Attacks on Cloud Environments and their Mitigation

Host-based Isolated and Coordinated Attacks

Suaad S. Alarifi

Thesis submitted to the University of London
for the degree of Doctor of Philosophy

2015

Attacks on Cloud Environments and their Mitigation

Department of Mathematics
Royal Holloway, University of London

Declaration of Authorship

I, Suaad S. Alarifi, hereby declare that this thesis and the work presented in it is entirely my own. Where I have consulted the work of others, this is always clearly stated.

Signed:

(Suaad S. Alarifi)

Date:

Preface

This thesis has been written based on experiments conducted in the Penetration Testing Laboratory, which operates under the Information Security Group at Royal Holloway, University of London (RHUL). The Ministry of Higher Education in Saudi Arabia sponsored the study and the Mathematics department at RHUL covered travel and conference expenses.

Throughout the research, my supervisor, Dr Stephen Wolthusen, supported me with his rich knowledge and experience in the field. He was always available via email, Skype, or face-to-face meetings in his office.

Summary

The main goal of this research is to improve the security of large-scale public IaaS (Infrastructure-as-a-Service) cloud computing environments on the provider side. We aim to help providers to be more aware of the cloud's overall security and to have a sense of control in areas where they actually have no control at all (i.e. the activities inside the hosted Virtual Machines, or VMs).

In an IaaS model, computing infrastructure is delivered as a service. Consumers can deploy and run their VMs in the cloud infrastructure that acts as a hosting environment and offers virtual resources to VMs to consume. Each VM is under the full control of its owner (the client) and contains operating systems, applications, and services.

The hosted VMs are managed off-premises by the clients' IT teams, which could be security naive, have malicious intentions, or just carelessly ignore security policies and good practices. Insecure VMs hosted in the public cloud share the service with other VMs that belong to different organisations which represent a major security threat.

Providers are trying to manage this threat through contracts and legal obligations. However, finding the source of the threat is a hard task if there are no security monitoring tools. Even though providers have no control over what is happening inside VMs, they are still responsible for protecting the hosted VMs, the infrastructure, keeping VMs from attacking each other, preventing attacks originating from their network, and most importantly being able to find the source of the threat. Therefore, detection systems are needed to monitor each of the hosted VMs without invading their privacy and with the minimum performance overhead.

Providers have to monitor VMs and detect any abnormal activities without requiring any instrumentation inside the VMs. Most of the cloud monitoring tools available today are designed for performance monitoring, not security purposes.

For this research, we developed two detection systems that are able to monitor VMs without any level of intrusiveness; we argue that this level of granularity is sufficient for capturing a number of relevant attack classes. The developed systems were able to detect abnormal activities within VMs and generate strong anomaly signals. The first detection system is based on a very low-demanding statistical method called bag of system calls (BoSC); the second system is based on a more computationally expensive machine learning method called hidden Markov model (HMM). The second system is designed specifically to monitor ephemeral VMs (VMs with a short life) because it requires less training data and less time to be ready.

In this research we also studied different cloud attacks and developed a cloud-specific Denial of Service (DoS) class of attacks that work by misusing two of the

main features of the cloud: over-commitment and migration. We call this newly developed class of attacks "Cloud-Internal Denial of Service" attacks, or CIDoS. This attack targets the architecture of the cloud, not the implementation, which makes it harder to defeat. Then we suggested some detection and prevention mechanisms. After that, we developed another attack that instrumented a CIDoS attack with reverse engineered migration algorithms in the cloud to extract parameters that help improve the DoS attack and make it harder to detect and defeat.

Acknowledgments

I take this opportunity to express my profound gratitude to and sincere regard for my guide Dr Stephen Wolthusen for his exemplary guidance, monitoring, and constant encouragement throughout the course of this thesis. My research skills have greatly improved because of his guidance, which has allowed me to start conducting research and publishing papers.

