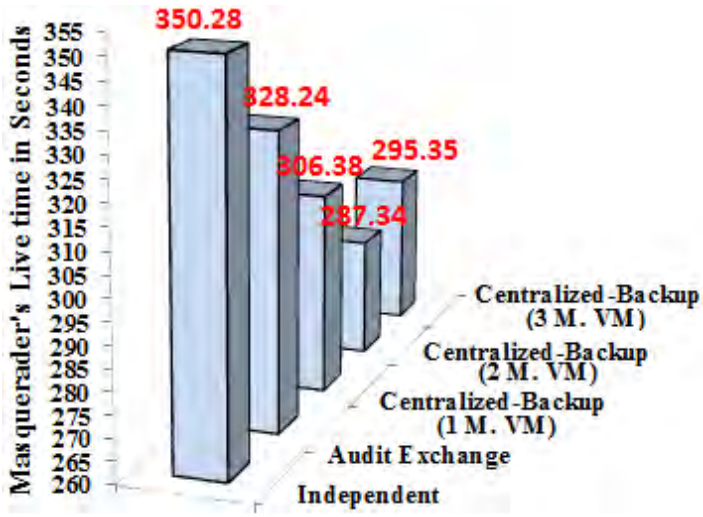


Figure 6.14: Average Masquerader live Time per session in the three correlation models

(C) Average Network Overload per session.

We compute the overhead on the cloud network in terms of the average amount of data that each model transmits in a session. According to the considered model, the VM(s) that runs the detection task can send user audits or current active session to the other VMs. Figure 6.15 confirms that the Independent model is the most lightweight one and that the network overhead of the Centralized-Backup model is directly proportional to the number of management VMs.

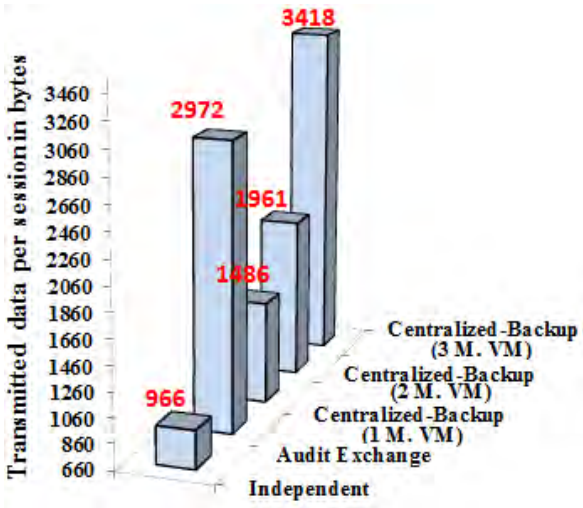


Figure 6.15: Average transmitted data per session in bytes in the three implementation models

(D) Average detection time per session

The average detection time is affected by the machine capabilities and available processing resources. The size of the test session, the corresponding training sessions, and number of user VMs in the cloud system are further important factors. Figure 6.16 shows that the detection time of the Independent model for a user who has audits distributed among three VMs depends upon *NN*, the number of cloud nodes running these VMs. In this way, the Independent model distributes the training audits among the cloud nodes and each node independently runs the detection process using some training records. The shortest detection time is achieved if the detection score in the first node is larger than the detection threshold, and this time increases as *NN* increases. The 3VMs audits label of the independent model in Figure 6.16 means that user audits are distributed across three nodes, each with one VM for that user. By comparing the 3-VM columns against the other models, we notice that the improvement due to the Independent decreases as *NN* increases. If *NN* is equal to 3, this model results in the worst detection time. Therefore, this model is ideal with a small number of users and VMs and it cannot be adopted in large cloud such as public or hybrid ones. On the other hand, the Centralized-Backup model has a reasonable detection time that may be reduced by increasing the number of management VMs. Hence, the model is more elastic and scalable and it may be adopted in large clouds.

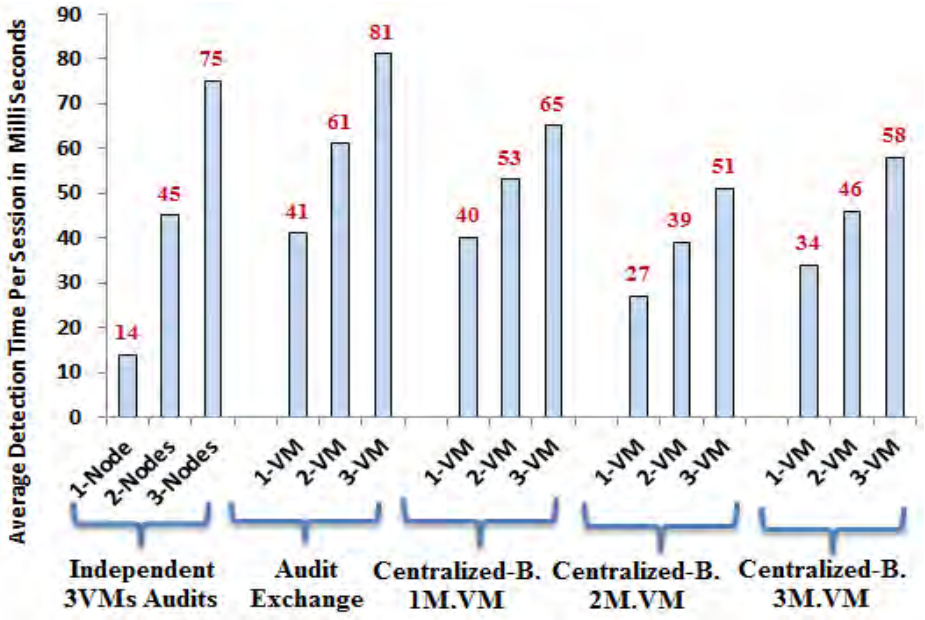


Figure 6.16: Average detection time per session in milliseconds in the three models

The previous experiments show that in the CIDS framework, the Independent model outperforms the Audit Exchange. The performance and the accuracy of the Independent model are acceptable in small and private cloud networks. Instead, the Centralized-Backup model works efficiently with CIDS-VERT in large clouds such as public and hybrid ones.

6.3. Detecting Masquerade in Network Environment Based on NetFlow Data Analysis

While current detection strategies directly monitor the hosts and IDS components to gather and process host data, a cloud system being massively distributed and interconnected requires a network centric approach to masquerade detection [130]. This approach preserves the privacy as it only needs statistical information on the network traffic and it can replace user identifiers, source and destination IP addresses with encrypted or anonymous values. A NetFlow analysis, e.g., an analysis of the network activities of a user, can generate a unique and useful user network profile to detect potential masqueraders whenever the host data is not accessible or legal/ethical restrictions related to the user privacy apply [130, 35]. Furthermore, while some organizations do not routinely collect host audits on all users, by

default any organization collects statistical information on network traffic for network administration, monitoring and troubleshooting. As a counterpart, a NetFlow analysis classifies network activities through their source IP address. Therefore, if several users share the same address e.g., users log to the same VM or remotely access their cloud VMs through the same server, the network activity profile will reflect the activity of the server or of the VM connected to this source IP address rather than the activity of a single user. This is the reason why our analysis detects the masquerade IP source address.

We propose three approaches to capture the NetFlow data namely, the log server, the inside-outside workstations, and the physical\virtual switch monitoring. The data of the log server approach consist of mail or FTP server logs. The log server pairs the source IP address with its NetFlow traffic captured by the sniffer tool and it determines the start and end of user sessions. The inside-outside workstations approach considers two nodes located, respectively, on the inside and on the outside of a router and uses a sniffer tool to capture data that crosses the router through any TCP/IP connection between the two nodes. In the physical\virtual switch monitoring, a network IDS sensor like Snort is installed in one physical node or in a VM that is connected to a physical switch port or to a promiscuous port on a virtual switch where all traffic is mirrored. In our CID-VERT framework, virtual network traffic is forwarded to a management VM(s) where Snort analyzes it and gets the required NetFlow data. In the CIDS framework, network traffic is forwarded to one physical host that runs SNORT and it is attached to the auditor system to capture the network audits for user sessions as in Chapter 3.

6.3.1 Feature Extraction from NetFlow data in the cloud Network.

Our goal is to use sequences of the destination IP addresses of the machines a user accesses and the name of the corresponding protocol in the same way host-based masquerade detection uses system call access patterns. The correlator component of both CIDS and CIDS-VERT frameworks defines the start and the end of user sessions as in the log server approach. After that, the auditor system in CIDS or the event correlator in CIDS-VERT filters the NetFlow data of the user host session according to the user source IP address. The NetFlow session length is equal to the host session length

computed by one of the four SWS approaches in Section 6.2.2.1. Figures 6.17 and 6.18 show the distribution of the destination IP addresses in the network activity profiles of two source IP addresses that correspond to local and server user sessions, respectively. In general, system users do not use the network environment because their activities are more related to some operating system tasks. These two figures show that server user sessions are more regular than local users ones because their destination IP addresses are more specific and consistent than the corresponding ones in local user sessions. In addition, server users implement their tasks through just a few protocols e.g., "FTP" to transfer files and "SMTP" to send emails, while the network activities of local users use a larger number of protocols. We also notice that NetFlow data is less regular than system call patterns because the consistency of user network activities is lower than that of host activities. In addition, a network activity profile reflects the behaviour of a source IP address that may be shared among several users. This may reduce both the regularity and the consistency of data in the network activity profile that in turn, reduces the accuracy of detection. Therefore, we modify the DDSGA scoring system, see Section 6.3.2, to be more flexible and take into account these features of NetFlow data.

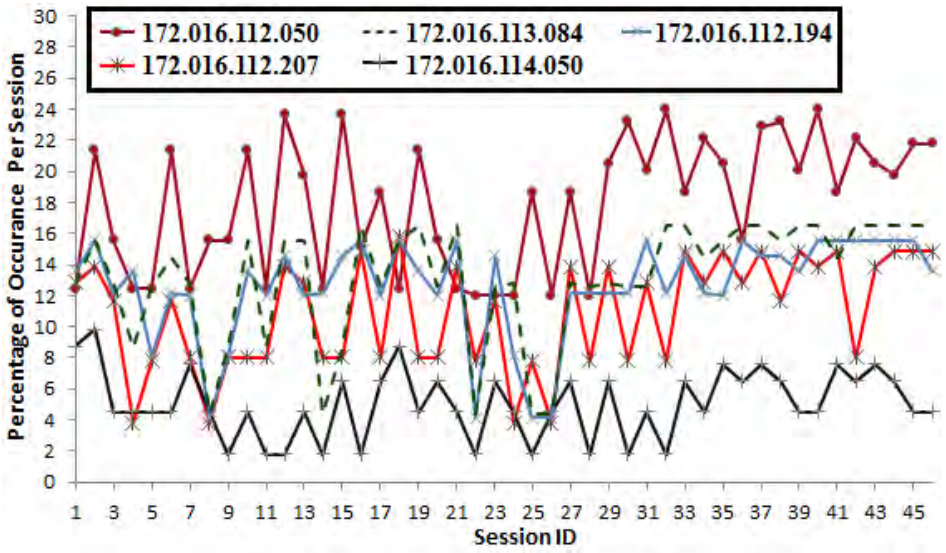


Figure 6.17: The distribution of NetFlow destination IP addresses in local user sessions (user ID 2143 in CIDD)

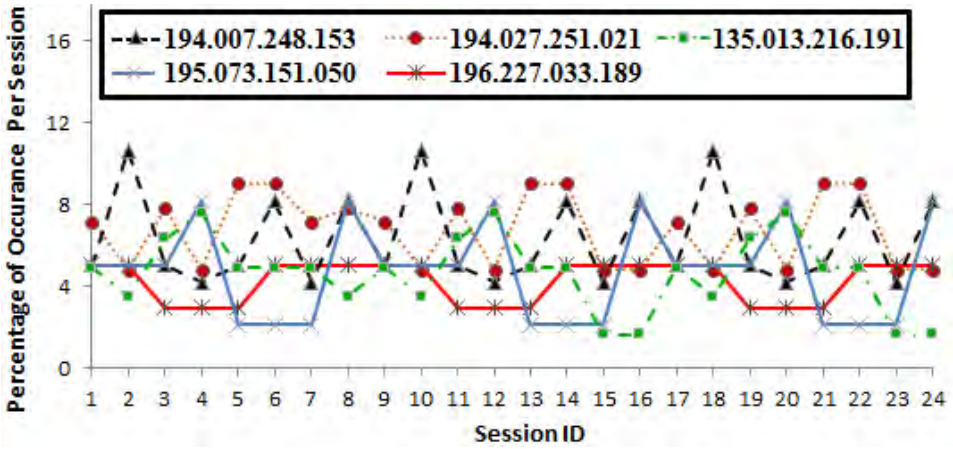


Figure 6.18: The distribution of NetFlow destination IP addresses in server user sessions (user ID 2059 in CIDD)

DDSGA is applied to NetFlow data that are tuples of the form:

{X (U, S, DT, M, T), $y_1(p_1,d_1)$, $y_2(p_2,d_2)$... $y_n(p_n, d_n)$}.

Where "*n"* is number of content patterns in the session and *X* is the session ID with the following input parameters:

- *U*: the current user id.
- *S*: the user source IP address.
- *DT*: the date and time of the session in the form (Week Day Time).
- *M*: a Boolean that defines if the session has a masquerade or not.
- *T*: the detection threshold for the session user.
- y_i: a session content pattern where p_n is the protocol name and d_i the destination address.

For example, a valid tuple is:

{3112-VM3 (2143, 135.013.216.191, W4, D3, 8:56:47, 0, 0.64), (telnet, 172.016.112.050), (domain/u, 172.016.112.020)}.

Again, DDSGA computes the detection score and determines whether the session has a masquerade patterns. Then, it checks "*M"* to compute the false alarms and hit ratio. Figure 6.19 shows two training sessions with the extracted NetFlow features for a local user and a server one.

Local User

= = = = = =

Session Head:

SessionID, UserID, SourceIP, Week, Day, Time, Real-Masquerade?

3112-VM3, 2143, 135.013.216.191, W4, D3, 08:46:27, 0

Session Contents: (Protocol Name, Destination IP)

(telnet,172.016.112.050),(domain/u,172.016.112.020),(smtp,172.016.113.105),
(smtp,172.016.112.194),(domain/u,172.016.112.020),(ftp,172.016.114.148),
(smtp,172.016.113.084),(finger,172.016.114.168),(http,172.016.112.050),.....

Server user

= = = = = =

Session Head:

SessionID, UserID, SourceIP, Week, Day, Time, Real-Masquerade?

2702-VM3, 2059, 172.016.114.169, W5, D2, 11:37:22, 0

Session Contents: (Protocol Name, Destination IP)

(telnet, 195.073.151.050),(smtp, 196.227.033.189),(smtp, 208.225.121.198),
(ftp, 187.187.187.187),(ftp, 209.001.120.050) ,(http, 198.068.020.079),.....

Figure 6.19: Two training sessions with the extracted NetFlow features for local user 2143 and server user 2059

6.3.2 The NetFlow Scoring System

The scoring system for NetFlow data, see Figure 6.20, has been defined by modifying the DDSGA scoring system.

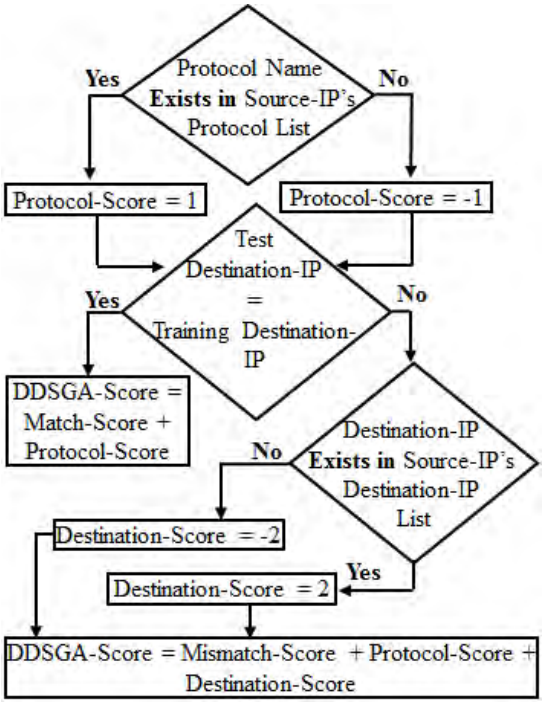


Figure 6.20: A flowchart for the modified NetFlow scoring system.

The system rewards a mismatched destination IP pattern in the test sequence in two cases: the protocol name has previously appeared in the protocol list of the source IP in the test sequence or the destination IP has previously appeared in destination-IP list of the source IP. The first case tolerates permutations of observed patterns of the combination (destination IP and protocol name) without reducing the detection score significantly. In fact, the destination IP and protocol name may be unrelated, because a user may access the same destination with distinct protocols for distinct tasks. Also the second case tolerates various permutations of previously observed patterns because a user may access a set of destinations in distinct orders. We have applied DDSGA to the NetFlow data for each source IP sessions in CIDD as described in Section 6.2.2 using the Centralized-Backup model of CIDS-VERT. Figure 6.21 and Table 6.4 show the detection accuracy of DDSGA in terms of both the ROC curve and Maxion-Townsend cost respectively.

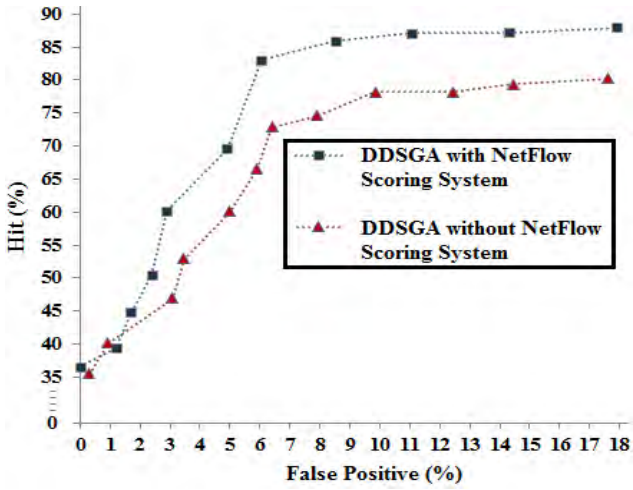


Figure 6.21: The ROC curve for the DDSGA approach on the NetFlow audits with and without the scoring system

Table 6.4: The best accuracy of the DDSGA approach on the NetFlow audits with and without the scoring system sorted by Maxion-Townsend cost

Approach	False Positive %	Hit %	Maxion-Town Cost
DDSGA with NetFlow scoring system	**6.05**	**83.05**	**53.253**
DDSGA without NetFlow scoring system	6.39	72.89	65.487

Figure 6.21 and Table 6.4 show that the NetFlow scoring system results in the highest hit ratio with corresponding lowest false positive rates. The scoring system increases the hit ratio by about 10.16% and reduces Maxion-Townsend cost by 12.234.

6.4. A Neural Network Model to Integrate the Host and Network Detection Outputs

The integration of host and NetFlow detections through a Threshold Logic Unit (TLU) [42], see Figure 6.22, improves the accuracy and the efficiency of the detection. The TLU works for each user independently to be adapted to the consistency of the user behaviour. A Group of TLUs consist of a complete neural network model for all users in the CIDD dataset. The TLU has 3 layers: one input layer of dimension n= 4 inputs, an hidden layer with 1 summing junction neuron, and an output layer with 1 neuron connected to an activation function that depends upon the TLU operational mode. The detection mode uses a threshold function which returns 0 if the sum of the input is less than a threshold (t_j), i.e., if there is a masquerade attack in session s. Otherwise, the function returns 1. In learning mode, the neural network uses the sigmoid function to range the outputs between 0 and 1 to adapt with the training phase and the weight adjustments, see Section 6.4.2

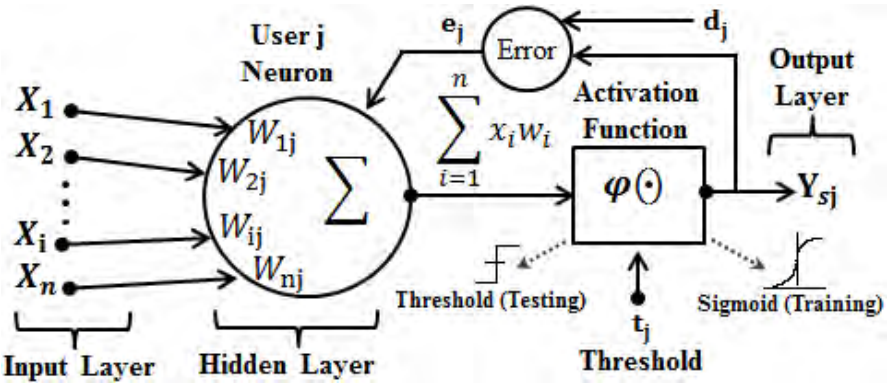


Figure 6.22: The neural network model in the training mode for one user

6.4.1 The Detection Mode of the TLU.

The mathematical model for the TLU output for a session s of user j *is* Y_{sj} as defined in Equation 6.7:

$$Y_{sj} = F(X_{sj}), X_{sj} = \{DDSGA_{NetScore}, S, P, E\} \quad (6.7)$$

where F is a non-linear activation function implemented by the TLU. This function is denoted by $\varphi(.)$ and it acts as a squashing function, such that the output of a TLU belongs to a given range. In detection mode, the neural network uses the Threshold Function as a transformation function. X_{sj} is the TLU input parameters of session s of user j. The input parameters are:

- $DDSGA_{NetScore}$: the overall detection score for the active user session according to user audits in both host and NetFlow data,
- *S:* The login source IP address,
- *P:* The login period,
- *E:* A Boolean value to signal any login error/failure in the active session.

The last three parameters are collect from the host audits as in Section 6.2.2.2. An example of a valid input record is:
{0.31, 172.016.114.169, Morning, 0}.

We use Equations 6.8 and 6.9 to compute the $DDSGA_{NetScore}$ parameter.

$$P_{Cmasq}(U_j) = \sum_{a=1}^{m} \left(\frac{P(U_j) * P(IP_a)}{\sum_{k=1}^{n} P(U_k)} \right) + P(U_j) \quad (6.8)$$

$$DDSGA_{NetScore}(U_j) = \theta_j - P_{Cmasq}(U_j) \quad (6.9)$$

Where:

- $P_{Cmasq}(U_j)$*:* the probability that the current active session of U_j has a masquerade patterns according to U_j behaviors in all cloud VMs. It includes the probability that the masquerader can be detected by the behaviour of its login IP(s).
- $P(U_j)$: the probability that the current active host session of U_j has a masquerade behaviour according to U_j behaviors in all cloud VMs as computed as in Section 6.2.2. This probability does not include user IP behaviors.
- *m:* the number of IP(s) that U_j uses to login to the cloud.
- *n:* the number of cloud users who share the same IP_a of U_j
- *k:* an index for the current user who shares the same IP of U_j.
- *a:* an index for the current IP address of U_j.
- $P(IP_a)$: the probability that IP_a reveals to be a masquerader as computed in Section 6.3.

Consider the case where U_1, U_2 and U_3 share IP_1 *and* IP_2 and the probabilities that IP_1 *and* IP_2 could be used by a masquerader are: $P(IP_1)$ = 0.4, and $P(IP_2)$ = 0.5. The probabilities that U_1, U_2, and U_3 reveal to be masqueraders according to their behaviors in all the cloud VMs are: $P(U_1)$=0.4, $P(U_2)$=0.3, and $P(U_3)$= *0.6*, and the detection threshold θ_j = 0.75. We apply the previous equations to compute $P_{Cmasq}(U_j)$ for each U_j to determine which one is a real masquerader according to both the corresponding host and network audits.

$P_{Cmasq}(U_1)$= ((0.4*0.4) / (0.4+0.3+0.6) + (0.4*0.5) / (0.4+0.3+0.6)) + 0.4 = 0.6769 < θ_j (not masquerader)
$DDSGA_{NetScore}(U_1)$= 0.75 - 0.6769 = 0.0731.

$P_{Cmasq}(U_2)$= ((0.3*0.4) / (0.4+0.3+0.6) + (0.3*0.5) / (0.4+0.3+0.6)) + 0.3 = 0.5076 < θ_j (not masquerader)
$DDSGA_{NetScore}(U_2)$= 0.75 - 0.5076= 0.2424.

$P_{Cmasq}(U_3)$= ((0.6*0.4) / (0.4+0.3+0.6) + (0.6*0.5) / (0.4+0.3+0.6)) + 0.6 = 1.0153 > θ_j (masquerader)
$DDSGA_{NetScore}(U_3)$= 0.75 - 1.0153 = - 0.2653.

6.4.2 The Training Mode of the TLU.

While DDSGA uses unsupervised learning, all TLUs are trained using a supervised learning algorithm. Since the TLU contains only one hidden layer with one neuron, a simple algorithm such as the Generalized Delta Procedure (GDP) [42], simplifies the learning process that occurs off-line during the training phase to compute the input weights for each user independently.

The GDP computes the error i.e., the distance between the desired response and the actual one, and backward propagates a fraction of it through the network. Each neuron uses this fraction to tune its weights and threshold values to reduce the network error for the same input. This procedure is repeated until the individual or total errors in the responses become smaller than a specified value. At this point, the learning ends and we can use the neural network to produce responses to new input data.

To find the weight change rule, we exploit that the sigmoid function is differentiable. If there is an error for the given input, the weights are adjusted according to Equation 6.10:

$$W_{ij} \leftarrow W_{ij} + C * e_j * f(1-f) * X_{ij} \quad (6.10)$$

Where,

- W_{ij}: The weight of input i to the hidden neuron j.
- C: A constant of learning. We take C to be universally 1.
- e_j: The error distance. $e_j = d_j - f$, d_j is the desired result from the training set (0 or 1).
- f: The actual result from the TLU with the sigmoid function, and $f(\text{input}) = 1 / (1+e^{-\text{input}})$, $f(1-f) \rightarrow 0$, where $f \rightarrow 0$ or $f \rightarrow 1$. This means that the weight change can occur only within 'fuzzy' region surrounding the hyper plane near the point $f = 0.5$.
- X_{ij}: The input i to the hidden neuron j.

The training also tunes the threshold parameter t_j. By applying the concept of Augmented Vectors [156], we add a new input to the TLU, T_{n+1}, and a corresponding weight, W_{n+1}. T_{n+1} always takes on a value 1 and W_{n+1} is adjusted as the other weights. The threshold value t_j is fixed at 0. After the training, we remove T_{n+1} and set t_j to the value of W_{n+1}.

The weights and threshold values in the TLU are randomly initialized but the weight of $DDSGA_{NetScore}$ is always the largest one because it mostly affects the TLU. The initial weight vector is: {0.85, 0.05, 0.05, 0.05}. Figure 6.23 shows how the learning iterations affects the error distance for local user 2143 using 48 samples of training sessions.

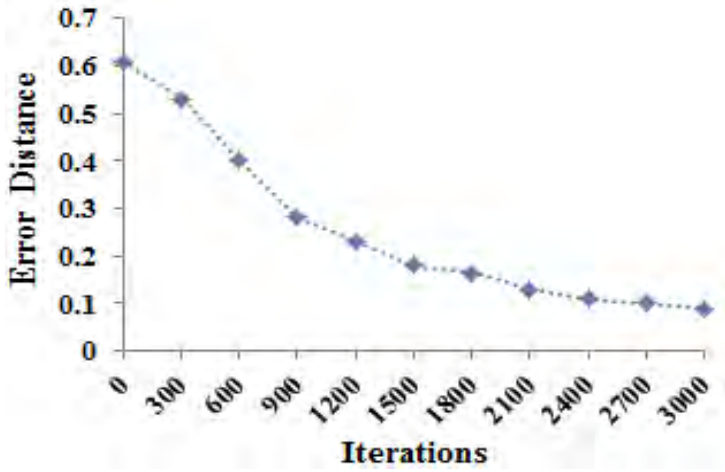


Figure 6.23: The effect of learning iterations on the error distance

6.4.3 Performance Evaluation of the Integrated Approach.

We compare the accuracy and the average detection time per session for the three detection approaches: neural network, network, and host based using the Centralized-Backup model with two management VMs. Figure 6.24 confirms that the neural network model results in the longest average detection time per session because, it integrates the outputs of other detections. However, it results in a better accuracy than the other models as shown in Figure 6.25 and Table 6.5.

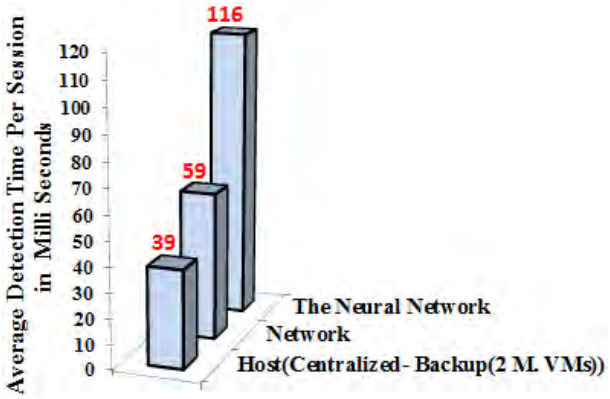


Figure 6.24: Average detection time per session in milliseconds in host, network, and the neural network models.

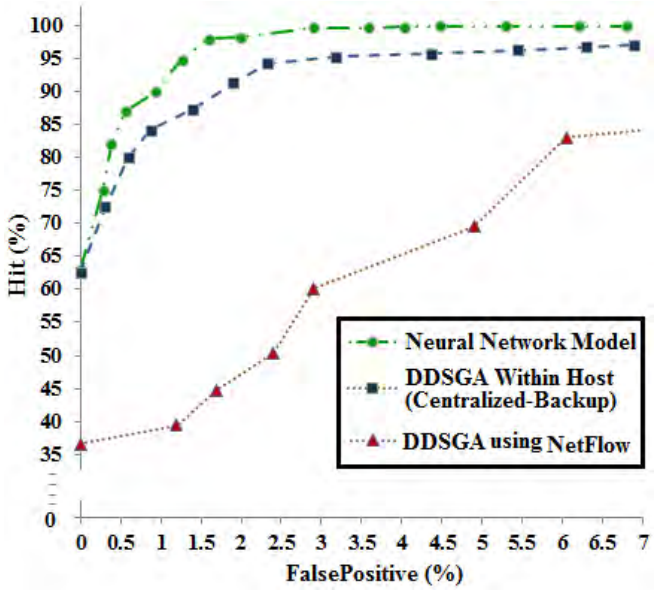


Figure 6.25: The ROC curve for the DDSGA approach using network, host, and neural network approaches

Table 6.5: The best accuracy of the three detection approaches sorted by Maxion-Townsend cost

Approach	False Positive %	Hit %	Maxion-Town Cost
Neural Network Model	**1.59**	**98.07**	**11.49**
DDSGA within Host system	2.32	94.24	19.68
DDSGA using NetFlow	6.05	83.04	53.25

Chapter 7

Detecting Masqueraders through Security Events and NetFlow Data

This chapter applies the detection strategies of Chapter 6 to different audit data and distinct operating system [157]. The experiments results that this chapter presents confirm those of Chapter 6. The chapter ends with a comparison between its strategies and the ones introduced in Chapter 6.

7.1 Overview

The modeling of user audits is a big challenge in clouds because they support distinct guest operating systems each with its own logging facilities that result in alternative users audits. As possible examples, we recall the Unix Syslog [158] process, the Windows System, Security, or Application event logs [33] and the open source OpenBSM [159] library in both FreeBSD and Mac OS X. Our solution focuses on those audits that almost any operating system produces, namely system calls and security events. While Chapter 6 has experimentally proved that DDSGA can achieve a high performance and accurate detection if applied to sequence of system call audits integrated with audits of a user network activity, this chapter evaluates the performance of strategies that apply DDSGA to sequence of security events audits from the host environment integrated with the NetFlow audits from the network environment based on CIDD dataset. We use the "Behaviours Triangle Model" introduced in Chapter 6 to model a consistent user profile in the host environment based on features extracted from the security events. Then, we build a NetFlow profile for each source IP address in terms of sequences of network actions captured by sniffing network communications as in Chapter 6. DDSGA computes two detection outputs by comparing the active log sessions in the host and the network environments against the corresponding profile. A neural network produces the final output as in Chapter 6. We also have considered the two intrusion detection frameworks, CIDS and CIDS-VERT. We evaluate the efficiency and the computational performance of the host based and of the network based detection in isolation and then the one of their integration.

7.2 Detecting Masquerades Based on Security Events Analysis

This section considers the detection in the host environment based on the security events of the host OS.

7.2.1 Feature Extraction from Security Events in Cloud VMs.

The guest operating systems of the VMs offer an efficient and highly configurable auditing system. An analysis of the corresponding audit data can simplify the detection of several threats against clouds. Here we consider the Windows Event Log Service that records events with information about hardware, software and system components in three logs: application, system and security [33]. Events in the application log are logged by programs and selected by the developers of these programs. The system log contains events logged by Windows system components, e.g. by drivers. The security log records events on valid and invalid logon attempts and on resource usage, such as file operation and process invocation. This log stores most of the information that defines the user behaviours. We extract from the security log six types of audits namely, Privilege Use, Account Management, System Event, Logon/Logoff, Detailed Track, and Object Access. Each of these audits has a group of audit actions. Among them, we focus on the sequence of user actions; i.e., accessed objects and invoked processes.

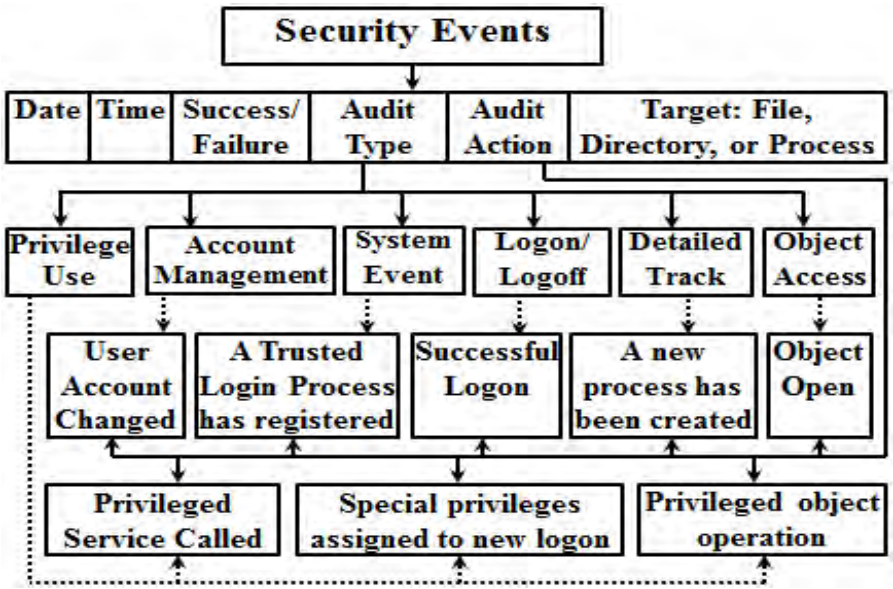


Figure 7.1: Extracted feature from security events

As shown in Figure 7.1, we extract:

(1) The sequence of files or directories opened by the user from the "Object Accessed" audit category.
(2) The sequence of process created or invoked by the user from the "Detailed Track" audit category.
(3) The name of the trusted login processes registered by the system from the "System Event" audit category.

We record the beginning and the end of each user session using the time of each successful logon event as defined through both the "Logon/Logoff" and the "System Event" categories. The Logon/Logoff audit category helps in defining valid and invalid logon attempts to define masquerade attacks. As described in Section 7.2.2.1, the actions extracted from the "Privilege Use" and "Account Management" categories are sensitive actions and their execution fires the detection process.

The feature extraction process is applied to the windows audits of CIDD, which is distributed among the operating systems of distinct cloud VMs. By extracting specific features rather than considering all those of security events, we achieve two advantages. First of all, this reduces the false alarm rates and improves the hit ratio because it reduces the number of log events and it focuses on those that characterize user behaviours. The reduced number of events and of their corresponding arguments also simplifies the DDSGA alignment as it reduces the permutations of these events. Furthermore, this solution speeds up the detection process and reduces the computational overhead by comparing the active session patterns to a few events. This is important in highly overloaded, multi user systems such as clouds. The regularity of the features extracted from the user audits and the consistency of the user normal behaviors help to distinguish normal behaviors from abnormal ones. Chapter 6 has introduced a "Behaviours Triangle Model" to build a distinct activity profile for each cloud user based on sequences of system calls for file\directory accesses and process execution activities. Using the same idea, we build a user profile in terms of three relationships based upon security events:

- User accesses a file\directory. This relation includes the file\directory access patterns.
- Process accesses a file\directory. This relation defines how process accesses a file\directory.

– User invokes a process. This relation defines the sequence of processes the user invokes. As in Chapter 6, we neglect forked processes.

As in Chapter 6, we distinguish human users CIDD data from system users data. In turn, human users are categorized into local and server users. We separate human user activities from system users ones through the user ID and then we check each user separately. Figure 7.2 shows an example of the features extracted from the Windows security events in CIDD.

No.	Host Name	Week -Day	Source IP	Time	Success/ Failure	Audit Type	User ID	Audit Action	Object/Image File Name
259	VM1	W1-D1	*172.16.12.1*	2:37:11	Success Audit	Logon/ Logoff	500	Successful Logon	-
260	VM1	W1-D1	*172.16.12.1*	2:37:11	Success Audit	Object Access	500	Object Open	\KnownDlls\MSVCRT.dll
261	VM1	W1-D1	*172.16.12.1*	2:37:11	Success Audit	Object Access	500	Object Open	\KnownDlls\ole32.dll
262	VM1	W1-D1	*172.16.12.1*	2:37:11	Success Audit	Detailed Tracking	500	A new process has been created	systray.exe
263	VM1	W1-D1	*172.16.12.1*	2:37:13	Success Audit	Object Access	500	Object Open	\KnownDlls
264	VM1	W1-D1	*172.16.12.1*	2:37:13	Success Audit	Object Access	500	Object Open	\KnownDlls\kernel32.dll
265	VM1	W1-D1	*172.16.12.1*	2:37:13	Success Audit	Object Access	500	Object Open	\KnownDlls\user32.dll
266	VM1	W1-D1	*172.16.12.1*	2:37:14	Failure Audit	Detailed Tracking	500	A new process has been created	FINDFAST.EXE

Session Head: UserID, SessionID, SourceIP, Period, Success/Failure(0/1)?, Masquerade(0,1)?
500, 259-VM1, 172.16.12.10, Afternoon, 0, 0
Session Contents: (Object-Path, Success/Failure(0/1))
(\KnownDlls\MSVCRT.dll,0),(\KnownDlls\ole32.dll,0),(systray.exe,0),(\KnownDlls,0),(\KnownDlls\kernel32.dll,0),(\KnownDlls\user32.dll,0),(FINDFAST.EXE,1) .....

Figure 7.2: A user session and its extracted features from Windows security events

DDSGA detects if any legal user was impersonated by a masquerader as in Chapter 6. Section 7.2.2 details how DDSGA detects masquerade attacks through sequences of features extracted from the Windows security events. As shown in Figures 7.1 and 7.4, we build the user profile in terms of the sequence of files the user accesses and of invoked processes. Figures 7.3, 7.4 and 7.5 show the distribution and regularity of the access patterns and of the invoked processes for, respectively, local, server, and system users.

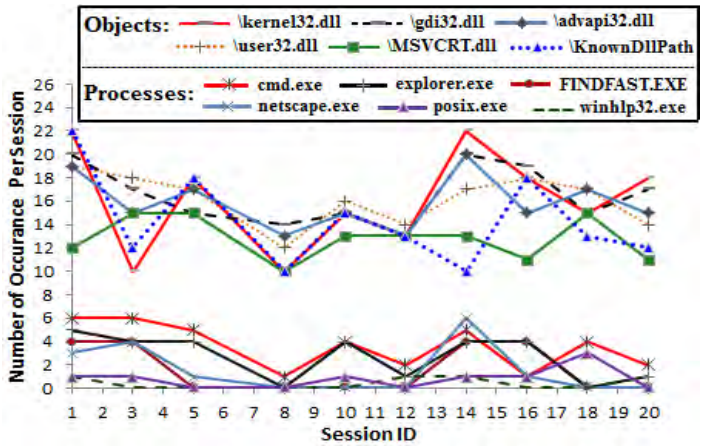


Figure 7.3: The distribution of objects (files and directories) in local user with ID 500 in CIDD.