I also take this opportunity to express a deep sense of gratitude to my country, Saudi Arabia, for their financial support and for their huge scholarship program which is a wise way to increase the awareness of Saudi people and to defeat extremism.

I also thank my university in Saudi Arabia, King Abdulaziz University, and my university in the UK, Royal Holloway University of London for giving me a chance and always supporting me.

I thank my father, who always believes in me, my mother for her support and help in taking care of my daughters, and my husband for his constant encouragement and support, without which this assignment would not be possible. Lastly, I apologise to my daughters, Haya and Rose, for being extremely busy, but we have to fully accept life's challenges.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The main goal of this research is to improve the security of public IaaS (Infrastructure-as-a-Service) large-scale cloud computing environments on the provider side. We aim to increase providers' awareness regarding the cloud's overall security status and help providers have a sense of control in areas where they have no control at all, such as the activities inside hosted virtual machines (VMs).

Cloud providers in an IaaS model are responsible for protecting both the infrastructure and the hosted VMs in addition to preventing VMs from attacking each other. They are also responsible for preventing any attack originating from their network from affecting the outside world.

Large-scale public cloud environments are open in nature, which gives attackers easy access to the cloud network as consumers. They can easily access the provider network sharing the same physical hosts with other bystander consumers. Malicious consumers might use their VMs and try to extract sensitive data from other co-resident VMs, prevent them from using the service, starve them, steal the service from them, use the service at their expense, attack the infrastructure, saturate the cloud network, or disrupt the service, all of which might affect the reliability of the service, in addition to affecting provider's reputation and might lead to a Service Level Agreement breach.

The cloud would be more secure if providers could successfully prevent attackers from being consumers in the first place; however, this is very challenging in large-scale public cloud environments where there are hundreds of thousands of VMs, some are permanent and others are temporary, and automated provisioning is one of the essential characteristics of the cloud. According to the NIST definition of cloud computing, "a consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed *automatically* without requiring human interaction with each service provider" [122].

There are plenty of research today focused on authentication in the cloud and checking the true identity of potential requesters of the service; however, this is a very complicated task and there will always be vulnerabilities; the attacker may frequently succeed in having a foot inside the cloud either by being a consumer, hijacking VMs, stealing legitimate consumers' credentials, attacking vulnerable services or unpatched VMs, or any other means.

Maintaining the security of the large-scale cloud is more challenging because VMs are 100% under the control of the cloud's clients.

Since:

1. providers have no control inside VMs and they are 100% under the control of the client (VM owner, also called the client or consumer),

2. monitoring VMs is essential to maintain the security of the cloud and protect other co-resident VMs,
3. hosted VMs are potential threats because the cloud is an open environment[1] and clients are free to choose what to install in their VMs (operating systems, applications, security tools, and possibility highly vulnerable applications) which means, the cloud environment should be considered hostile,
4. providers are responsible for maintaining the security of the cloud and preventing VMs from attacking each other, the infrastructure, or the outside world. They are responsible for monitoring and detecting attacks on their network and within hosts.

There is a need for tools that allow providers to monitor hosted VMs and detect any potential threat. Having no control inside VMs makes the provider's responsibility of maintaining security a hard goal to achieve. The straightforward solution for this problem is to allow providers to have control within VMs by installing agents or monitoring applications inside each of them. This solution is applicable in limited private IaaS clouds but not in large-scale public clouds. This is true for many reasons:

- Accessing each of the possibly *millions* of hosted VMs in a large-scale public cloud (which is the environment we target in this research) and collecting detailed data about all of their activities, analysing these data, extracting meaningful security information, and making decisions based on that information will definitely impose a great performance overhead, which will increase the cost and carbon footprint of the service.
- The cost model in the cloud is *pay for what you use*. Therefore, any activities inside VMs consume resources, and clients are billed for these. If security monitoring tools are deployed inside VMs, who will be charged for the resources consumed by the monitoring tools? If the clients are charged for the monitoring activities, this means they will pay for resources they have not used, which is against the cloud cost model.
- Allowing providers to access hosted VMs will heavily affect clients' privacy; monitoring applications that are managed by the provider will allow access to clients' private data.
- There is already a lack of trust between providers and clients; allowing providers to access VMs will amplify this problem. Clients are not sure if providers are maintaining the security of the cloud properly, following the best security practices, and protecting their data. On the other hand, providers cannot be sure of whether a client is innocent or a hacker and using the service properly or for malicious purposes. Providers also cannot be sure if clients are following the best security practices by using safe applications and operating systems, applying security patches, and maintaining their credentials properly. Allowing providers to access clients' VMs will amplify the trust problem.

[1] A public cloud offers services (automated provisioning) to almost anyone that can pay the cost, i.e. businesses, governments, individuals, and possible attackers [181]. This is discussed further in the state of the art chapter, chapter 3.

- The ability to monitor VMs from the inside might increase the provider liability for VMs' illegal activities.

Therefore, for security (trust and privacy), legal, and performance reasons, limiting the depth of intrusiveness benefits providers and should be one of their main targets. We aim to improve the security of large-scale public IaaS by developing tools that allow providers to monitor VMs and detect any anomalous behaviour inside VMs without accessing them. The idea is to build an identity or a character for each VM and detect any change in behaviour and consider it as anomaly. This idea will also allow a provider to assess the security condition of clients, and thus the cloud's overall security status. It will also help providers have a sense of control in areas where they have no control at all, such as the activities inside hosted VMs.

In this research, we proposed a method to monitor system calls at the VM host level without requiring any instrumentation within VMs.[2] We argue that this level of granularity is sufficient to capture numerous relevant attack classes. This, together with the efficiency of the approach, is shown through experiments and statistical analysis in a Linux KVM-based reference scenario.

We monitored system calls because they reflect every activity occurring inside the monitored VMs. The only way of communication between a VM and its host is through system calls. We dealt with the VM as a black box and found two streams of data coming out of it that might give indications about normal and abnormal activities occurring inside. The first source of data is the system calls, and the second is network traffic (packets). The monitoring of inbound and outbound network traffic is generally performed using network-based intrusion detection systems (NIDSs) and firewalls, which are outside of the scope of this research. We focus on monitoring VMs using hypervisor host-based intrusion detection systems (HIDS). The proposed systems are not an alternative to NIDS, but rather are complementary.

Furthermore, system call-based HIDS has been proven, in the literature, to be very efficient at creating representative normal behaviour profiles for hosts, in addition to providing a high detection rate, low false positive rate, reasonable overhead, and successful generation of strong anomaly signals in case of attacks [63], [93], and [128]. There are many methods to process system calls; our first detection system is based on a very low-demand statistical method called the bag of system calls (BoSC), which is used to monitor all of the system calls invoked by VMs.

The hypothesis behind this first detection system (BoSC) is that a normal behaviour profile can be built for VMs hosted in large-scale public IaaS environments using sequences of system calls *only*. A *strong enough* anomaly signal is generated when comparing abnormal sequences of system calls with the normal behaviour profiles that are built using a bag of system calls representation method that ignores the order of system calls and relies only on frequencies.

Collecting system calls does not require accessing VMs or installing any script on them (knowing that no instrumentation within VMs is a main requirement). System calls might also reflect the security status of the VM and useful security

[2]System calls are the interface between processes and the operating system. In cloud environments, VMs communicate with the host kernel and send requests for resources through system calls.

information can be extracted from them. Furthermore, monitoring the system calls invoked by a VM has a very limited effect on a VM's privacy.

BoSC was derived from *bags of words modelling*, which is a popular representation method in speech recognition. In BoSC, the order of the system calls is ignored, and only their frequencies are considered. Researchers in [93] have proven that the frequencies of system calls are enough to build a normal behaviour profile for processes and detect anomalies reliably.