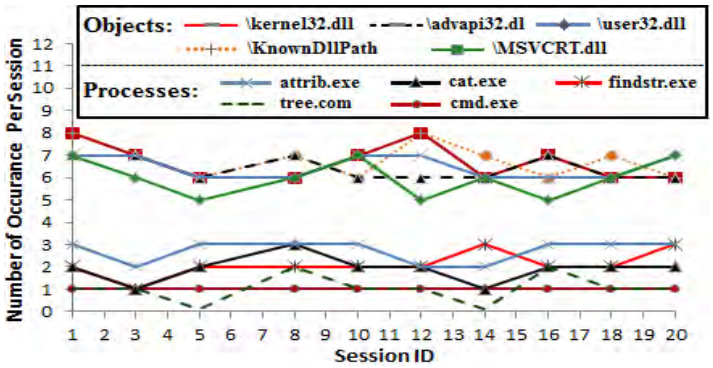


Figure 7.4: The distribution of objects (files and directories) in server user with ID 1031 in CIDD.

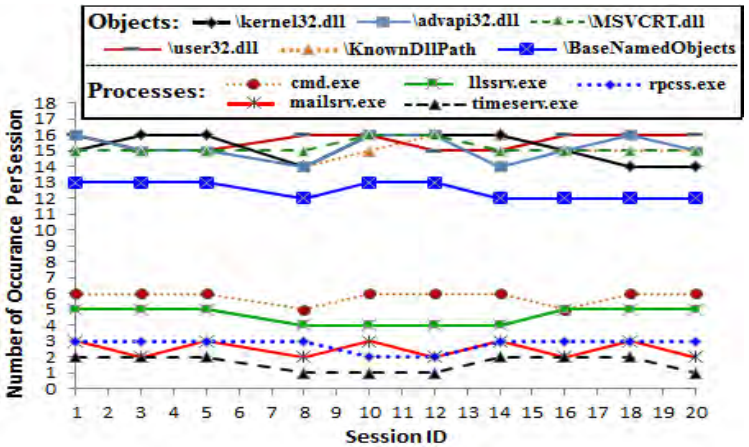


Figure 7.5: The distribution of objects (files and directories) in system user with ID "SYSTEM" in CIDD.

Figures 7.3 - 7.5 show that the behavior of server and system users are highly predictable. As a consequence, deviations in the behaviours of these users will be detected in a more accurate way than for local users. Since the invoked processes are more consistent and predictable than the access patterns, their integration with these patterns can improve detection.

7.2.2 Detecting Masquerade in Windows VMs

To adapt the three phases of DDSGA to Windows security audits in CIDD, Chapter 6 has defined three parameters, namely: the Sliding Window Size (SWS), the scoring system, and the correlation model. The detection phase applies either the Independent or the Centralized-Backup one, according to the characteristics of the cloud system.

7.2.2.1 Choosing the Best Sliding Window Size

In Chapter 6, we have estimated the SWS factor through four approaches. Among them, the most efficient ones are the MCE and the TSLSA. We apply MCE to compute the regularity of the training data using an entropy model described in Chapter 6. We adapt this approach for both server and system users because their training data are much regular than those of local users, while we use the TSLSA approach to adapt the dynamic activities of the local user. The two approaches are evaluated through the detection accuracy using the Receiver Operator Characteristic (ROC) curves [113] and the masquerader live time. We highlight the two approaches in the following.

(a) The Minimum Conditional Entropy (MCE):

This approach is based upon conditional entropy [39] and it measures the regularity of the training data for each user using different *SWS* values to choose the one that results in more regular data with corresponding lower entropy. Figure 7.6 shows the conditional entropy, see Chapter 6, for three kinds of users in the CIDD dataset, i.e., local, server, and system users respectively. The server and system user curves in Figure 7.6 have more than one minimum due to the high data regularity for these users. A lower entropy indicates more regular data and a better detection performance.

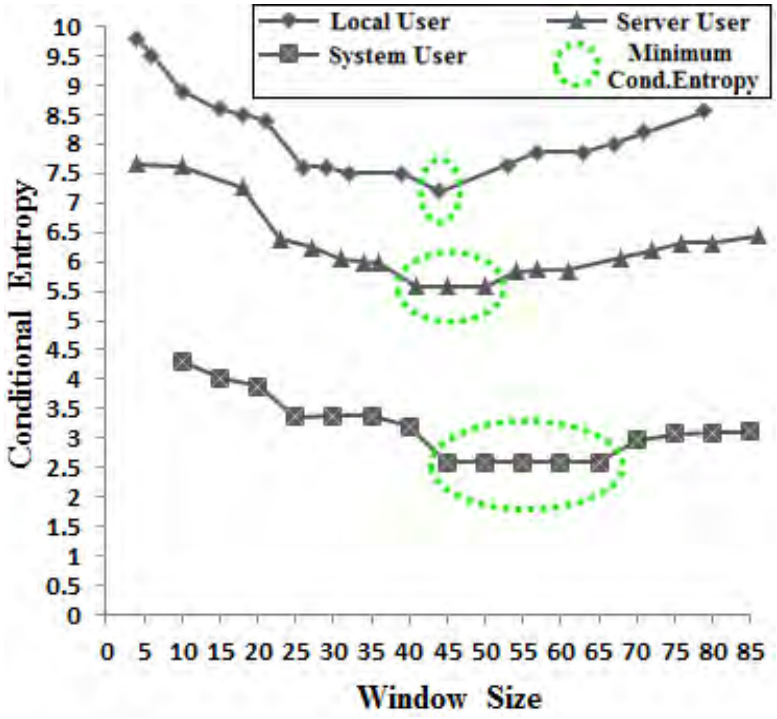


Figure 7.6: Conditional entropy under different SWS for a local user, a server user and a system one.

The results in Figures 7.7 and 7.8 confirm those of Chapter 6 where the accuracy and masquerader live time of the conditional entropy model are acceptable for server and system users due to their highly regular data. Instead, the performance of this model is not satisfactory for local users that have a much lower regularity.

(b) Test Session Length with Sensitive Action (TSLSA):

The TSLSA approach sets the SWS to the length of the active test session but detection can start anytime the user executes a sensitive action. These actions depend upon the OS and type of audits and include the privileged use of system resources and changes to the user account. We consider them because an attacker always tries to exploit critical resources to misuse system resources or to steal some details of a user account. Table 7.1 shows some examples of sensitive actions in Windows audits of CIDD.

Table 7.1: Examples of sensitive actions

User ID	Week	Time	Audit Type	Audit Action	Action Details
500	W1	2:27:55	Account Management	User Account Changed	jouni targets account Name: EYRY
500	W1	2:27:55	Account Management	User Account Changed	katina targets account Name: EYRY
500	W1	2:27:55	Account Management	User Account Changed	kiara targets account Name: EYRY
SYSTEM	W1	2:52:32	Privilege Use	Privileged Service Called	Service:LsaRegisterLogonProces
ANONYMOUS LOGON	W1	2:52:33	Privilege Use	Special privileges assigned to new logon	Privileges: SeChangeNotifyPrivilege
SYSTEM	W1	2:53:04	Privilege Use	Privileged Service Called	Service:LsaRegisterLogonProcess
500	W1	2:56:33	Privilege Use	Privileged object operation	Privileges: SeShutdownPrivilege

TSLSA achieves an acceptable accuracy and a shorter masquerade live time than MCE for all categories but for system users that usually trigger sensitive actions to carry out some operating system functions. We evaluate the performance of MCE and TSLSA for the three user categories through the ROC curve in Figure 7.7. We apply the DDSGA approach in the case of the Centralized-Backup model with the lexical and return checks of the scoring system detailed in Sections 7.2.2.2 and 7.2.2.3. However, the adoption of the Independent model results in the same conclusion.

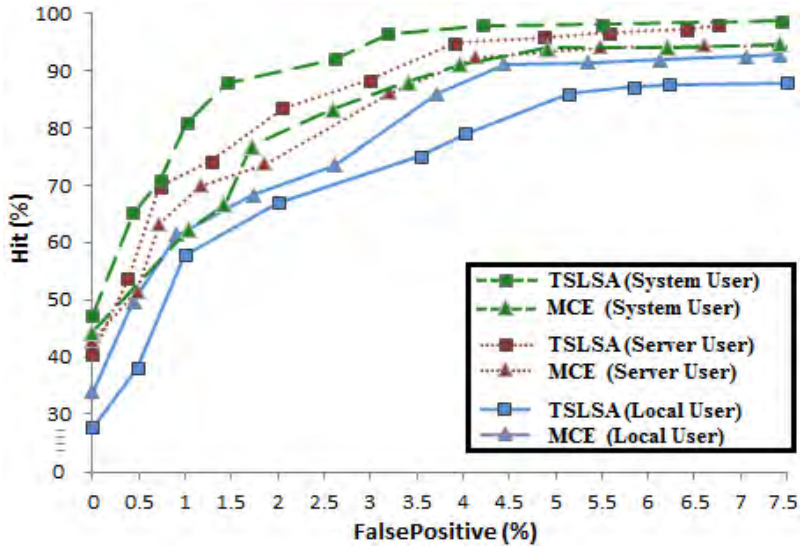


Figure 7.7: The ROC for the three sliding window selection methods for a local, a server, and a system user in CIDD.

The best detection results are those with the highest hit ratio, minimum false positive rate in the ROC curve and the smallest Maxion-Townsend cost [Maxion-Cost]. Table 7.2 summarizes the results.

Table 7.2: A comparison of the two SWS approaches for local, server and system users

SWS Approach	User Category	False Positive %	Hit %	Maxion-Town Cost
TSLSA	Local	4.44	91.24	35.43
MCE	Local	5.14	86.04	44.83
MCE	Server	3.91	94.91	28.54
TSLSA	Server	4.13	92.42	32.36
MCE	System	3.19	96.54	22.60
TSLSA	System	3.97	91.24	32.58

To evaluate the live time of a masquerader in the two approaches, can be evaluated through the chart in Figure 7.8 that show the time for some masquerade sessions in the training data of users in the three categories.

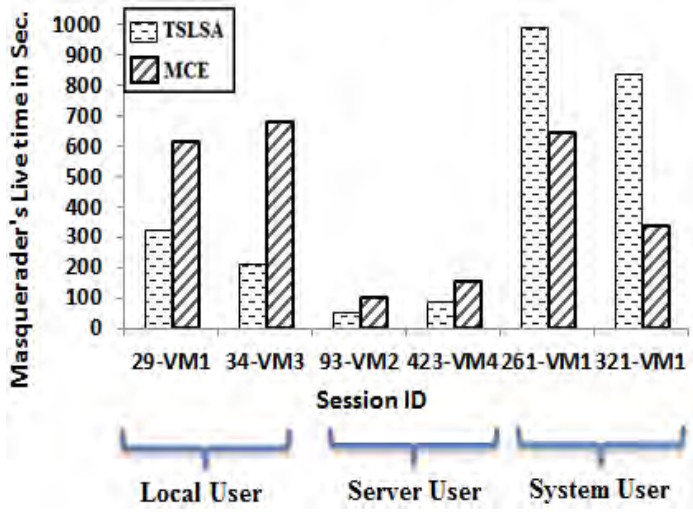


Figure 7.8: Masquerader live times for local, server, and system users

Figure 7.8 shows that the live time of a masquerader is related to both the approach to determine the SWS and the characteristic of the user activities; e.g., sessions of system users are longer than those of other users as they represent some operating system activities. The frequent sensitive actions triggered by system users reduce the accuracy of TSLSA. MCE reduces the

masquerade live time because of the highly predictable and regular data of system user sessions. Server users rarely trigger sensitive actions and their sessions are among the shortest ones. In this way, TSLSA results in a shorter live time of a masquerader than MCE. The length of a local user session changes according to the user behaviours and the regularity of these sessions is very low. This reduces the accuracy of MCE with respect to TSLSA.

7.2.2.2 Scoring System

The input record of DDSGA is similar to the one in Chapter 6:

$$\{X\ (U, S, P, F, M, T), y_1(r_1), y_2(r_2)... y_n(r_n)\}$$

Where,

- *X:* the session ID followed by its input parameters,
  - *U*: the current user id.
  - *S*: the login source IP address.
  - *P*: the login period,
  - *F*: a Boolean value to define whether there is a login failure in the session.
  - *M*: a Boolean value to define if the session has a masquerade or not.
  - *T*: the detection threshold for the session user.
- y_i: the input parameter i.e., a file or process name.
  - r_i: the return parameter of the executed pattern, is zero if the pattern was successfully and one otherwise.
  - *n*: the number of patterns in the session.

S, P, and F are the input of the neural network described in Section 7.4

As an example, a valid record is:

{253-VM1 (500, 172.16.12.1, ”Morning”, 0,0,0.62), \KnownDlls\user32.dll (0), \KnownDlls\ole32.dll (0), systray.exe(0), …..}.

DDSGA computes the detection score for each session and compares it against “*T*” to determine whether it has a masquerade patterns. Then, it compares this decision against “*M*” to compute the false alarms and hit ratio. Figure 7.9 shows how the scoring system in Chapter 6 is modified according to the extracted features.

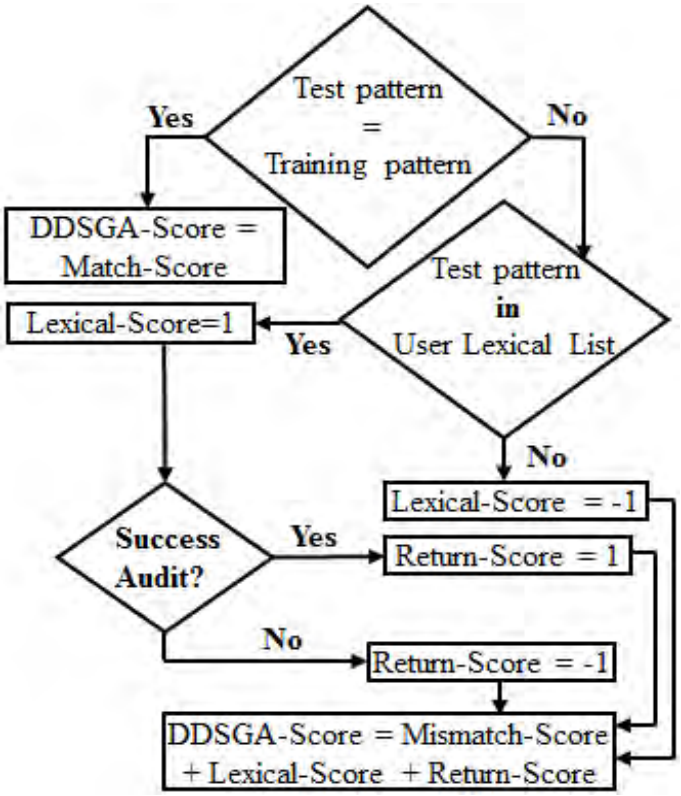


Figure 7.9: A flowchart for the modified DDSGA scoring system.

As in Chapter 6, the scoring system may reward a mismatched pattern in the test sequence to tolerate some permutations of previously observed patterns without reducing the detection score significantly. The next section highlights the security and the computational evaluation of the scoring system in the two correlation models.

7.2.2.3 The Independent and Centralized-Backup Models

With reference to distinct cloud deployment models, Chapter 6 has defined and evaluated three correlation models namely, Audit Exchange, Independent, and Centralized-Backup. In [142, 143] we have proved experimentally that the last two models are the most efficient ones. The first two models work with the original CIDS framework while the third one works with CIDS-VERT. This section evaluates the Independent and Centralized-Backup models using the testbeds described in Chapter 3.

In the Independent model, detection achieves a satisfactory efficiency and both the processing and network overheads are lower than those in the Centralized Backup model. This results in a lower detection time, see Figures 7.10, 7.12, and 7.13. As a counterpart, a masquerader survives for a longer time than in Centralized-Backup because of the time to check the audit data in all cloud nodes, see Figure 7.11. Furthermore, the Independent model does not scale to clouds with a large number of VMs and users.

Due to its centralized storage of user audits, the Centralized Backup achieves the best detection efficiency. Furthermore, no audit data is lost and the masquerader surviving is very short, see Figures 7.10 and 7.11. Figures 7.12 and 7.13 show that, both the network overhead and detection time are acceptable. As a counterpart, this is a resource intensive model that speeds up the detection phase and protects the IDS components from attackers by reserving some management VMs.

7.2.2.4 Evaluation of the Correlation Models

We focus our evaluation on local users because, as previously noticed, the large deviation in their behaviours reduces the detection accuracy. We consider the four main factors mentioned in Chapter 6 namely: (A) The accuracy and efficiency using both the ROC curve and Maxion-Townsend cost, (B) Average live time of a masquerader session, (C) Average transmitted data per detection session, and (D) Average detection time per session.

(A) The Accuracy and Efficiency

Using the same approach of Chapter 6, we evaluate the accuracy and efficiency of the two correlation models. We focus on the effects of the alignment parameters that the DDSGA computes for each user, namely the detection threshold, the scoring system rewards and the penalties on the accuracy parameters i.e., false positive, false negative rates, and the hit ratio. Figure 7.10 shows the ROC curves for the two models in the cases of the scoring system, the Centralized-Backup model without the scoring system, and the No-Correlation model with the scoring system.

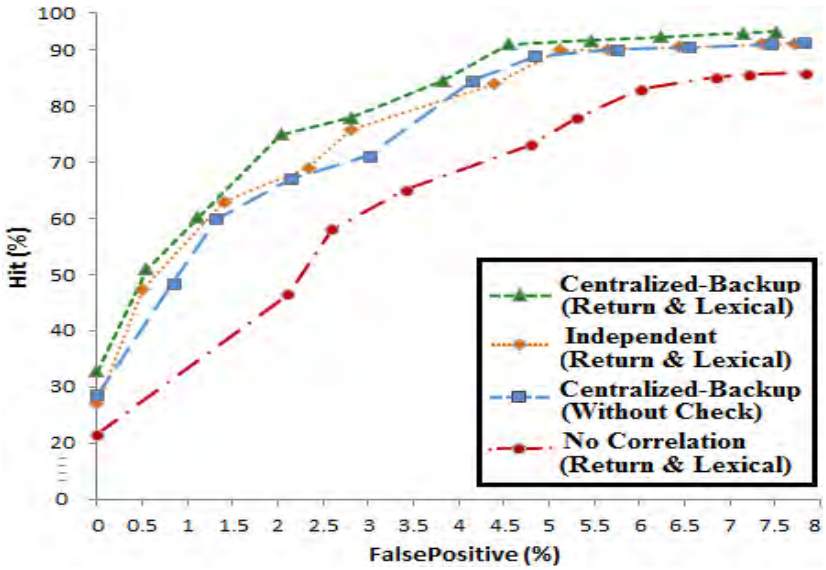


Figure 7.10: ROC curves for the three correlation models

Table 7.3: The best accuracy of the three correlation Models

Correlation Model	False Positive %	Hit %	Maxion-Town Cost
Centralized-Backup model with return and Lexical Check	**4.54**	**91.06**	**36.21**
Independent model with return and lexical check	5.11	90.03	40.63
Centralized-Backup model without the scoring system (without check)	4.84	88.92	40.15
No Correlation Centralized-Backup model with return and lexical check	6.01	82.99	53.07

As shown in Figure 7.10 and Table 7.3, the Centralized-Backup model with the scoring system results in the best hit ratio with the corresponding lowest false positive rates. The scoring system increases the hit ratio by about 2.14% and reduces Maxion-Townsend cost by 3.94. The correlation among audits of the same user is the key element of detection in cloud systems as it improves the hit ratio by about 8.07% and reduces the Maxion-Townsend cost by 16.86.

(B) Average Masquerader Live Time

We compute this time over all masquerade sessions for the two correlation models after determining in an experimental way the optimal number of management VMs in the Centralized Backup model. As shown in

Figure 7.11, this model achieves the shortest masquerader live time when using three management VMs.

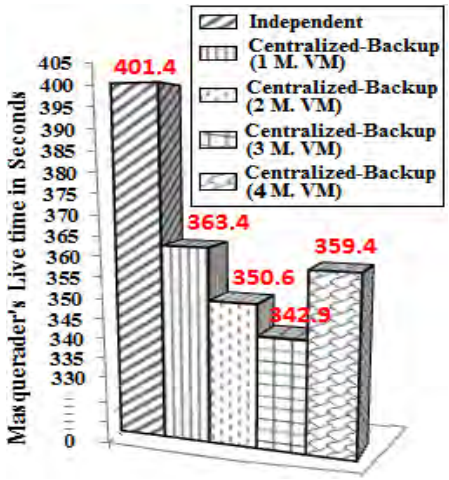


Figure 7.11: Average masquerader live time per session in the three correlation models

(C) Average Network Overload per session.