We built a cloud network in the laboratory, collected streams of system calls from VMs (normal and malicious), and then built a BoSC classifier that we trained using normal data and tested using both normal and malicious data. We monitored the entire VM (with its operating system, applications, and services) as a single process. The system we developed works specifically for cloud environments, and we found by experiment and through hypothesis testing that it generated a strong enough anomaly signal and could detect anomalies reliably with a 100% detection rate and no false positives.

The other system we developed was based on the more computationally expensive machine learning method: the hidden Markov model (HMM). This system was designed to work in cloud environments to monitor ephemeral VMs because of its ability to work on the fly; it requires less training data and less time to be ready and start detecting anomalies.

The hypothesis behind the second detection system is that a normal behaviour profile can be built for ephemeral VMs hosted in IaaS environments using a limited number of system call sequences. A *strong enough* anomaly signal is generated when comparing abnormal sequences of system calls with the normal behaviour profiles that are built using an HMM representation method.

Ephemeral VMs are VMs that live for a short period of time (i.e. an hour or two) and perform specific tasks, after which they terminate. Ephemeral VMs and the model behind them are very popular in the cloud; they suit the cloud's nature. For instance, when an organisation needs to analyse terabytes of data, hundreds of VMs can be initiated in the cloud for an hour or two to perform the analysis. They will then terminate, which saves time and also ensures that consumers will still pay the same amount of money as they would be charged if only one VM was installed and worked for a long period of time.[3]

The intrusion detection system (IDS) used to monitor ephemeral VMs has to be able to finish training and create the normal behaviour profile using a small amount of data in a short period of time. HMM-based detection systems might require more computational power than BoSC systems, but computational power is not a scarce resource in the cloud.

The hypervisor HMM-based IDS we developed works specifically for cloud environments. We found by experiment that it consumed a small amount of data for training (about 780,000 system calls to train the classifier, which is about 19 MB) and provided a 97% accuracy, 100% detection rate, and 5.66% false positive rate [4]. Hypothesis testing was used to verify that the anomaly signal was strong enough.

[3] This is particularly true if there is enough data parallelism [11].

[4] The accuracy is the proportion of data correctly classified (true positives and true negatives). The detection rate is the proportion of malicious data correctly classified (true positives only). The false positive rate is the proportion of data that is falsely classified as malicious (false positives only).

We also used numerous ideas and tricks to reduce the size of the normal behaviour profiles and the time and computational power needed to build the classifiers and detect anomalies. The details of these ideas and tricks are discussed in detail in chapter 4.

For this research, we also studied different cloud attacks and developed a cloud-specific denial of service (DoS) class of attacks that works by misusing two of the main features of the cloud: 'over-commitment' and 'migration'.

Migration is the process of moving a running (or offline) VM from one host to another for load balancing and/or energy saving. For example, a VM might be migrated to save energy by evacuating and then turning off a low-utilised host or to create more room in an over-utilised host. The process of moving a VM from one host to another is very expensive because it consumes bandwidth and computational power, and affects the security by increasing the attack surface. During the migration, the entire VM is transmitted through the network, which gives the attacker the chance to attack the VM passively by eavesdropping for sensitive data or information. If the attacker succeeds in taking a snapshot of the VM in the transmission channel, this will give him or her the chance to attack the VM offline without raising suspicion or leaving any trace. All parts of the VM have to be migrated, including the running memory, which increases the danger of data loss and requires tools to maintain the integrity and confidentiality of the transmitted data. Furthermore, authentication and authorisation mechanisms should be in place, and the security of the source and destination servers should be considered. Finally, there should also be monitoring and alerting mechanisms for VMs during migration to detect any stealing, tampering, or dropping of the data. Migration decision mechanisms might also be a target for attackers, initiating migration orders for false reasons or cancelling a valid order.

If it is a live migration, the VM (including the running memory) has to be moved while running. Turning off the VM to migrate it would cause a disruption of the service. Ideally, migration should be done without the consumers' awareness, without the need for them to adapt or track the change in location [15]. Migration and over-commitment are essential to efficiently manage the cloud's resources.

Over-commitment occurs when cloud servers are allowed to host more VMs than they can afford, relying on the expectation that not all of the VMs will be used to the maximum at the same time [182]; an idea similar to over-booking in the airline industry[5]. A high over-commitment ratio increases the utilisation of cloud servers. However, it might cause a service level agreement violation if the provider fails to provide what it promises [182]. As we will discuss in detail later, over-commitment is a key factor for allowing elasticity and reducing cost by sharing resources.

We developed a cloud denial of service attack that targets these two features. We call this newly developed class of attacks a *"Cloud-Internal Denial of Service attack"*, or *CIDoS*. This kind of attack targets the architecture of the cloud rather than its implementation, which makes it harder to defeat. The main idea is to trigger the very expensive migration process for fake reasons. Depending on the success

[5]In the airline industry, airline companies usually offer a greater number of tickets than the seats available on the flight, relying on expectations that a percentage of passengers will miss the flight [182].

and extensiveness of this attack, it could cause a complete paralysis of the cloud. If VMs enter a continuous migration state, the entire cloud will have difficulty functioning and the availability will be heavily affected which might lead to a breach of the service level agreement (SLA) and loss of reputation and potential customers.

The attackers in a CIDoS attack are a group of co-resident VMs that communicate through covert channels. They coordinate their resource consumption rates by building on each other in a technique similar to jamming signals to break thresholds that trigger migration. The attack is actualised by a novel protocol based on a group key agreement (JKT protocol [86]) to establish a session key, packet delay variation (to synchronise the time), pulse-code modulation (to digitise the workload patterns), and leader election protocol (to choose a leader for the attack). The attack pattern is calculated and distributed using spread spectrum techniques. The attack, like with many cloud attacks, requires co-residency, a co-residency check, and covert channels for communication. The protocol has a very low complexity $O(n + T)$ for the key agreement protocol and requires the sending of a very small stream of data (less than 100 bits in the example we studied).

We then suggested some detection and prevention mechanisms for the CIDoS class of attacks that are based on calculating the workload correlation of VMs over time, which are actualised using a novel protocol. We also suggested methods of prevention and responses to the attack.

After that, we developed another attack that employs a CIDoS attack in addition to reverse engineered migration algorithms in the cloud to extract parameters that help to improve DoS attacks and make them harder to detect and defeat. We designed a stealthy and randomised probing strategy to learn the parameters and thresholds used in the process of making migration decisions in the cloud. In a CIDoS attack (without the improvement), it is assumed that the workload thresholds that need to be broken to trigger the migration are known to the attackers. In reality these thresholds are hidden, and it was necessary to design a new attack to discover them, which was the improvement of the CIDoS attack.

In the improvement, we perform sequential runs of the attack to extract and reveal these parameters and thresholds. The attack is based on CIDoS, together with statistical analysis. We will call the sequential runs of the attack *tests* because the attacker here is testing the cloud management algorithms to collect measurements. We also designed a formal model for the migration decision process, and then created a dynamic algorithm to extract the required hidden parameters. The mechanisms to extract these thresholds adapt to dynamic changes in cloud management algorithms. Revealing the parameters is a security breach in itself; furthermore, the revealed parameters can be used in a CIDoS attack to allow the malicious VMs to *accurately* generate the needed workload to cause the required effect (trigger migration). It is vital that the generated workload be no more than what is required to reduce the possibility of detection and make the attack live longer; attackers will try to keep the host on edge. Attackers can also use their understanding of the applied migration policies and algorithms to avoid being migrated, as we will see in chapter 5.

To outline the main goals, this research was guided by two main questions.