These experiments have evaluated the network overhead of the two correlation models in terms of the average data transmitted to analyze one session. According to the correlation model, the VM(s) that runs the detection task can send to the other VMs the user audits or the active session data. As shown in Figure 7.12, the Independent model is the lightest one. This figure also confirms that the network overhead of Centralized-Backup increases with the number of management VMs.

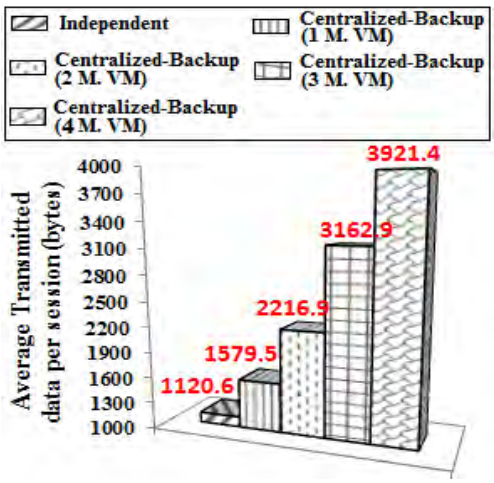


Figure 7.12: Average transmitted data per session in the three models

(D) Average detection time per session

The average DDSGA detection time is affected by the available processing resources, the size of the test session, the corresponding training sessions, and number of user VMs. Figure 7.13 shows that the average detection time of the Independent model when the user audits are distributed across three VMs depends upon NN, the number of cloud nodes running these VMs. In fact, the Independent model distributes the training audits across the nodes and each node independently runs the detection process using a few records. The detection time is minimized if the detection score in the first node is larger than the detection threshold, and this time increases as NN increases. The 3VMs audits label of the independent model in Figure 7.13 means that user audits are distributed across three nodes, each running one user VM. Therefore, we can compare the 3-VM columns against the other correlation models and notice that the Independent model achieves a noticeable improvement. This improvement is reduced as the NN increases to 3 and becomes larger than in all other models. Therefore, this correlation model is ideal with a small number of users and VMs and it cannot be adopted in large cloud such as public or hybrid ones. On the other hand, the Centralized-Backup model has a reasonable detection time that may be reduced by increasing the number of management VMs. Hence, the model may be adopted in large cloud because it is more elastic and scalable.

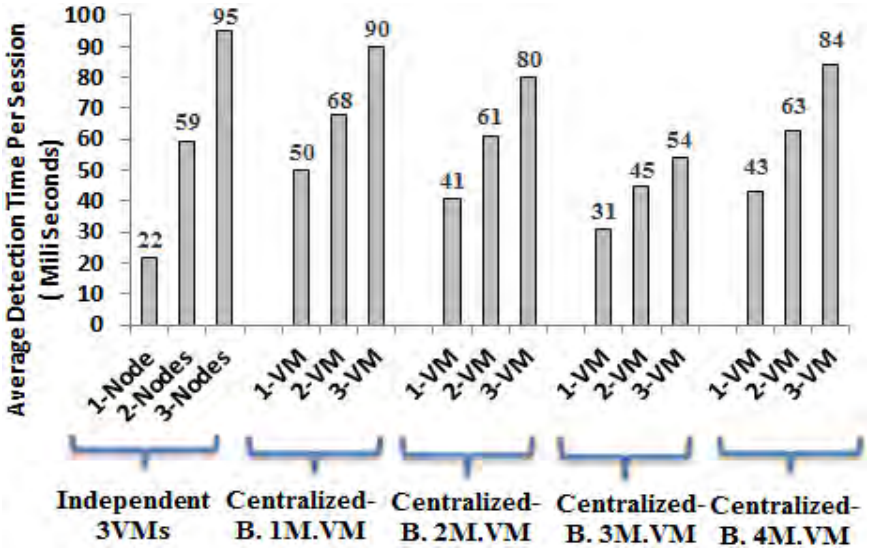


Figure 7.13: Average detection time per session in the three models

7.3 Detecting Masquerade Based on NetFlow Data Analysis

Chapter 6 outlined the advantages of network centric approaches and proposed three approaches to capture the NetFlow data of interest. We describe how these approaches are applied to the cases of interest.

7.3.1 Feature Extraction from NetFlow data.

Chapter 6 has described how both CIDS and CIDS-VERT frameworks use the correlator components to define the beginning and the end of the user sessions outside the local system and to filter the NetFlow data corresponding to the user host session according to the source IP address. In the case of Windows audits, the distribution of NetFlow data for each of three CIDD user categories supports the same conclusion in Chapter 6 about the NetFlow features as summarized below:

- System and server user sessions are more regular than those of local users because their destination IP addresses are more specific and consistent.
- NetFlow data are less regular than security events, because user network activities have a lower consistency than user host ones. The sharing of the source IP among several users may further decrease data consistency and regularity in the network profile together with the accuracy of detection.

The DDSGA scoring system takes into account all these features as detailed in both [143] and Section 6.3.2.

7.3.2 The NetFlow Scoring System Evaluation

We have evaluated the DDSGA approach through the NetFlow data corresponding to the sessions of each source IP in CIDD as in Section 7.2.2.2 using the Centralized-Backup model of the CIDS-VERT framework.

Figure 7.14 and Table 7.4 show the detection accuracy of DDSGA over the NetFlow data in terms of both the ROC curve and Maxion-Townsend cost.

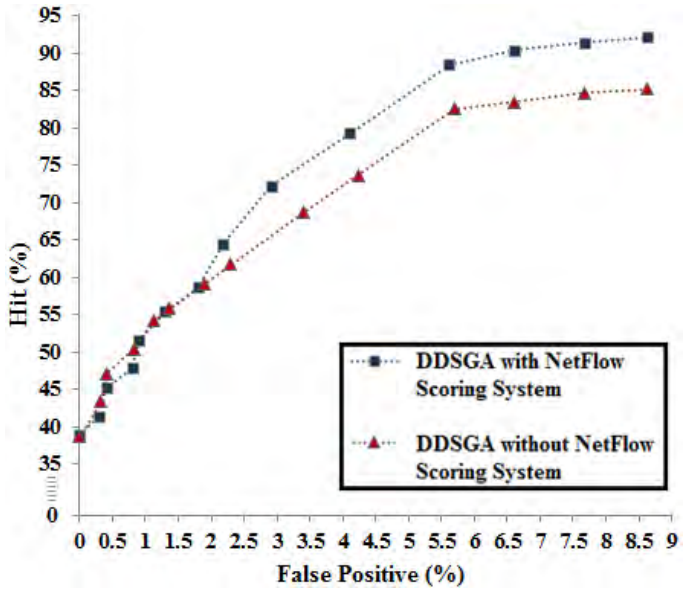


Figure 7.14: ROC curve for DDSGA on the NetFlow audits

Table 7.4: The best accuracy of the DDSGA approach on the NetFlow audits

Approach	False Positive %	Hit %	Maxion-Town Cost
DDSGA with NetFlow scoring system	**5.61**	**88.41**	**45.25**
DDSGA without NetFlow scoring system	5.7	82.51	51.69

As shown in Figure 7.14 and Table 7.4, the NetFlow scoring system improves the hit ratio by 5.9% and reduces Maxion-Townsend cost by 6.44.

7.4 Integrating Host and Network Detections using A Neural Network Model

To improve the accuracy and efficiency of detection, we implement an integrated approach that uses the neural network model of Chapter 6 and considers both host and NetFlow audits. The detection and training modes of the neural network model are adjusted as in Sections 6.4.1 and 6.4.2 respectively.

We compare the accuracy and the average detection time per session of the three detection approaches: the host based, the network based, and the integrated one, i.e. the neural network, using the Centralized-Backup model with three management VMs. As shown in Figure 7.15 and Table 7.5, the integrated approach results in the highest accuracy. As counterpart, Figure

7.16 confirms that it also results in the longest average detection time per session because it needs the outputs of the other approaches.

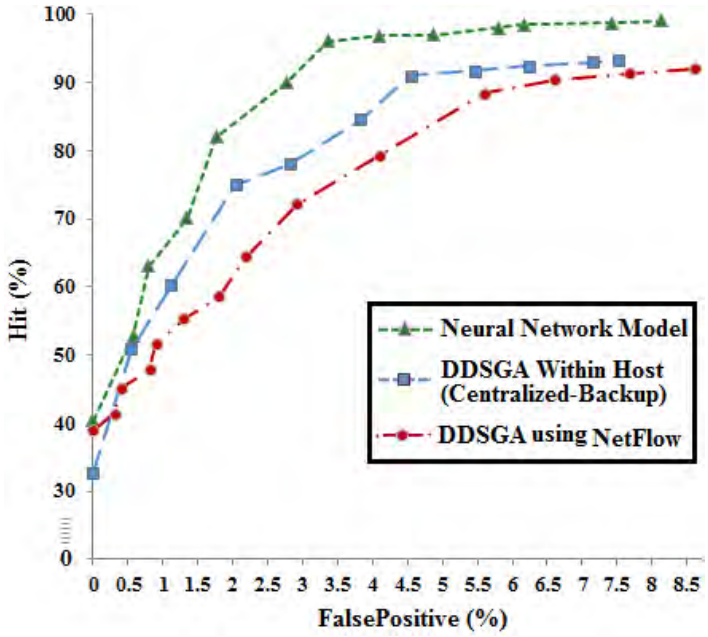


Figure 7.15: The ROC curve for the three approaches

Table 7.5: The best accuracy of the three detection approaches

Approach	False Positive %	Hit %	Maxion-Town Cost
Neural Network Model	**3.35**	**96.08**	**24.02**
DDSGA within Host system	4.54	91.06	36.18
DDSGA using NetFlow	5.61	88.41	45.25

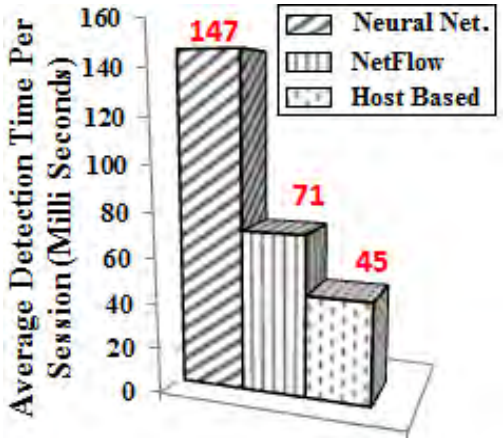


Figure 7.16: Average detection time per session for the three approaches.

7.5. A Comparison between the Two Detection Approaches

In the following, we compare our proposed masquerade detection solution, denoted as Security Events (SE) and the one we introduced in Chapter 6 and that is based on the integration between the System Calls (SC) and the NetFlow. Since both SE and SC are among the most important audit sources to build a user profile to detect several kinds of attacks, a comparison can highlight some important issues in these approaches. In particular, we are interested in how parameters such as the number of training records and the consistency of both system calls and security events audit influence, among others, the detection accuracy and time, the masquerade live time, and the related overheads. Table 7.6 highlights the comparison between the proposed detection approaches through SE and SC for the local user category and using the Centralized-Backup model of CIDS-VERT framework.

Table 7.6: Masquerade detection through system calls and security events

Accuracy (ROC Curve)					
Approach			False Positive	HIT	Maxion-Cost
Neural Network (SC)			**1.5948**	**98.07**	**11.5**
Centralized-Backup (SC)			2.32	94.24	19.68
NetFlow (SC)			6.05	83.0464	53.254
Neural Network (SE)			**3.35**	**96.08**	**24.02**
Centralized-Backup (SE)			4.54	91.06	36.18
NetFlow (SE)			5.61	88.41	45.25
Average Detection Time(Milli Seconds)					
Neural Network (SC)	116	NetFlow (SC)	59	Host(2 M. VMs) (SC)	39
Neural Network (SE)	147	NetFlow (SE)	71	Host(3 M. VMs) (SE)	45
The Conditional Entropy and SWS for Host Audits					
Entropy (SC)	5.4	SWS (SC)	12		
Entropy (SE)	7.2	SWS (SE)	44		
Average Masquerader Live Time(Sesonds) in Host Audits					
SC (2 M. VMs)	287.34				
SE (3 M. VMs)	342.9				
Average Transmitted Data Per Session (Bytes)					
SC (2 M. VMs)	1961.9				
SE (3 M. VMs)	3162.9				

Table 7.6 shows that the SC audits results in a better accuracy than the SE one. The reason is that SC audits have large number of training records, 6 weeks of audits in CIDD. This helps in training both the update phase of

DDSGA and the neural network model. Instead, the SE audits have only 2 weeks of training audits. The analysis of the SC audits results in a better accuracy than the SE one, because SC audits reflect all system activities. Furthermore, they provide some basic statistical information about user network activities that can be integrated with information on the user behaviours in the NetFlow audits. In turn, this simplifies the integration process. Furthermore, as shown by the conditional entropy in Table 7.6, SC audits are more regular than the SE one. This helps in recognizing a user normal behaviour and improves the accuracy of detection.

Table 7.6 also shows that the session length corresponding to the SWS value is directly proportional to masquerade live time, network overhead, number of management VMs, and detection time. The SC audits have shorter session length than the SE one and, consequently, they may reduce masquerade live time, network overhead, management VMs, and detection time. Both NetFlow audits and the statistical information of the SE sessions are more consistent than the corresponding ones of the SC. This increases the accuracy of detection in the NetFlow of SE audits by 5.36% with respect to that of SC audits. As a result, the hit ratio of the neural network model increases from its corresponding host ratio by 5.02%, with respect to 3.83% of the SE. The False Positive rate is reduced by 1.19%, with respect to 0.72% of the SE. However, the SC NetFlow audits contain more training records than the SE one, but the consistency of these audits is much lower than in the SE NetFlow audits. This is due to the number of users that share the same IP addresses that is much larger in the SC NetFlow audits than in the SE NetFlow audits.

Chapter 8

Efficient IDS Deployments through a Hierarchical Architecture

This chapter focuses on signature based analysis techniques that, in general may result in a best accuracy than behaviour based ones. The chapter introduces a hierarchical architecture [CIDS-2Deployments] of CIDS-VERT framework that supports two deployments, Distributed and Centralized, and it outlines their architectures, components, and relative advantages. Furthermore, it discusses how to correlate and summarize distinct HIDS and NIDS alerts. Finally, the chapter experimentally evaluates the accuracy of the proposed deployments to confirm the improvements with respect to current IDSs.

8.1 The Hierarchical Architecture of our Cloud IDS

As discussed in Section 1.9, a cloud defense strategy has to satisfy some further requirements with respect to traditional ones. In particular, it should:

(1) Be distributed and scalable.

(2) Avoid single points of failure.

(3) Correlate the user behaviours in distinct environments.

(4) Integrate different service models.

This section briefly outlines the hierarchical structure of the proposed cloud IDS [CIDS-2Deployments] together with its implementation models. The IDS can detect attacks in several classes, e.g. masquerade, host, network, and DDoS against all cloud models through signature based techniques and behavior based ones.

This chapter evaluates the detection accuracy of the IDS using two deployment models, Centralized and Distributed, based on the CIDS-VERT framework that provides high scalability and resilience for large clouds.

Figure.8.1: The hierarchical architecture of the proposed cloud IDS

Figure 8.1 shows the hierarchical architecture of the proposed IDS. The six layers are described in the following:

1) Infrastructure Layer:

It defines the cloud physical specifications. In the considered case, the CID-VERT testbed consists of an HP C3000 Cloud blade with six nodes. One head node works as a front side interface 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 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. CID-VERT 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:

Besides managing the virtual VMs, virtual switches, and virtual SAN driver, it provides system security functions such as Isolation, Inspection, and Interposition. The IDS can support different frameworks such as VMware cloud [56], Microsoft private cloud [13], Open Stack [147], Eucalyptus [12]. In the 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 maps the VMs onto the physical cores. To provide a full heterogeneous environment, each testbed node hosts 3 VMs that runs, respectively, Windows XP Professional SP3, UNIX (Solaris) and Linux (Centos). Each VM is assigned one core of the Quad core and 3 GB RAM. Each VM runs a HIDS sensor and an event collector component to collect events and logs from the VM operating system and forward them to a centralized management VM to analyze them through DDSGA [DDSGA] and HIDS analyzers. Some VMs run a NIDS component as a sensor or a server based on the applied deployment. One VM runs the CPU Death Ping [50], LOIC [49], and the Metasploit [46] library. Section 8.4.1 explains the attack scenario in our experiments.

4) Intrusion Detection Layer:

This layer runs the main three IDS components: the HIDS, the NIDS and DDSGA. In the experiments, the HIDS is OSSEC [61] and the NIDS is Snort. This layer also defines the Centralized deployment model and the Distributed one. Section 8.2 outlines the deployments of all the IDS components in the VMs.

5) Alert Integration and Reporting Layer:

It integrates and correlates the alerts from host and network IDSs. To provide a standard, coherent representation of alerts and describe the relationship between simple and complex alerts, it uses the IDMEF Message format [76] described in Section 2.1.3. To simplify the handling of attacks, it also highlights the critical alerts. Sections 8.3.1 and 8.3.2 detail the integration and correlation processes.

6) Web Interface Layer:

It offers a central location to admin and to manage the IDS components, as it runs the management VM and it handles the web pages to visualize the IDS charts and dashboards, see Section 8.4.3.

8.2 THE DISTRIBUTED AND CENTRALIZED DEPLOYMENTS

To evaluate our IDS in a realistic setting, we have partitioned the cloud into two virtual zones, VZ_1 and VZ_2, to distribute the DDoS Zombies into two distinct virtual cloud networks. The VMs are connected to virtual switches through virtual NIC cards, see Figures 8.2 and 8.4. In turn, virtual switches are connected to physical switch ports through the port mirroring facilities of the Hypervisor layer. Zone VZ_1 includes $node_0$ and $node_2$, each running three VMs with distinct operating systems. A further VM on $node_2$ runs the Metasploit and LOIC attack libraries. Zone VZ_2 includes $node_1$, $node_3$, and $node_4$, each hosting 3 VMs as in VZ_1. In the following, we describe the two proposed deployments and outline in Section 8.4 the experiments using the two deployment options.

8.2.1 The Distributed Deployment

As implied by its name, this model distributes the detection overhead among several cloud VMs. The final decision correlates the outputs of the IDS sensors in these VMs.

In each virtual zone, the 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 events from the VM operating system and forwards them to the server component in a VM in the zone. Agents are installed in all VMs except those running the HIDS servers. An HIDS server analyzes the 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. 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 network attacks. To avoid a single point of failure, the VM that hosts the HIDS and NIDS servers is backed up by a VM in another node in the same zone. This VM acts as a hot spare of the active one because a copy of the status of the active server is updated in the backup VM through

heartbeat messages. CIDS-VERT framework in Chapter 3 refers to the "Management VMs" that runs the NIDS and HIDS servers and its corresponding backup VMs.