The first research question was *by monitoring only the system calls invoked by a VM and without prior knowledge or any instrumentation within the VM, are we able to create*

a character of the VM representing its normal behaviour and is this level of granularity sufficient to capture a number of relevant attack classes?

The hypothesis under this part is that a normal behaviour profile can be built for IaaS-hosted VMs by monitoring the sequences of system calls invoked by them. We assumed that VMs generate sufficient data for training, VMs are not initially malicious, and they are limited in the number of used services and applications (their behaviours are relatively consistent).

In this part, we proposed two systems. The hypothesis for the first system is that a normal behaviour profile can be built for VMs hosted in an IaaS environment using sequences of system calls, and a strong enough anomaly signal is generated when comparing abnormal sequences of system calls with the normal behaviour profiles. The normal behaviour profiles are built using a bag of system calls representation method, which ignores the order of system calls and only relies on frequencies.

The hypothesis for the second system is that a normal behaviour profile can be built for ephemeral VMs hosted in IaaS environments using limited number of system calls sequences. A strong enough anomaly signal is generated when comparing abnormal sequences of system calls with the normal behaviour profiles that are built using an HMM representation method.

The second research question is *by coordinating the resource consumption of a group of co-resident malicious VMs, can we force the cloud orchestration service to trigger a migration order?* Under this part, we hypothesise that in a large-scale public IaaS cloud, if m malicious co-resident VMs coordinate their workload consumption to break the cloud migration algorithms thresholds, the migration order will be triggered for false reasons.

1.1 Short Background

One of the best definitions of cloud computing is NIST SP 800-145 [122]; it is the de facto standard for defining cloud computing, describing its essential characteristic, service, and deployment models.

The NIST defines cloud computing as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models" [122].

Cloud computing provides the best solution for data analysis, distributed storage, and high-performance computing [15]. It also helps distributed government services, agencies, and organisations reduce their spending on IT and increase their efficiency. In February 2011, the U.S. government issued the Federal Cloud Computing Strategy that describes cloud computing as a "profound economic and technical shift (with) great potential to reduce the cost of Federal Information Technology (IT) systems while improving IT capabilities and stimulating innovation in IT solutions" [90].

IaaS is one of the cloud service models and NIST defined it as "the capability provided to the consumer is to provision processing, storage, networks, and

other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls)" [122].

Public service providing is one of the deployment models, the other being private, community, and hybrid.[6] The NIST defined a public service model as "the cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider" [122].

The service in a private cloud is provided for a single organization and could even be deployed on-premises, which makes the cloud security close to that of the traditional IT security architecture. A public cloud has a different architecture that requires rethinking security and developing new solutions [102].

A public large-scale cloud introduces new threats and vulnerabilities. One of its main characteristics is that it's accessed through the public Internet. Therefore, consumers need a working, secure, and reliable Internet connection to enjoy the service. If the there is a problem in the Internet infrastructure reliability (i.e. failure, network congestion, or attack), the cloud service will not be reliable either, from the consumer perspective [15]. Another characteristic is that cloud environments have a heavy workload that has to be processed and monitored, which requires special security tools that can tolerate a heavy workload without falling over. Furthermore, there is the problem of adapting to new service paradigms from a security perspective [148]. There are also the threats introduced by multi-tenancy (sharing the same physical host and address space); access policies and hypervisors are the only separation between co-resident VMs belonging to different organisations. Any flow in the implementation or access policies management could affect all aspects of consumers' security [15].

Large-scale cloud providers are usually a homogeneous environment; they use the same systems, configurations, and protocols. Therefore, any discovered vulnerability might cause severe damage and propagate very quickly. This homogeneity usually attracts hackers because it is worth the effort; one discovered vulnerability can be applied in many places [138].

1.2 Published Papers

As an outcome of the research findings, five research papers were published with the following titles:

1. "Detecting Anomalies in IaaS Environments Through Virtual Machine Host System Call Analysis" [3]. This paper was published in 2012 in IEEE and it discussed the first detection system (BoSC) to monitor regular cloud VMs from outside.