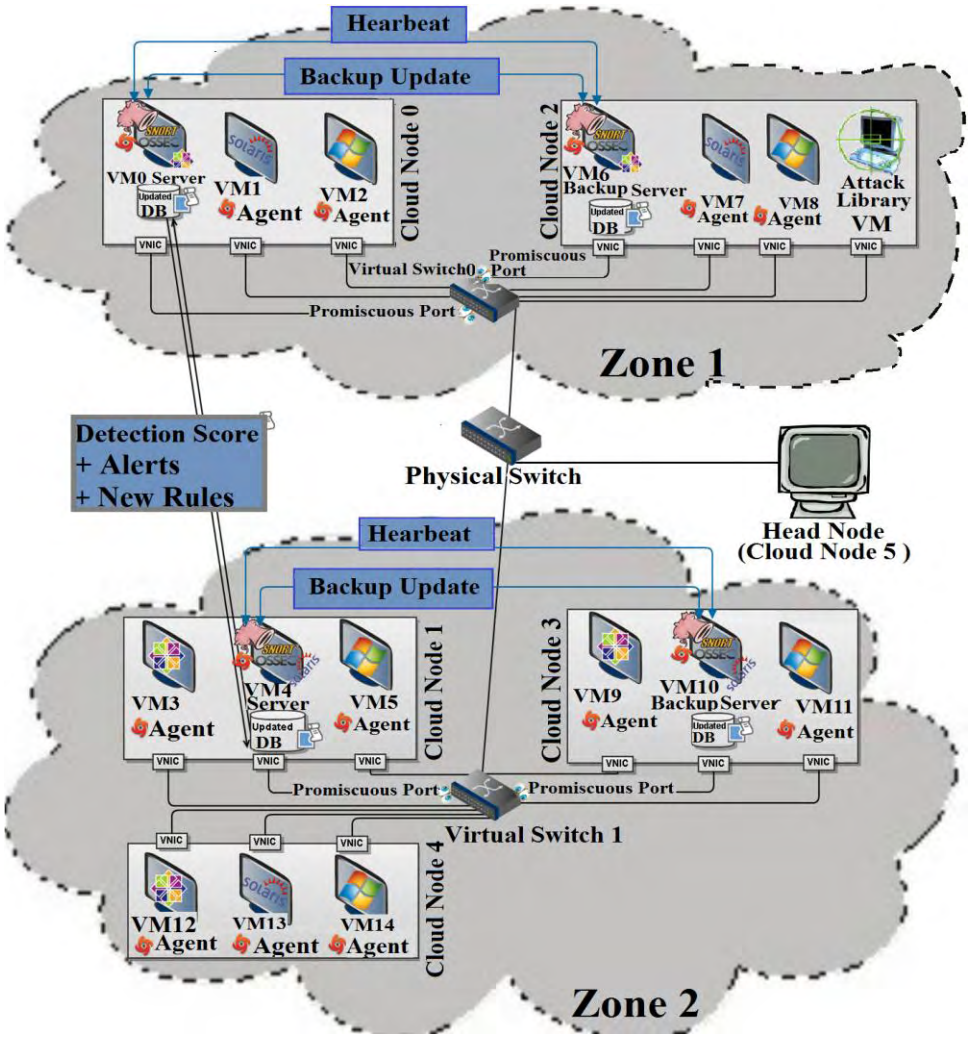


Figure 8.2: The distributed deployment

As shown in Figure 8.2, VM_0, hosted in cloud $node_0$ of VZ_1, is the active management VM that runs both the Snort and the OSSEC servers and it is backed up to VM_6 in $node_2$. The Snort server is connected 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, VM_4, hosted in $node_1$ of VZ_2, runs the OSSEC and Snort servers and is backed up by VM_{10} hosted in $node_2$. The signature databases in both VM_0 and VM_4 are

simultaneously updated and the two VMs exchange the notification alarms. Furthermore, the final detection score correlates the scores of each Snort server.

8.2.1.1 DDoS Detection scenario using the Distributed Deployment

In an attack, several VMs in distinct virtual cloud zones may behave as DDoS zombies and generate a stream of packets towards 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 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 scores, the notification alerts, and the new signature rules. An update of the signature database as in [49] may enable Snort to detect the DDoS attacks by the LOIC and CPU Death Ping libraries. As an example, the following rules detect specific behaviors for each protocol [49]:

(a) The UDP traffic at specific ports is analyzed and the number of opened connections in a short time interval is compared against a threshold. While the UDP protocol is stateless, it is possible to define and track a UDP connection according to the change in the timeout field of each UDP packet [160, 49].
(b) The TCP rule checks whether an ACK TCP flag is set and check the packets size.
(c) 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 decision, integrates the scores as in Equation 8.1.

$$VS = \begin{cases} 1, & V(a) > 1/2 \\ F, & V(a) = 1/2 \\ 0, & V(a) < 1/2 \end{cases} \quad \text{..... (8.1)}$$

Where:

- *V(a)*: Voting of alert *a*. It is the percentage of IDSs that have sent this alert.
- *F*: A flag to denote if *c*, the number of packets of a given type, a host may send in a given interval, is larger than a threshold *t*,

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

Figure 8.3 shows the DDoS detection scenario in this deployment.

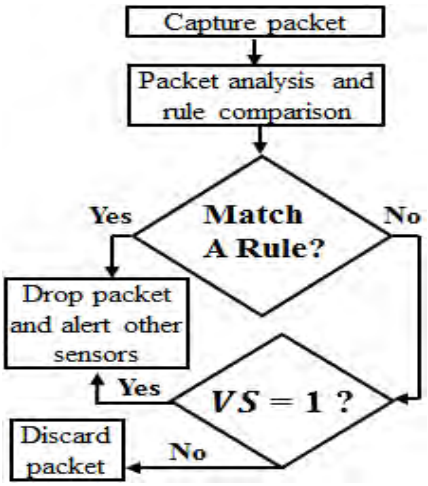


Figure 8.3: The DDoS detection flowchart in distributed deployment

8.2.1.2 Evaluation of the Distributed Deployment

This deployment:

(a) Distributes the computational overhead among several cloud VMs.

(b) Reduces the network overhead because it does not forward events to a central location.

(c) Avoid a single point of failure due to backup VM.

As a counterpart, the integration of the outputs of several IDSs increases the detection time and reduces the accuracy with respect to a centralized deployment.

8.2.2 The Centralized Deployment

It uses the same components of the other model and it does not change the HIDS functionalities. Instead, it forwards all network packets to a centralized database to be analyzed by one Snort server. This changes both how the servers analyze and collect the packets and the handling of backups for both the centralized database and the Snort server. As far as concerns the first change, we recall that Snort can run in three modes: sniffer, packet logger, and detection. In the distributed deployment, all Snort servers run in the detection mode. Instead, in the centralized deployment, the servers run in packet logger mode and log the packets to a centralized database. The VM hosting this database is backed up by another VM in the other zone to avoid

a single point of failure. Each server is connected to the central database and to the backup VM and it runs in the detection mode to match the packets against Snort rules. Each server communicates its alerts to those in the other zone.

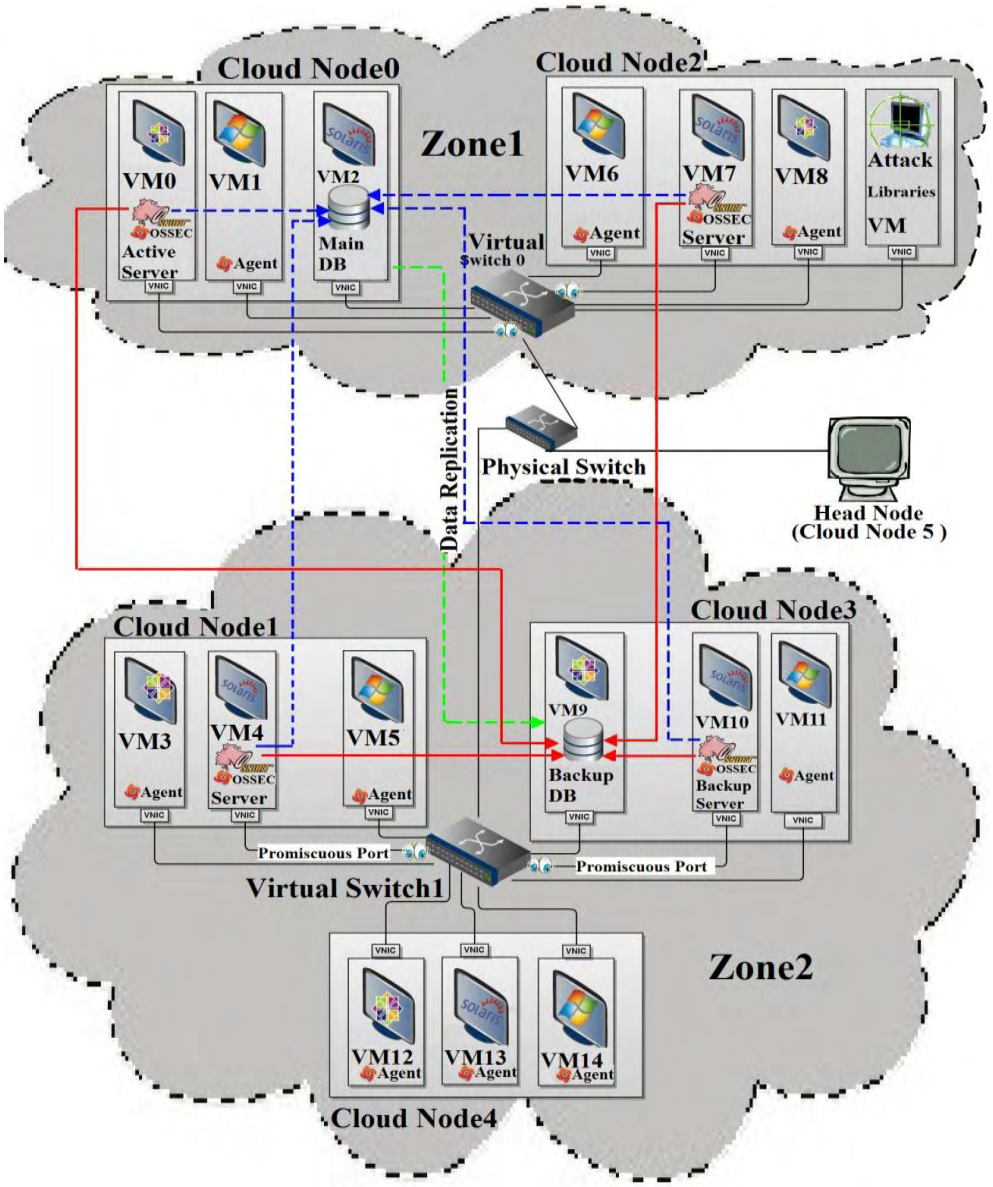


Figure 8.4: The centralized deployment

As shown in Figure 8.4, VMs 2 and 9 host, respectively, the centralized database and the backup one. VM_0 runs the active Snort server in VZ_1 while

VM_7 runs the backup one. In the same way, VM_4 runs the active Snort server in VZ_2 and VM_{10} runs the backup one. Just one server analyzes the packets in the database and exchanges the heartbeat messages with its backup. The dashed and the solid lines in the figure connect the IDSs to, respectively, the active centralized database and the backup one.

8.2.2.1 DDoS Detection scenario using the Distributed Deployment

The centralized deployment can easily detect DDoS attacks, because it collects in a single location the packets and network events to match them against rules in the central database. This improves the detection accuracy and enables the administrator to monitor the cloud from a central management VM. Figure 8.5 shows the DDoS attack detection scenario in the centralized deployment option.

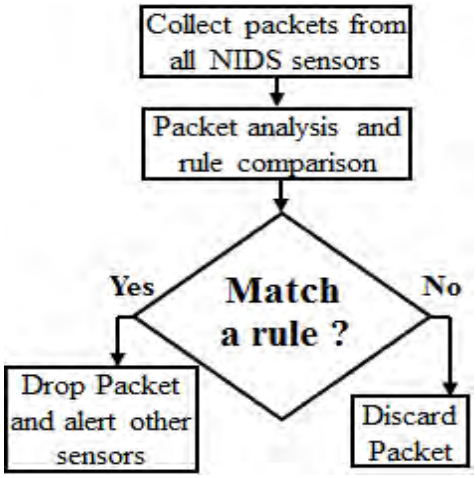


Figure 8.5: The DDoS detection flowchart in centralized deployment

8.2.2.2 Evaluation of the Centralized deployment

This deployment is characterized by a central analysis that reduces the detection time and increases accuracy. Furthermore, backup VMs increase the overall reliability. The counterpart is the large overhead due to both the central VM and the cloud network that forwards all packets to this VM.

8.3 ALERTS INTEGRATION, CORRELATION, AND RISK ASSESSMENT

8.3.1 Alerts Integration:

This layer collects alerts from several detectors and integrates them through a normalization process and a prioritization one.

– **The normalization process:**

It formats any detector alert into the IDMEF protocol to simplify their analysis and correlation in the next layer. To this purpose, it extracts information from the alert fields with different names and data formats and represents this information in a consistent and common format. Further information may be added to the normalized alert based on details on the data source or on 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:**

To handle the prioritization systems of distinct detectors, this process maps alert priorities into a single range from 0 to *n*, where *n* is defined by system administrators. Consider, as an example, that Snort alerts have a maximum priority of 3 while OSSEC alerts have maximum priority of 12.

The following examples 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">
```

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" type="string">5</AdditionalData> </Alert>
</IDMEF-Message>
```

8.3.2 Alerts Correlation and Summarization:

It correlates a large number of normalized alerts from different detectors to highlight the few critical ones. It looks for evidences of an alert to discover if it signals a true attack and then it correlates the 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 [161] that consists of several steps by the same attacker. The correlation process:

(a) Reduces false positives alerts.

(b) Summarizes the huge number of alerts to the cloud administrator.

(c) Deals efficiently with multi-stages attacks.

The correlation engine is implemented by OSSIM [62] described in Section 1.10. OSSIM uses a tree of logical conditions (rules) or AND/OR tree, see Figure 8.6.

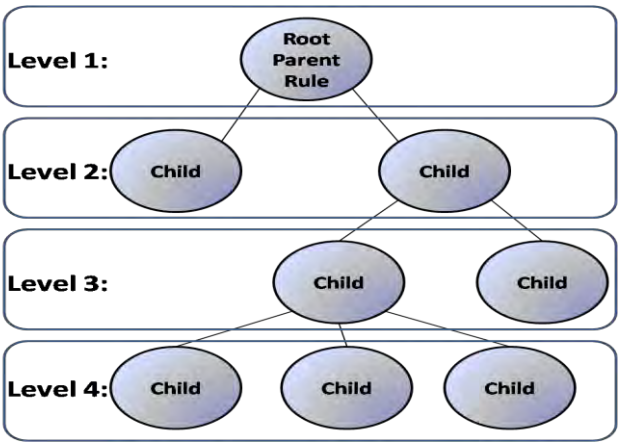


Figure 8.6: An example for a correlation tree

The correlation stops if the root parent rule at level 1 is not matched. Otherwise, the engine considers the various levels 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, it computes a reliability value in each level to determine the final risk as detailed in Section 8.3.3.

We show now two correlation examples. In the first one, the correlation helps to detect a brute force attack against an SSH Server [62]. Here, the alerts are produced by several instances of the same analyzer, i.e., Snort. In

the second example, distinct analyzers, i.e., OSSEC and Snort, produce the alerts and the correlation helps to detect the Reverse Shell attack.

– Example 1: The engine builds a four levels tree to detect the brute force attack against an SSH Server, see Figure 8.7. As an example of the childs of the tree, we show the rule that 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 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 matching rule conditions.
- time_out: Waiting time before the rule expires.
- from and to : Source IP and Destination IP.
- sensor: the firing sensor of the event.
- Reliability: is used to compute the risk value, see Section 8.3.3.

Figure 8.7 shows the four levels of the correlation tree:

- **Level 1:** A root rule that will be matched by an authentication failed alert. After updating the reliability value and computing the risk, the correlation engine jumps to the next level.
- **Level 2:** Two rules with two possible actions. The left rule is matched by a successful authentication alert. The correlation engine updates the reliability value, computes the risk, and then stops. Instead, the second right child rule is matched by the reception of ten authentication failure alerts. The engine updates the reliability value and computes the risk and jumps to level 3. Then, it will evaluate both level 3 and 4 in the same way and finally fires an alarm.

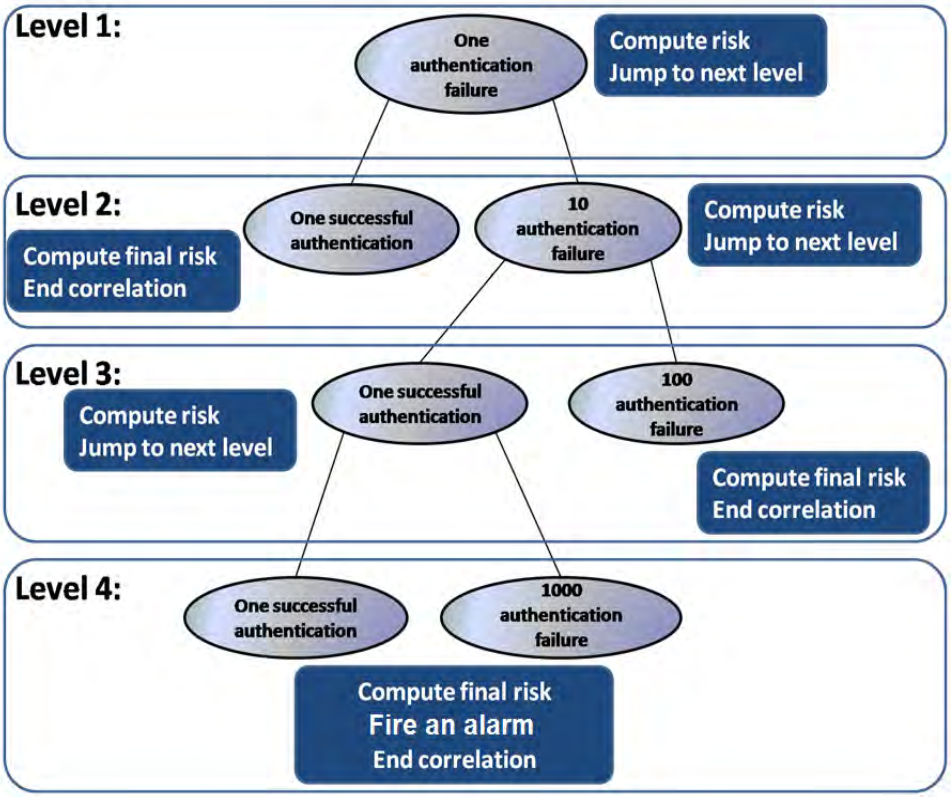


Figure 8.7: Four levels of the correlation tree to detect a brute force attack.

– Example 2: To detect the Reverse Shell attack, the engine correlates the alerts from both OSSEC and Snort. This is a multi-stages attack implemented as in the scenario in Figure 8.8 [161]:

Figure 8.8: The Reverse Shell attack scenario

In this scenario, the appropriate IDS, i.e., OSSEC or Snort, fires an alert for each step of the attack. While an analysis of individual alerts may be

useless, their correlation conveys useful information. The engine applies both the normalization and prioritization processes. The final correlation tree has four levels:

- **Level 1 root rule:** This rule is matched by scanning 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. After updating the reliability value and computing the risk, the engine jumps to level 3.
- **Level 3:** This rule is matched by a file uploading alert from OSSEC. After updating the reliability value and computing the risk, the engine jumps to level 4**.**
- **Level 4:** This rule is matched by a Snort alert that denotes the activation and access shell using reverse TCP. After updating the reliability and risk values, the engine fires an alarm.

8.3.3 Risk Assessment:

This value estimates the risk for the cloud asset based on the alerts that have been fired. It is computed at each correlation level through Equation 8.2:

$$RISK = (Asset * Priority * Reliability)/NF \quad (8.2)$$

Where:

- Asset denotes the value of the resource under attack and it ranges from 0 to *A*, the maximum for the assessment. The user set this value when configuring the IDS.
- 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 it is prioritized as in Section 8.3.1.
- Reliability (0-*R*) is the probability that the attack defined in a correlation level is real. It changes in each level as detailed in section 8.3.2.
- NF is a normalized factor based on *A*, *P*, *R* and maximum risk value (*M*), where $NF = (A * P * R) / M$

Risk Assessment Methodology:

Risk is computed when matching the alerts from each IDS in the cloud against the rules in each level. The computation is repeated at each 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 value is at least one, an alarm will be fired. Figure 8.9 shows the correlation and risk assessment processes with *N* correlation levels. Each level has different number of rules.

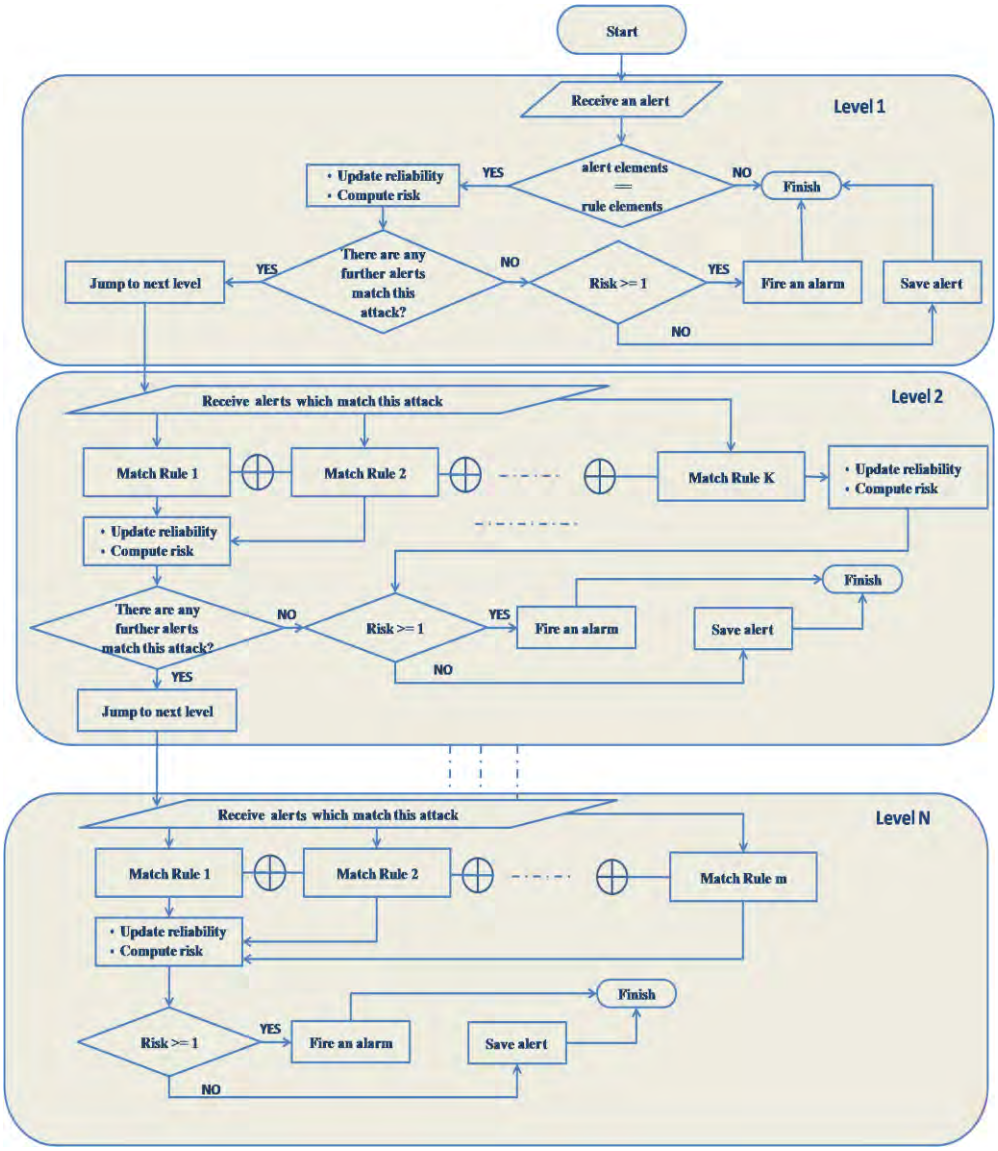


Figure 8.9: The correlation and risk assessment flowchart

To show how each correlation level computes the risk value, we consider the tree in Example 1 of Section 8.3.2 to detect the "SSH Brute Force" attack. [62]:

First Level: This rule is matched by 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"/>
```

A rule is matched if all the elements of an alert match those of the current rule. 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: These rules are matched if 11 SSH authentication failure alerts are received in less than 40 seconds. The first alert matches one rule in this level and the other 10 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="a" 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" plugin_sid="1,2,3,6,9"sticky="true"/>
```

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. If no further alerts that satisfy this attack are received, the correlation process ends and the engine fires an alarm 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 and risk values to, respectively, 4 and (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 is fired because the risk is larger than one.

8.4. EXPERIMENTAL RESULTS

8.4.1 Attack Scenario

To evaluate the detection accuracy of the proposed IDS, we run an attack scenario through the Metasploit library. We also consider a DDoS attacks scenario that uses both the LOIC and CPU death ping libraries independently.

Figure 8.10 explains these scenarios where the Metasploit library installed in VZ_1 attacks VM_6 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 VM_8 in VZ_1 and in VMs 5, 11, and 14 in VZ_2. Each agent attacks one VM in each zone. Consequently, the agents of VM_8 attacks VM_6 and VM_3, and the agents of VM_5 attack VM_3 and VM_6. The agents of VM_{11} attack VM_{12} and the VM with the Metasploit library. Finally, the agents of VM_{14} attack VM_{12} 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.

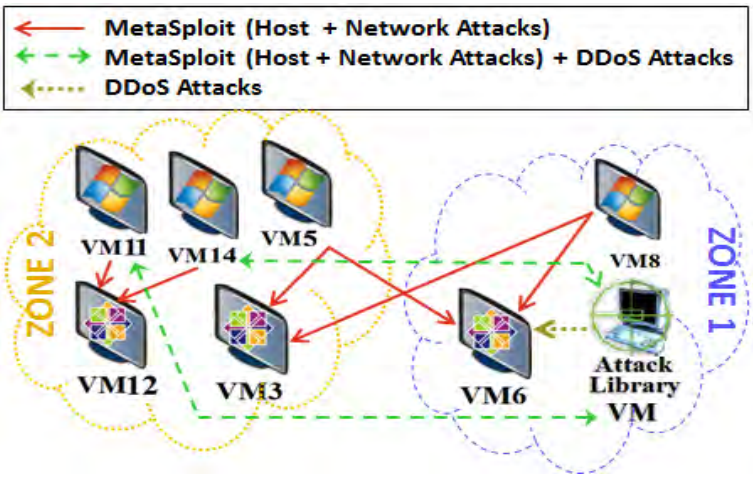


Figure 8.10: The host, network, and DDoS attacks scenarios

8.4.2 Performance Evaluation for the Two Deployments

This section evaluates the proposed IDS by analyzing the traffic of each Snort sensor in the Centralized VMs, VZ_1, and VZ_2. Furthermore, it evaluates the accuracy and the computational performance of the Centralized

and Distributed deployments. Figure 8.11 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 8.12 shows the spikes of the DDOS attacks due to 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 splitted into four parts to show the network traffic 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 can no longer respond to its users. In part C, the IDS starts to handle the attack and blocks the illegal packets. Finally, in part D, the traffic rate returns to normal again.

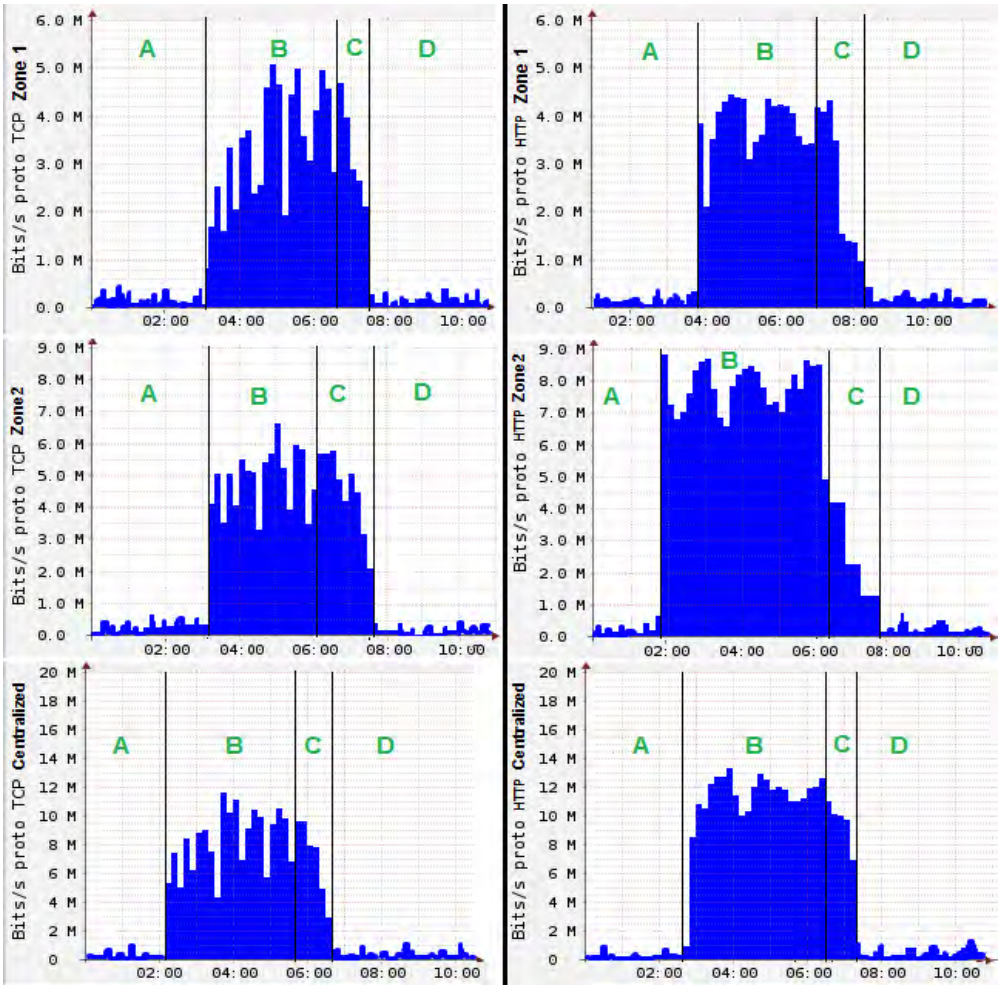


Figure 8.11: The DDOS by TCP and HTTP floods

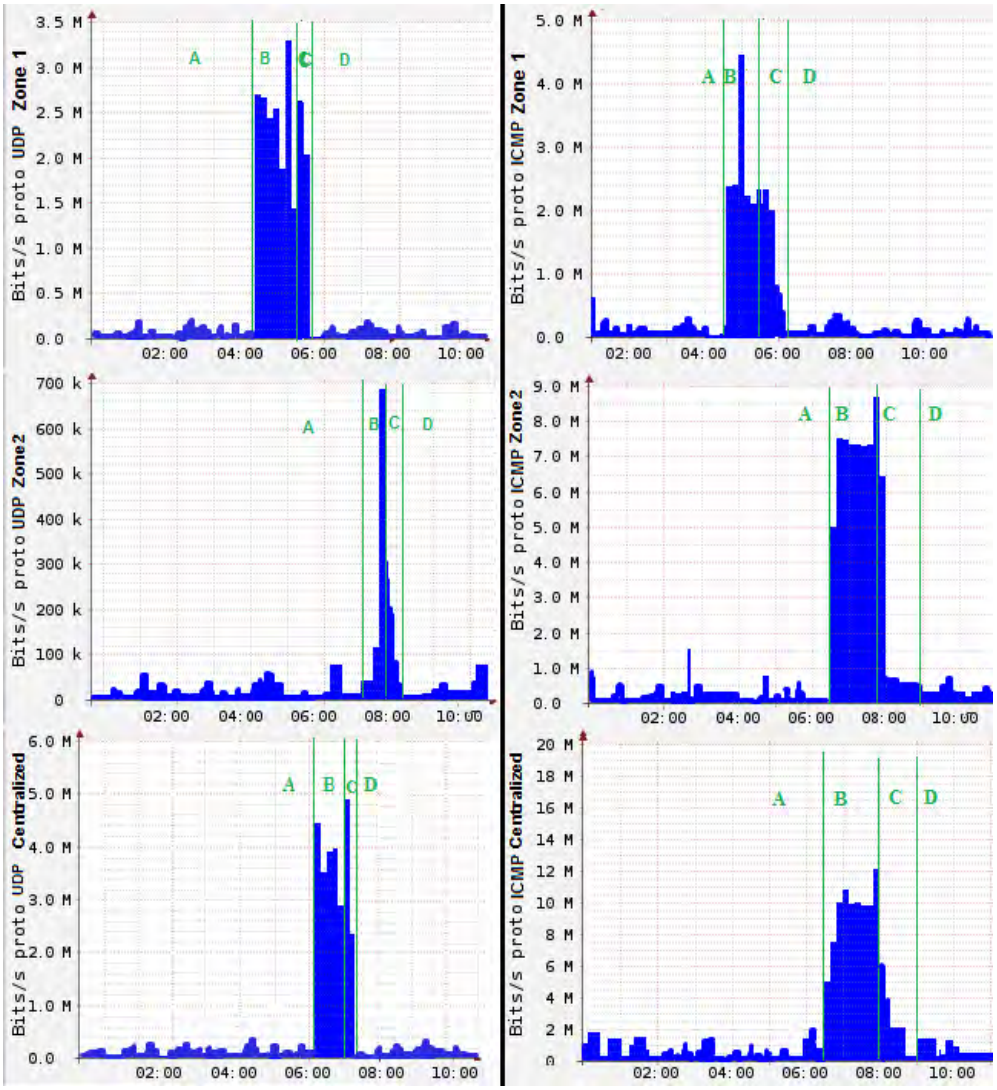


Figure 8.12: The DDoS by UDP and ICMP floods

The two deployments are compared in terms of detection accuracy and computation time over 20,000 data packets. Figure 8.13 shows that the Centralized deployment signals a higher number, 34.3%, of true alerts than the Distributed one. This is due to the centralized decision on detection. However, Figure 8.14 shows that the Distributed deployment has a lower computation time, 28.8%, than the Centralized one because it distributes the detection overhead among several sensors and drops any packet that matches a rule without any further analysis.

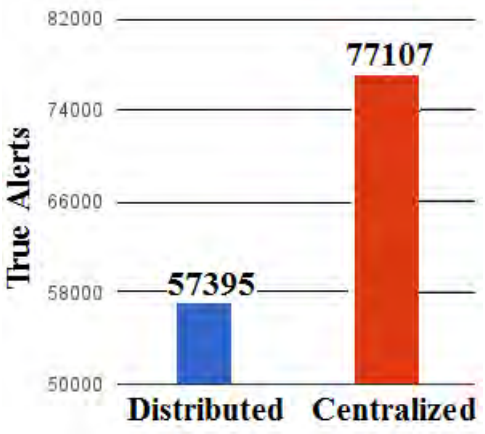


Figure 8.13: Number of true alerts

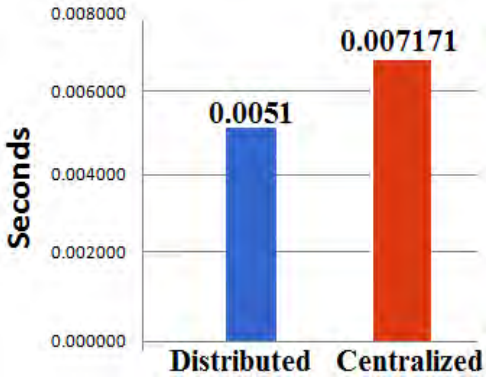


Figure 8.14: The computation time

8.4.3 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 8.15 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 8.3.1 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 [169 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 [769 events]	3	OSSEC	2012-09-04 11:41:00	2012-09-04 11:51:17	192.168.137.223:ANY	192.168.137.210	open	

Figure 8.15: A snapshot of detected host and network attacks

Figure 8.16 shows the top 5 alerts with high risk value fired by both OSSEC and Snort IDSs in distinct cloud locations. Figure 8.17 shows the top 10 VMs that signaled multiple alerts.

Figure 8.16: The top 5 alerts with high risk value fired by OSSEC and Snort.

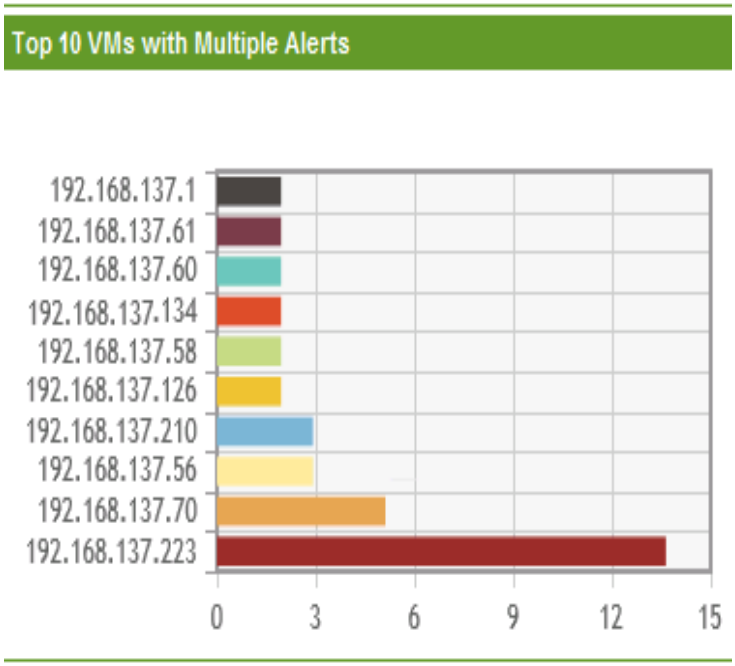


Figure 8.17: The top 10 VMs with multiple alerts in the cloud system.

Conclusion

Cloud Computing is a new term that denotes the use of IT services and resources that are accessed on a service basis provided by enterprises and that their users access via the internet.

Even if there is a large consensus on the benefits of cloud computing, concerns are being raised about the security issues introduced through the adoption of this model and to the lack of control by the cloud users on some architectural levels. The effectiveness and efficiency of traditional protection mechanisms are being reconsidered as the characteristics of this innovative computing model and the control on shared resources widely differ from those of traditional architectures. Cloud computing environments are easy targets for intruders and pose new risks and threats to an organization because of their service and operational models, the underlying technologies, and their distributed nature. In particular, some kind of sharing is intrinsic to cloud computing and cannot be avoided. In turns, this blurs the traditional distinction between private and shared resources.

In principle, IDSs are among the efficient security mechanisms that can handle most of the threats of cloud computing. However, several deficiencies of current IDSs technologies and solutions hinder their adoption in a cloud.

This thesis has proposed and developed a cloud based intrusion detection system that satisfies the cloud requirements and deals with several classes of attacks against all cloud deployment models.

The architecture of the proposed IDS is fully distributed to provide a scalable and elastic solution and avoid a single point of failure. Furthermore, the IDS isolates the user tasks from cloud nodes and achieves a high coverage of attacks by integrating both knowledge and behaviour based techniques. The IDS adapts with distinct cloud computing environments and it collects and correlates the user behaviours from the cloud VMs and integrates the alerts from different IDSs into a single report.

We have introduced two frameworks of the proposed IDS to support distinct cloud deployment models namely, the Cloud based Intrusion Detection System, CIDS, and its full virtual version, CIDS-VERT. CIDS P2P architecture hinders scalability but it achieves a high performance and

low network overhead in small or private clouds. Instead, the better scalability and controllability of CIDS-VERT makes it the ideal solution for hybrid and public clouds. Furthermore, the management and the configuration of CIDS–VERT are rather simpler than those of CIDS.

To efficiently correlate the behaviour of the same user in distinct cloud nodes, we have proposed, developed and evaluated three alternative models that define the exchange of audit data and of alerts between the IDS components. The first two models, Audit Exchange and Independent, work with CIDS framework, while the third, the Centralized-Backup works with CIDS-VERT.

Three are the main contributions of this thesis, namely:

(1) CIDD, a cloud intrusion detection dataset.
(2) The behaviour based detection.
(3) The signature based detection

The first contribution has defined a cloud intrusion detection dataset, CIDD, the first dataset that can support the training and the evaluation of any cloud IDS. Current datasets are not suitable for these purposes because they neglect the typical behaviours of a cloud user and lack real attack patterns. CIDD solves these deficiencies and provides the complete audit parameters to support the detection of more than hundred instances of attacks and masquerades. CIDD consists of both knowledge and behavior based audit data and has real instances of host and network based attacks and masquerades. CIDD provides complete audit parameters from heterogeneous environments e.g., Windows, UNIX, and NetFlow, to evaluate the effectiveness of detection techniques. The comparison in Chapter 4 confirms the larger efficiency of CIDD with respect to current datasets. To build CIDD, we have developed a log analyzer and correlator system to parse and analyze the host based log files and network packets.

The second contribution of the thesis is the definition of the Data-Driven Semi-Global Alignment, DDSGA, approach and of three behavior based detection strategies. DDSGA is focused on the detection of anomalous user behaviour generated by masquerade attacks. Masquerading is by far one of the most critical attacks because once the attacker logs in successfully to a cloud, he/she can maliciously control the huge amount of resources it

includes. DDSGA improves both the security efficiency and the computational performance of SGA, a fast detection technique with low false positive rate that has not yet achieved the accuracy and performance for practical deployment. DDSGA aligns the sequence of the current user session to the previous ones of the same user and the presence of several misalignments is a strong indicator of a masquerade attack.

From the security efficiency perspective, DDSGA supports a more accurate modeling of distinct users as it introduces distinct parameters to model their behaviours. DDSGA can tolerate changes in the low-level representation of the commands functionality through two scoring systems that categorize user commands to align distinct commands in the same class without reducing the alignment score. Furthermore, to tolerate changes in the user behavior, DDSGA updates the signatures that describe this behavior according to the current user behavior All these features result in a strong reduction in false positive and missing alarm rates and as well as an increase in the detection hit ratio. According to our experiments, DDSGA achieves a better performance than SGA. As an example, it improves the hit ratio by about 21.9% and reduces Maxion-Townsend cost by 22.5%.

From the computational perspective, DDSGA simplifies the alignment by dividing the signature sequence into a smaller set of overlapped subsequences. Furthermore, it speeds up the detection and the update processes by running them in parallel.