[6] The definition of private, community, and hybrid deployment models can be found in NIST SP 800-145 [122], we only focus on the public model.

2. "Anomaly Detection for Ephemeral Cloud IaaS Virtual Machines" [4]. This paper was published in 2012 in the Springer and it covered the detection system for ephemeral VMs.
   Chapter 4 covers the details of the previous two papers.
3. "Robust Coordination of Cloud-Internal Denial of Service Attacks" [5]. This paper was published in 2013 in IEEE. The paper discussed the details of the CIDoS attack and the algorithm we developed to coordinate the attack.
4. "Mitigation of Cloud-Internal Denial of Service Attacks" [6]. This paper was published in 2014 in IEEE and discussed mitigation strategies for CIDoS attacks.
5. "Dynamic Parameter Reconnaissance for Stealthy DoS Attack within Cloud Systems" [7]. This paper has been accepted and will be published soon in the Springer. It discusses the improvement of the CIDoS attack and methods and algorithms to extract hidden cloud parameters.
   Chapter 5 covers the details of the last three papers.

1.3 Thesis Outline

The next chapter, chapter 2, contains definitions and background information; we discuss the definition of cloud computing, the main cloud models and characteristics, and their architectures. We also discuss security standards, concerns, and monitoring in the cloud.

The state-of-the-art in chapter 3 covers the literature on IaaS cloud security in general, hypervisor-based IDSs, cloud-specific IDSs, and a survey of different attacks against the IaaS cloud, with a discussion on the mechanisms used. The proposed detection systems with the details of the experiments carried out and the evaluation of the systems are discussed in chapter 4. We also evaluate other similar systems and discuss attacks against a system call-based IDS.

In chapter 5, we discuss the CIDoS class of attacks, related literature, the threat model, possible mitigation strategies, and the extension of the attack by discovering hidden cloud parameters and using them to strengthen the CIDoS attack.

In the concluding chapter, chapter 6, we summarise the problems we discovered and the proposed solutions. We also summarise the CIDoS class of attacks and its improvement and discuss future work.

Chapter 2

Background

"As of now, computer networks are still in their infancy, but as they grow up and become sophisticated, we will probably see the spread of 'computer utilities' which, like present electric and telephone utilities, will service individual homes and offices across the country" [10].

LEONARD KLEINROCK CHIEF SCIENTISTS OF ARPANET IN 1969.

In this chapter, we provide an overview of the context of this study. The purpose of this chapter is to help the reader understand what cloud computing is (especially public IaaS, which is the subject of the study), by discussing the idea of cloud computing, its architecture, usage pattern, the user's and provider's roles, and cloud security concerns, monitoring, and best practices. Furthermore, providers' security responsibilities and area of control are discussed in detail to make the reader aware of the providers' limitations and help them understand the context of the solutions proposed in this research. The basics of virtualisation security are also covered here, in addition to background information about intrusion detection systems in general.

2.1 Cloud Computing Idea and History

In datacentres, large amounts of money are wasted on electricity, maintenance, hardware, space, and operating expenses for very low-utilised servers. The estimation of server utilisation in datacentres ranges from 5% to 20% [11]. Consolidating servers by installing more services on them is not always a suitable solution, especially if the service has a high variant workload. Usually, many services have a peak workload that "exceeds the average by factors of 2 to 10" [11]. Servers in datacentres must be able to provide the service at the peak time, which might make them almost idle during off-peak times which waste energy and money.

Servers should also be ready to provide the service during predicted times of increases in consumption, in addition to coping with unexpected jumps in consumption due to good reasons (i.e. more consumers) or bad reasons (i.e. a denial of

service attack). For instance, if a high-profile user on Twitter broadcasts a message about a service, an unexpected jump in consumption might occur, which might stress the aforementioned service hosting resources. The service should be able to cope with the situation and not falls over due to the sudden increase in consumption.