A main reason of the low performances of current detection approaches is that they do not correlate the behaviour of a user in distinct environments, host and network, and in distinct cloud nodes. To solve this issue we have developed three detection strategies. The first strategy applies DDSGA to sequences of correlated audits from the VMs operating systems. We have evaluated this strategy on two distinct kinds of audits, system calls and security events. The second strategy analyzes NetFlow data from the network environment. The third strategy correlates the user behavior in host and network environments by integrating the other two strategies through a neural network. In this way, we convert masquerade detection from a binary problem to a classification or machine learning one. The evaluation has considered the three alternative correlation models mentioned before through both CIDS and CIDS-VERT frameworks based on CIDD data.

Chapter 6 has defined two extensions of DDSGA to detect masquerade attacks through, respectively, system call sequences and NetFlow data. We build a consistent profile of system calls through a "Behaviours Triangle Model" that focuses on system calls to implement file operations and process activities. A consistent NetFlow profile is built for each source IP address in terms of the sequences of nodes that are accessed and the protocols that are used. To efficiently correlate the user behaviour, we have evaluated the three correlation models i.e., Audit Exchange, Independent, and Centralized-Backup, using both CIDS and CIDS-VERT frameworks. Empirically, we have verified that correlation strongly improves the hit ratio by about 19.64% and reduces the Maxion-Townsend cost by 23.24. These experiments also show that the Independent model works much better with CIDS than the Audit Exchange model. This model is the ideal solution for small and private clouds as it achieves good accuracy and computational performance with low network overhead and short masquerade live time. Instead, the Centralized-Backup model works efficiently with CIDS-VERT in large clouds with good accuracy and computational performance, low network overhead and short masquerade live time.

After tuning and optimizing the correlation of user behaviour in each of the two detection subsystems, we have correlated the subsystem by integrated their results through a neural network. This results in the best overall accuracy, 98.07%, with respect to 94.24% of the host based and 83.04% of NetFlow based detection. As expected, it also results in the largest detection time, and the largest survival time of a masquerader because it waits for the results of both subsystems.

The experiments results of Chapter 7 concerns two subsystems that detect masquerade attacks through, respectively, sequences of security events and NetFlow audits. Consistent host based user profiles and NetFlow data profile for each source IP address are built as in Chapter 6. DDSGA compares the active log sessions in both host and network environments against the corresponding profile and computes the detection outputs for each subsystem. Then, it integrates these outputs using a neural network. As in Chapter 6, the proposed neural network model results in the best accuracy, 96.08%, with respect to 91.06% of host based and 88.41% of NetFlow based. The correlation improves the hit ratio by about 8.07% and reduces Maxion-

Townsend cost by 16.86. The experimental results further confirm the conclusion of Chapter 6 that the independent model is ideal for small and private cloud networks while big clouds like hybrid and public ones should prefer the Centralized-Backup one.

Finally, the third contribution is related to signature based detection. We introduce a hierarchical architecture that overcomes some limitations of current IDSs and supports two deployments, a Distributed and a Centralized one for the proposed IDS. The deployments use host based and network based IDSs that exploit signature based analysis techniques. The Distributed deployment distributes the computational overhead among several cloud VMs and reduces the network overhead while avoiding a single point of failure. However, its accuracy is lower than the one of the Centralized deployment. The latter also has a shorter detection time and a large overhead for the central VM and the cloud network. According to our experiments, the Centralized deployment improves the detection rate and also signals a higher number, 34.3%, of true alerts than the Distributed deployment. However, the Distributed deployment has a better detection time, 28.8%, than the Centralized one. 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 and reduces both false alarms and the number of alerts from HIDS and NIDS.

The diagram in Figure 1 resumes the work in this thesis to define, implement, and evaluate a general, efficient, and accurate cloud IDS that can be adopted in a very large number of clouds and that covers alternative deployment models.

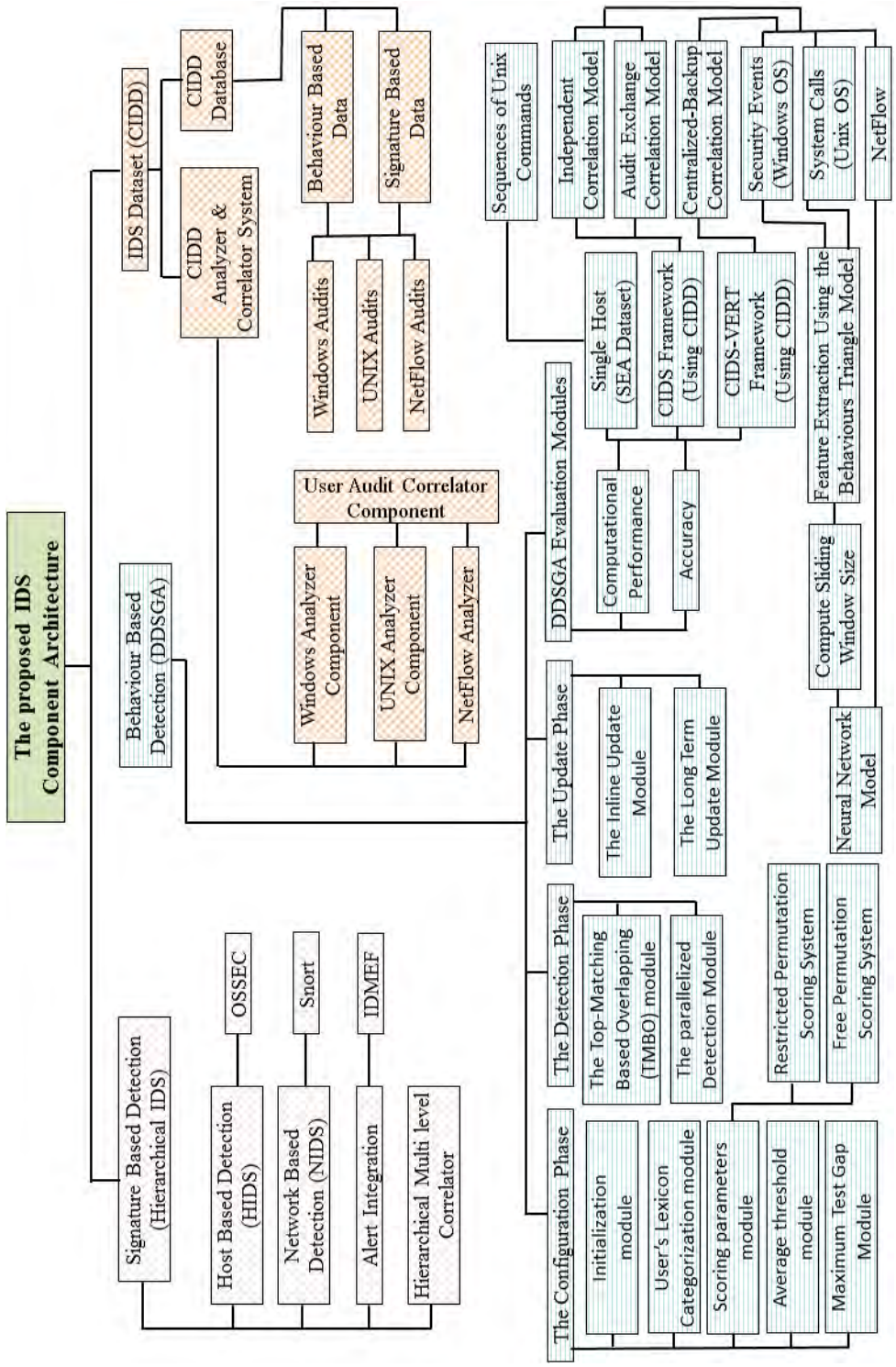


Figure 1: The Proposed Cloud IDS Components Diagram

As far as concern the possible developments of this thesis, we plan to integrate the behaviour based detection of DDSGA with current signature based detection techniques. DDSGA can detect anomalous behaviours for both users and hosts in a network. Hence, if the hosts or their users have a profile of normal behaviours, DDSGA can compare it against anomalous actions to block an anomalous user or host. We have introduced a similar analysis based on the masquerade actions in system calls and NetFlow in Chapter 6 and another one based on the security event and NetFlow in Chapter 7. 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. Another possible development concerns the adoption of alternative machine learning approaches to train our IDS and to maintain the validity of the proposed IDS over the system's life time. It would also be interesting to develop an adaptive control strategy for managing and evaluating cloud system performance and resilience under normal and abnormal conditions

Finally, we plan to extend our IDS system to provide autonomous capabilities particularly autonomous response and self-resilience and to provide a security measure to evaluate vulnerabilities and risks in a system as an essential milestone to build trust in cloud environments. In [165, 166 and 167], we have introduced a mechanism to build a security measure based on the assessment of the risks and the criticality of the security events. Furthermore, self-resilience is supported by, (a) preventing altering or modification of security events in the data storage, (b) avoiding single point of failure by replicating the intrusion detection components. The auto response actions are based on a set of polices defined by the system administrator. Lastly, we have built an early warning and forecasting model to predict host and network anomalies using a Hidden Markov Model and Holt Winter forecasting Algorithm [168].

References

[1] "Top Threats to Cloud Computing", Cloud Security Alliance, http://www.cloudsecurityalliance.org/csaguide.pdf, Version 1.0 (2010)

[2] "Security Guidance for Critical Areas of Focus in Cloud Computing", Cloud Security Alliance, http://www.cloudsecurityalliance.org/guidance/csaguide.pdf

[3] 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.

[4] Foster, I.; Yong Zhao; Raicu, I.; Lu, S., "Cloud Computing and Grid Computing 360-Degree Compared", Grid Computing Environments Workshop, 2008. GCE '08, vol., no., pp.1-10, 12-16 Nov. 2008

[5] http://wso2.com/cloud/stratos/

[6] "Windows Azure System" http://www.microsoft.com/windowsazure/windowsazure/

[7] Google App Engine, http://code.google.com/appengine/, 2008.

[8] Ali E. El-Desoky, Hisham A., Abdulrahman A. Azab, "A Pure Peer-to-Peer Desktop Grid Framework with Efficient Fault Tolerance", ICCES 24, Nov. 2007.

[9] Abdulrahman A. Azab, Hisham A Kholidy, "An Adaptive Decentralized Scheduling Mechanism for Peer-to-Peer Desktop Grids", (The 2008 International Conference on Computer Engineering & Systems) 25-27 Nov, 2008

[10] Abdulrahman Azab, Hein Meling, "Broker Overlay for Decentralized Grid Management", JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY, January 31, 2010.

[11] Amazon Elastic Compute Cloud, http://aws.amazon.com/ec2, 2008.

[12] Eucalyptus, http://eucalyptus.cs.ucsb.edu/, 2008.

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

[14] J. Brodkin. "Gartner: Seven cloud-computing security risks", http://www.networkworld.com/news/2008/070208-cloud.html, 2008.

[15] "Nubifer Cloud Portal", http://www.nubifer.com/SaaS-pass-hass-cloud-products/top-tier-cloud-platforms.html

[16] "Ubuntu Cloud Portal", http://cloud.ubuntu.com/2010/12/announcing-ubuntu-cloud-portal/

[17] Hisham A. Kholidy," HIMAN-GP: A Grid Engine Portal for controlling access to HIMAN Grid Middleware with performance evaluation using processes algebra", 2nd International Conference on Computer Technology and Development (ICCTD 2010), 2-4 Nov 2010.

[18] Akshay Luther, Rajkumar Buyya, Rajiv Ranjan, and Srikumar Venugopal, "GridCrypt: High Performance Symmetric Key Cryptography Using Enterprise Grids ", Laurence Yang and Minyi Guo (editors), Wiley Press, New Jersey, USA. Fall 2004.

[19] Mostafa-Sami M. Mostafa, Safia H Deif, Hisham A Kholidy, "ULTRA GRIDSEC: Peer-to-Peer Computational Grid Middleware Security Using High Performance Symmetric Key Cryptography", the 5th International Conference on Information Technology-New Generations, Las Vegas, Nevada, USA, 7- 9 April 2008.

[20] Hisham A Kholidy, Abdulrahman A. Azab, Safia H Deif, "Enhanced 'ULTRA GRIDSEC': Enhancing High Performance Symmetric Key Cryptography Schema Using Pure Peer-to-Peer Computational Grid Middleware (HIMAN)" in the Third International Conference on Pervasive Computing and Applications, 06-08 Oct 2008.

[21] Hisham A. Kholidy, Khaled S. Alghathbar, "Adapting and accelerating the stream Cipher algorithm "RC4" using "Ultra Gridsec" and "HIMAN" and use it to secure HIMAN Data", The Journal of Information Assurance and Security, JIAS/2009/SI4-008, Atlanta, USA, July 30, 2009. URL: http://www.softcomputing.net/jias/jias2009.html

[22] G. Popek, R. Goldberg. "Formal Requirements for Virtualizable Third Generation Architectures", Communications of the ACM. Volume 17, number 7, pages 412-421, 1974.

[23] J. Sugerman, V. Ganesh, L. Beng-Hong. Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor. Proceedings of the USENIX Annual Technical Conference, 2001.

[24] N. Kelem, R. Feiertag. "A Separation Model for Virtual Machine Monitors", Research in Security and Privacy. Proceedings of the IEEE Computer Society Symposium, pages 78-86, 1991.

[25] T. Garfinkel, M. Rosenblum. "A Virtual Machine Introspection Based Architecture for Intrusion Detection", Proceedings of the Network and Distributed System Security Symposium NDSS, 2003.

[26] http://blogs.msdn.com/b/virtual_pc_guy/archive/2006/07/10/661958.aspx

[27] http://download.microsoft.com/download/A/C/A/ACA22617-4FB8-45AD-BAEC-591A8E794A14/2Architecture_Future.ppt

[28] http://www.virtualbox.org/wiki/VirtualBox

[29] "Hosted Xen Project (HXen) ", http://blog.xen.org/index.php/2009/04/01/hosted-xen-project-kxen-available/

[30] http://www.vmware.com/products/vsphere/esxi-and-esx/index.html

[31] http://www.xen.org/

[32] "System Calls Definition", http://en.wikipedia.org/wiki/System_call

[33] "Event Log Service", http://support.microsoft.com/kb/308427

[34] "NetFlow Definition", http://en.wikipedia.org/wiki/NetFlow

[35] Austin Whisnant, Sid Faber, "Network Profiling Using Flow", TECHNICAL REPORT, CMU/SEI-2012-TR-006, August 2012.

[36] NetFlow/FloMA: Pointers and Software Provided by SWITCH. - One of the most comprehensive list including all the open source and research works

[37] http://www.vmware.com/company/news/releases/vmw-cloud-infrastructure-071211.html

[38] http://blogs.manageengine.com/netflowanalyzer/2011/03/11/netflow-vs-sflow/

[39] "Entropy" http://en.wikipedia.org/wiki/Entropy

[40] "Conditional Entropy" http://en.wikipedia.org/wiki/Conditional_entropy

[41] F. C. Hoppensteadt and E. M. Izhikevich, Weakly connected neural networks, Springer, (1997), p. 4. ISBN 978-0-387-94948-2.

[42] http://www.seeingwithc.org/topic5html.html

[43] http://technet.microsoft.com/en-us/library/cc959354.aspx

[44] http://en.wikipedia.org/wiki/Attack (computing)

[45] Gruschka, N.; Jensen, M.;,"Attack Surfaces: A Taxonomy for Attacks on Cloud Services," Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on, vol.,no., pp. 276-279, 5-10 July 2010, doi: 10.1109/CLOUD.2010.23

[46] http://www.metasploit.com/

[47] 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.

[48] Zaroo, P.; "A survey of DDoS attacks and some DDoS defense mechanisms", Advanced Information Assurance (CS 626), 2002

[49] 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

[50] "CPU DEATHE PING Attack", http://www.hackerbradri.com/2012/08/cpu-death-ping-20.html

[51] "NESSUS System", http://www.tenable.com/products/nessus

[52] 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

[53] 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.

[54] 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

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

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

[57] 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.

[58] http://www.citrix.com/products/xenserver/overview.html

[59] David Chappell, "THE MICROSOFT PRIVATE CLOUD", A Technical Overview Report, August, 2011.

[60] "VMware Workstation 9.0 Release Notes", September 2012. http://www.vmware.com/support/ws90/doc/workstation-90-release-notes.html.

[61] "OSSEC System", http://www.ossec.net/main/

[62] OSSIM Manual, http://www.alienvault.com/documentation/index.html

[63] Rebecca Bace and Peter Mell, "NIST Special Publication on Intrusion Detection Systems", National Institute of Standards and Technology, 16 August 2001.

[64] Herve Debar, Marc Dacier, Andreas Wespi, "Towards a taxonomy of intrusion-detection systems", Computer Networks, Volume 31, Issue 8, 23 April 1999, Pages 805-822, ISSN 1389-1286, DOI: 10.1016/S1389-1286(98)00017-6.

[65] Shahaboddin Shamshirband1, Nor Badrul Anuar, Miss Laiha Mat Kiah, and Ahmed Patel, "An Appraisal and Design of Multi Agent Systems Based on Cooperative Wireless Intrusion Detection Computational Intelligence Techniques", the science and technology of sensors and biosensors journal (Sensors ISSN 1424-8220).

[66] Elshoush, H.T.; Osman, I.M. Alert correlation in collaborative intelligent intrusion detection systems—A survey. Applied Soft Computing 2011, 11, 4349-4365.

[67] Blasco, J.; Orfila, A.; Ribagorda, A. In Improving Network Intrusion Detection by Means of Domain-Aware Genetic Programming, ARES '10 International Conference on Availability, Reliability, and Security, 2010, 2010; 2010; pp. 327-332.

[68] Phillip A. Porras and Alfonso Valdes_ Live traffic analysis of tcp/ip gateways, Proc. ISOC Symposium on Network and Distributed SystemSecurity (NDSS'98) (San Diego, CA, March 1998) (Internet Society).

[69] Teresa Lunt and R. Jagannathan, "A prototype real-time intrusion detection expert system", Proc. Symposium on Security and Privacy (Oakland, CA, April 1988) 59-66.

[70] Naji Habra, Baudouin Le Charlier, Aziz Mounji, and Isabelle Mathieu, "Asax: Software architecture and rule-based language for universal audit trail analysis", in Proc. of 2nd European Symposium on Research in Computer Security (ESORICS), Toulouse, France, November 1992, LCNS 648, Springer-Verlag, Berlin, Germany.

[71] Phillip Porras and Richard Kemmerer, "Penetration state transition analysis - A rule-based intrusion detection approach", Proc. 8th Annual Computer Security Applications Conference 220-229 (IEEE Computer Society press, IEEE Computer Society press, November 1992)

[72] Paul Helman and Gunar Liepins, Statistical foundations of audit trail analysis for the detection of computer misuse, IEEE Transactions on Software Engineering, 19 September 1993, 886-901.

[73] Herve Debar, Monique Becker, and Didier Siboni, "A neural network component for an intrusion detection system", Proc. 1992 IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA, May 1992

[74] Thomas Spyrou and John Darzentas, "Intention modeling: Approximating computer user intentions for detection and prediction of intrusions", In S.K. Katsikas and D. Gritzalis, editors, Information Systems Security (Samos, Greece, May 1996) 319-335, (Chapman and Hall).

[75] Paul Spirakis, Sokratis Katsikas, Dimitris Gritzalis, Francois Allegre, John Darzentas, Claude Gigante, Dimitris Karagiannis, P. Kess, Heiki Putkonen, and Thomas Spyrou, "SECURENET: A network-oriented intelligent intrusion prevention and detection system", Network Security Journal, 1994.

[76] Debar, H., Curry, D., Feinstein, B.: "The Intrusion Detection Message Exchange Format", Internet Draft Technical Report, IETF Intrusion Detection Exchange Format Working Group (July 2004).

[77] Roschke, S., Cheng, F., Meinel, "Intrusion Detection in the Cloud", The 8th International Conference on Dependable, Autonomic and Secure Computing (DASC-09) Chengdu, China, December 12-14, 2009

[78] Karen Scarfone and Peter Mell, "Guide to Intrusion Detection and Prevention Systems (IDPS)", National Institute of Standards and Technology, Special Publication 800-94, Feb. 2007.

[79] "Cisco Guard XT System" http://www.cisco.com/en/US/products/ps5894/index.html

[80] "SAMHAIN IDS", http://www.la-samhna.de/samhain/

[81] "OSIRIS IDS" http://osiris.shmoo.com/index.html

[82] "eEYE RetinaIDS" http://www.visus-it.com/eeye.php

[83] http://www.symantec.com/connect/articles/introduction-distributed-intrusion-detection-systems

[84] Jansen W., "Intrusion detection with mobile agents", Computer Communications 25 (15): 1392-1401, 2002.

[85] Jansen W., Karygiannis, T. 1999, "Mobile agents and security". Special Publication 800-19, National Institute of Standards and Technology (NIST)

[86] Porras, A. Ph.; Neumann, P. G.: EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances, Proc. of the National Information Systems Security Conference, Oct. 6, 1997, Baltimore, Maryland.

[87] Brett C. Tjaden, Lonnie R. Welch, Shawn D. Ostermann, David Chelberg, et. al, "INBOUNDS: The Integrated Network-Based Ohio University Network Detective Service", 4th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2000) and 6th International Conference on Information Systems Analysis and Synthesis (ISAS 2000), Orlando, Florida, July 23-26, 2000.

[88] Harold S. Javitz, Alfonso Valdes, "The NIDES Statistical Component Description and Justification", in SRI Technical Report, 1994

[89] Lin J., Wang X., Jajodia S. Abstraction-Based Misuse Detection: High-Level Specifications and Adaptable Strategies. csfw, p. 190, 11th IEEE Computer Security Foundations Workshop (1998).

[90] Smaha, S. E.; Winslow, J.: Misuse detection tools, Computer Security Journal 10(1994)1, Spring, 39 - 49.

[91] Christoph, G. G.; Jackson, K. A.; Neumann, M. C.; Siciliano, Ch. L. B.; Simmonds, D. D.; Stallings, C. A.; Thompson, J. L.: UNICORN: Misuse Detection for UNICOS, Proc. of the Supercomputing '95, San Diego, CA.

[92] Jeffrey M. Bradshaw, "An Introduction to Software Agents," In Jeffrey M. Bradshaw, editor, Software Agents, Chapter 1. AAAI Press/The MIT Press, 1997.

[93] W Jansen, P Mell, T Karygiannis, Marks, "Applying Mobile Agents to Intrusion Detection and Response (1999)", National Institute of Standards and Technology Interim Report – 6416

[94] 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

[95] Scerri, Paul and Vincent, Régis and Mailler, Roger, booktitle "Coordination of Large-Scale Multiagent Systems", Springer US, isbn: 978-0-387-27972-5, p. 231-254, http://dx.doi.org/10.1007/0-387-27972-5_11, note:10.1007/0-387-27972-5_11, 2006.

[96] Ricardo Leite, Fabiano Oliveira, Constantino Ribeiro, Jauvane Oliveira, Bruno Schulze and Edmundo.Madeira, "Functionalities in Grid Computing with Active Services", 1st International Workshop on Middleware for Grid Computing, June 2003, Rio de Janeiro, Brazil.

[97] TALNAR Vanish, BASU Sujoy, KUMAR Raj, "An Environment for Enabling Interactive Grids", In IEEE INTERNATIONAL SYMPOSIUM ON HIGH PERFORMANCE DISTRIBUTED COMPUTING, 12, 2003, Seattle, Washington, USA. Proceedings Washington: IEEE Computer Society, 2003. p. 184-195.

[98] O. Choon and A. Samsudin, "Grid-based intrusion detection system," in Proc. 9th Asia-Pacific Conference on Communications, vol. 3, pp. 1028-1032, September 21-24,2003.

[99] S. Kenny and B. Coghlan, "Towards a grid-wide intrusion detection system," in Proc. European Grid. Conference (EGC2005), pp. 275-284, Amsterdam, Netherlands, February 2005.

[100] F-Y. Leu, Fang-Yie Leu, Jia-Chun Lin, Ming-Chang Li, Chao-Tung Yang, and Po-Chi Shih, "Integrating Grid with Intrusion Detection", Proc. Int'l Conf. Advanced Information Networking and Applications (AINA 05), vol. 1, IEEE CS Press, March, 2005, pp. 304–309

[101] M. Tolba, M. Abdel-Wahab, and I. Taha, and A. Al-Shishtawy, "GIDA: Toward enabling grid intrusion detection systems", in Proc. 5th IEEE/ACM Int. Symp. on Cluster Computing and the Grid (CCGrid2005), Cardiff, UK, May 9-12, 2005.

[102] Fang-Yie L., Jia-Chun L., Ming-Chang L., and Chao-Tung Y., "A Performance-Based Grid Intrusion Detection System," in Proc. 29th Annual IEEE International Computer Software and Applications Conference (COMPSAC2005), pp. 525-530, July 26-38, 2005.

[103] Guofu Feng, Xiaoshe Dong, Weizhe Liu, Ying Chu, and Junyang Li, "GHIDS: Defending Computational Grids against Misusing of Shared Resource", Proc. Asia-Pacific Conf. Services Computing (APSCC 06), IEEE CS Press, December, 2006, pp. 526–533.

[104] Schulter, A.; Navarro, F.; Koch, F.; Westphall, C.B., "Intrusion Detection for Computational Grids", Proc. 2nd Int'l Conf. New Technologies, Mobility, and Security, IEEE Press, November, 2008, pp. 1–5.

[105] Vieira, K.; Schulter, A.; Westphall, C.B.; Westphall, C.M.;, "Intrusion Detection for Grid and Cloud Computing," IT Professional, vol.12, no.4, pp.38-43, July-Aug. 2010.

[106] Hisham A. Kholidy, "A Study for Access Control flow Analysis with a proposed Job analyzer component based on Stack inspection methodology", 10th International Conference on Intelligent System Design and Applications (ISDA 2010), 29 Nov -2 Dec 2010.

[107] "Condor System", http://www.cs.wisc.edu/condor/

[108] "Globus System", http://www.globus.org/

[109] Dan S. Wallach, Edward W. Felten, "Understanding Java Stack Inspection", In IEEE Symposium on Security and Privacy,p 2-5,1998.

[110] A. H. Phyo and S. M. Furnell. A detection-oriented classification of insider and misuse. In Proceedings of the Third Security Conference, 2004.

[111] Lee, W. and Stolfo, S. Data mining approaches for intrusion detection. In Proc.7th USENIX Security Symposium, 1998.

[112] Maxion, R. A., and Townsend, T. N. 2002. Masquerade Detection Using Truncated Command Lines. In Proceedings of the International Conference on Dependable Systems and Networks, Washington, D.C., June 2002, 219-228.

[113] Brown, Christopher D.; and Davis, Herbert T. (2006); Receiver operating characteristic curves and related decision measures: a tutorial, Chemometrics and Intelligent Laboratory Systems, 80:24–38.

[114] Schonlau, M., DuMouchel, W., Ju, W., Karr, A. F., Theus, M., and Vardi, Y. 2001. Computer Intrusion: Detecting Masquerades. Statistical Science 16(1), 58-74.

[115] Dumouchel, W. (1999). Computer intrusion detection based on Bayes Factors for comparing command transition probabilities. Technical report 91, National Institute of Statistical Sciences. Available at www.niss.org/downloadabletechreports.html.

[116] Ju, W. and Vardi, Y. (1999). A hybrid high-order Markov chain model for computer intrusion detection. Technical report 92, National Institute Statist. Sci. Available at www.niss.org/downloadabletechreports.html.

[117] Davison, B. D. and Hirsh, H. (1998). Predicting sequences of user actions. In Predicting the Future: AI Approaches to Time Series Problems. Technical report WS-98-07 (Proceedings of AAAI-98/ICML-98 Workshop) 5–12. AAAI Press, Madison, WI.

[118] Lane, T. and Brodley, C. E. (1998). Approaches to online learning and concept drift for user identification in computer security. Proceedings, The Fourth International Conference ofKnowledge Discovery and Data Mining, August 27–31, New York. 259–263.

[119] Burges Christopher JC. A tutorial on support vector machines for pattern recognition. Data Mining and Knowledge Discovery 1998;2(2):121e67.

[120] Szymanski, B., and Zhang, Y. 2004. Recursive Data Mining for Masquerade Detection and Author Identification. In Proceedings of the 5th IEEE System, Man and Cybernetics Information Assurance Workshop, West Point, June 2004, 424-431.

[121] Dash, S.K., Reddy, K.S., Pujari, A.K.: Episode based masquerade detection. Lecture Notes in Computer Science, vol. 3803. Springer, Berlin, pp. 251–262 (2005)

[122] Alok Sharma, Kuldip K. Paliwal. Detecting masquerades using a combination of Naïve Bayes and weighted RBF approach. Journal in Computer Virology, 3(3):237-245, 2007

[123] Subrat Kumar Dash, Krupa Sagar Reddy, Arun K. Pujari: Adaptive Naive Bayes method for masquerade detection. Security and Communication Networks 4(4): 410-417 (2011)

[124] Coull, S. E., Branch, J. W., Szymanski, B. K., and Breimer, E. A. 2003. Intrusion Detection: A Bioinformatics Approach. In Proceedings of the 19th Annual Computer Security Applications Conference, Las Vegas, NV, December 2003, 24-33.

[125] Scott E. Coull, Joel W. Branch, Boleslaw K. Szymanski, Eric A. Breimer. 2008. "Sequence alignment for masquerade detection". Journal of Computational Statistics & Data Analysis. 52, 8 (April 2008), 4116-4131. DOI=10.1016/j.csda.2008.01.022 http://dx.doi.org/10.1016/j.csda.2008.01.022

[126] A. Garg, R. Rahalkar, S. Upadhyaya, K. Kwiat, "Profiling Users in GUI Based Systems for Masquerade Detection", Proceedings of the 2006 IEEE Workshop on Information Assurance, pp.48-54.

[127] Imsand, E., Hamilton, J. GUI Usage Analysis for Masquerade Detection. in the 2007 IEEE Workshop on Information Assurance. June 20-22, West Point, NY, USA. Pages 270-277

[128] A. Wespi, M. Dacier, and H. Debar, "Intrusion Detection Using Variable-Length Audit Trail Patterns," in Recent Advances in Intrusion Detection - Lecture Notes in Computer Science, vol. 1907, pp. 110-129, Springer-Verlag, 2000.

[129] C. Marceau, "Characterizing the behavior of a program using multiple-length N-grams," in the 2000 workshop on New security paradigms, (Ballycotton, County Cork, Ireland), pp. 101-110.

[130] Chris Strasburg, Sandeep Krishnan, Karin Dorman, Samik Basu, Johnny S. Wong, "Masquerade Detection in Network Environments", Proceedings of the 2010 10th IEEE/IPSJ International Symposium on Applications and the Internet, 2010.

[131] Aubrat Kumar, Sanjay Rawat, G. Vijaya Kumari, and Arun K. Pujari, "Masquerade Detection Using IA Network", CPSec 2005.

[132] Allen, J.: maintaining knowledge about temporal intervals. Communications of the ACM, 26, (1983) 832-843

[133] Smith, T. F., and Waterman, M. S. 1981. Identification of Common Molecular Subsequences. Journal of Molecular Biology 147, 195-197.

[134] Maximiliano Bertacchini and Pablo Fierens. A Survey on Masquerader Detection Approaches. In Proceedings of Congreso Iberoamericano de Seguridad Inform´atica, pages 46-60, Montevideo, Uruguay. Universidad de la Rep´ublica de Uruguay, 2008. ISBN: 978-9974-0-0593-8.

[135] Malek Ben Salem, Shlomo Hershkop, Salvatore J. Stolfo. "A Survey of Insider Attack Detection Research" in Insider Attack and Cyber Security: Beyond the Hacker, Springer, 2008

[136] "Schonlau webpage", http://www.schonlau.net/intrusion.html

[137] Greenberg, S.: Using unix: Collected traces of 168 users. Research Report 88/333/45, Department of Computer Science, University of Calgary, Alberta, Canada (1988), Saul Greenberg's homepage: http://pages.cpsc.ucalgary.ca/~saul/

[138] Lane, T., Brodley, C.E.: An application of machine learning to anomaly detection. In: In Proceedings of the 20th National Information Systems Security Conference. (1997) 366-380

[139] RUU dataset: http://sneakers.cs.columbia.edu/ids/RUU/data/"

[140] Posadas, R., Mex-Perera, J.C., Monroy, R., Nolazco-Flores, J.A.: Hybrid method for detecting masqueraders using session folding and hidden markov models. In Gelbukh, A.F., Garcia, C.A.R., eds.: MICAI. Volume 4293 of Lecture Notes in Computer Science., Springer (2006) 622-631

[141] Maxion, R. A. 2003. Masquerade Detection using Enriched Command Lines. In International Conference on Dependable Systems (DSN-03). San Francisco, CA, June 2003, 5-14.

[142] Hisham A. Kholidy, Fabrizio Baiardi, "CIDS: A framework for Intrusion Detection in Cloud Systems", The 9th International Conf. on Information Technology: New Generations (ITNG), Las Vegas, Nevada, USA, 2012

[143] http://www.di.unipi.it/~hkholidy/projects/cids/

[144] 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.

[145] D. Andersson, M. Fong, and A. Valdes, "Heterogeneous Sensor Correlation: A Case Study of Live Traffic Analysis," Third Ann. IEEE Information Assurance Workshop, Jun. 2002.

[146] A. Valdes and K. Skinner, "An Approach to Sensor Correlation,"Proc. Recent Advances in Intrusion Detection, Oct. 2000.

[147] Fredrik Valeur, Giovanni Vigna and Richard A. Kemmerer, "A Comprehensive Approach to Intrusion", IEEE Transactions on Dependable and Secure Computing, Jul. 2004

[148] "Open Stack System", http://www.openstack.org/

[149] http://blog.eukhost.com/webhosting/what-is-the-difference-between-public-clouds-and-a-private-cloud/

[150] Hisham A. Kholidy, Fabrizio Baiardi, "CIDD: A Cloud Intrusion Detection Dataset for Cloud Computing and Masquerade Attacks", the 9th International Conference on Information Technology: New Generations (ITNG), Las Vegas, Nevada, USA, 2012. http://www.di.unipi.it/~hkholidy/projects/cidd/

[151] "DARPA Intrusion Detection System Group",

[152] http://www.ll.mit.edu/mission/ communications/ist/corpora/ideval/data/index.html

[153] Wang, K., and Stolfo, S. J. 2003. One Class Training for Masquerade Detection. In Proc. of the 3rd IEEE Conference Data Mining Workshop on Data Mining for Computer Security, Florida, November 2003

[154] "DDSGA Approach", http://www.di.unipi.it/~hkholidy/projects/DDSGA/

[155] N. Nguyen, P. Reiher, and G. Kuenning, "Detecting Insider Threats by Monitoring System Call Activity", Proceedings of the 2003 IEEE Workshop on Information Assurance, NY June 2001.

[156] E. Eskin, W. Lee, and S. J. Stolfo. Modeling system calls for intrusion detection with dynamic window sizes. In Proceedings of DARPA Information Survivability Conference and Exposition II (DISCEX II), Anaheim, CA, 2001.

[157] E. Eskin,W. N. Grundy, and Y. Singer. Protein family classification using sparse markov transducers. In Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology, Menlo Park, CA, 2000. AAAI Press.

[158] "Augmented Matrix", http://en.wikipedia.org/wiki/Augmented_matrix

[159] Hisham A. Kholidy, Fabrizio Baiardi, Salim Hariri, "Detecting Masquerades in Clouds through Security Events and NetFlow Data Analysis", in Journal of Computer Science and Technology (JCST), under review in December 2012.

[160] "System Log Files", http://en.wikipedia.org/wiki/Syslog

[161] "Open BSM System", http://en.wikipedia.org/wiki/OpenBSM

[162] "UDP stateless connections and their establishment",

[163] http://www.frozentux.net/iptables-tutorial/chunkyhtml/x1555.html

[164] "Detection of Multistage Attack in Federation of Systems Environment", Przemysław Bereziński, J. Śliwa, R. Piotrowski, B. Jasiul, Military Communication Institute

[165] Hisham A. Kholidy, Abdelkarim Erradi, Sherif Abdelwahed, Fabrizio Baiardi, "HA-CIDS: A Hierarchical and Autonomous IDS for Cloud Environments", Fifth International Conference on Computational Intelligence, Communication Systems and Networks (CICSyN) Madrid, Spain, June, 2013.

[166] Hisham A. Kholidy, Abdelkarim Erradi, Sherif Abdelwahed, Fabrizio Baiardi, "Autonomous Response, Self-Resilience, and Prediction in a Cloud IDS", under review in April 2013, the ACM Cloud and Autonomic Computing Conference (CAC 2013), Miami, Florida, USA August, 2013.

[167] Hisham A. Kholidy, Abdelkarim Erradi, Sherif Abdelwahed, Fabrizio Baiardi, "A Hierarchical, Autonomous, and Forecasting Cloud IDS ", under review in May 2013, The 5th International Conference on Modeling, Identification and Control (ICMIC2013), Cairo, Aug31-Sept 1-2, 2013.

[168] C. Chatfield (1978). The Holt-Winters Forecasting Procedure. Journal of the Royal Statistical Society. Series C (Applied Statistics), Vol. 27, No. 3, pp. 264-279.

Our Publications Underlying the Thesis Work:

- **Journals:**

[1] Hisham A. Kholidy, Fabrizio Baiardi, Salim Hariri, Esraa M. ElHariri, Ahmed M. Youssouf, and Sahar A. Shehata, "A Hierarchical Cloud Intrusion Detection System: Design and Evaluation", in International Journal on Cloud Computing: Services and Architecture (IJCCSA), Vol.2, No.6, December 2012, DOI: 10.5121/ijccsa.2012.2601.

[2] 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 second review in March 2013. http://www.di.unipi.it/~hkholidy/projects/DDSGA/

[3] 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.

[4] Hisham A. Kholidy, Fabrizio Baiardi, Salim Hariri, "Detecting Masquerades in Clouds through Security Events and NetFlow Data Analysis", in Journal of Computer Science and Technology (JCST), under review in December 2012.

- **Conferences**

[1] Hisham A. Kholidy, Fabrizio Baiardi, "CIDS: A framework for Intrusion Detection in Cloud Systems", The 9th International Conf. on Information Technology: New Generations (ITNG), Las Vegas, Nevada, USA, 2012. Conference publisher: IEEE.

http://www.di.unipi.it/~hkholidy/projects/cids/.

[2] Hisham A. Kholidy, Fabrizio Baiardi, "CIDD: A Cloud Intrusion Detection Dataset for Cloud Computing and Masquerade Attacks", the 9th International Conference on Information Technology: New Generations (ITNG), Las Vegas, Nevada, USA, 2012. Conference publisher: IEEE.

http://www.di.unipi.it/~hkholidy/projects/cidd/

[3] Hisham A. Kholidy, Abdelkarim Erradi, Sherif Abdelwahed, Fabrizio Baiardi, "HA-CIDS: A Hierarchical and Autonomous IDS for Cloud Environments", Fifth International Conference on Computational Intelligence, Communication Systems and Networks (CICSyN) Madrid, Spain, June, 2013. Conference publisher: IEEE.

[4] Hisham A. Kholidy, Abdelkarim Erradi, Sherif Abdelwahed, Fabrizio Baiardi, "Autonomous Response, Self-Resilience, and Prediction in a Cloud IDS", under review in April 2013, the ACM Cloud and Autonomic Computing Conference (CAC 2013), Miami, Florida, USA August, 2013. Conference publisher: ACM.

[5] Hisham A. Kholidy, Abdelkarim Erradi, Sherif Abdelwahed, Fabrizio Baiardi, "A Hierarchical, Autonomous, and Forecasting Cloud IDS ", under review in May 2013, The 5th International Conference on Modeling, Identification and Control (ICMIC2013), Cairo, Aug31-Sept 1-2, 2013. Conference publisher: IEEE.

[6] Hisham A. Kholidy, "HIMAN-GP: A Grid Engine Portal for controlling access to HIMAN Grid Middleware with performance evaluation using processes algebra", The 2nd International Conference on Computer Technology and Development ICCTD 2010, pp 163-168, Cairo, 2010.

[7] Hisham A. Kholidy, "A Study for Access Control Flow Analysis With a Proposed Job Analyzer Component based on Stack Inspection Methodology", the 2010 10th International Conference on Intelligent Systems Design and Applications (ISDA), pp 1442-1447, Cairo, Egypt, vol. IEEE Catalog Number: CFP10394-CDR, 2010

[8] Hisham A. Kholidy, Chatterjee N., "Towards Developing an Arabic Word Alignment Annotation Tool with Some Arabic Alignment Guidelines", the 2010 10th International Conference on Intelligent Systems Design and Applications (ISDA), pp 778-783, Cairo, Egypt, vol. IEEE Catalog Number: CFP10394-CDR, 2010

Thank You