Deploying servers that are able to provision the service for peak times and both predicted and unexpected increases in workload causes a huge waste of money and an increase in carbon emissions. The more pronounced the variation between these three factors and the average utilisation, the more unnecessary the expenses and pollution.

What is more, new businesses (and new developers) usually have problems predicting the expected workload of a newly provided application or service. They might over-provision (thus wasting money, which is a sensitive issue for new businesses) or under-provision (thus losing potential customers, revenue, and reputation). Workload prediction is a very challenging but essential task; new services and applications have to be equipped with enough computation power, bandwidth, and IT staff.

The computing industry has evolved by offering very flexible computing power as a service that:

- reduces the criticality of workload prediction,
- reduces the waste of resources due to low-utilised servers, thus reducing carbon emissions,
- shifts capital expenses to operational expenses, which helps new businesses start with lower capitals,
- allows easy and fast release of machines when the job is done,
- allows easy and fast deploy of new machines when needed,
- provides backup machines as part of a disaster recovery plan,
- scales up and down based on the workloads,
- limits the loss of money if the new business has fail,
- requires less IT personnel,
- allow fast deployment of new services (because there is no need to buy and install physical servers), and
- reduces maintenance effort.

Furthermore, as stated in [11], "companies with large batch-oriented tasks can get results as quickly as their programs can scale, since using 1000 servers for one hour costs no more than using one server for 1000 hours. This elasticity of resources, without paying a premium for large scale, is unprecedented in the history of IT."

The saving will also become more feasible in extremely large-scale cloud providers due to deploying datacentres at low-cost locations and decreasing the cost of electricity, network bandwidth, operations, and hardware [11].

Microsoft Hotmail, introduced in 1996, can be considered the first software-as-a-service (SaaS) cloud computing service.[1] One of the most popular platform-as-a-service (PaaS) providers is salesforce.com, which was established in 1999 to deliver

[1] Microsoft Hotmail is an email service provided online, replacing client-server architecture mode.

enterprise applications via a simple website [143]. Amazon (started in 2006) is one of the first infrastructure-as-a-service (IaaS) providers in the market; it is currently the most popular one. Before the emergence of cloud computing, Amazon already had the required infrastructure to make a large-scale public cloud provider. Then Microsoft, Google, VMware, and other blue-chip companies followed Amazon and started providing public cloud services.

The national institute of standards and technology (NIST) defined cloud computing in *Special Publication 800-145* [122]. Their definition has become the de facto definition of cloud computing. The first draft of the definition was submitted to the international committee for information technology standards (INCITS) as the U.S. contribution [58]. It took the NIST more than two years and over 15 revisions of the definition to reach the final version.

2.2 Cloud Computing Definition

The NIST defined cloud computing as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models" [122].

2.2.1 Cloud Characteristics

Cloud computing, according to the NIST, has five main characteristics. First, consumers should be able to increase the used resources based on their demand automatically with almost no human intervention from the provider side - *'on-demand self-service'*. Second, the service should be able to accessed through the network using any type of device, such as tablets, laptops, and computers - *'broad network access'*. Third, cloud resources are pooled and used by different users concurrently (multi-tenancy); the user might choose the location or region for its machines but without being aware of the exact location - *'resource pooling'*.

The fourth characteristic is *'rapid elasticity'*: consumers can increase and decrease the used resources on demand and sometimes automatically. Resources should be available to consumers at any time and in any quantity; they should appear unlimited to consumers. The last characteristic is *'measured service'*: consumers pay only for what they use - pay-per-use model [122].

Some of these characteristics have long been part of the computing industry (such as *on-demand self-service* and *broad network access*) and so the security risk they pose is well known. Other characteristics, especially when combined with other cloud features and characteristics, are posing rather new security challenges. One such characteristic is *resource pooling* in a public IaaS, which introduces a wide range of threats caused by *multi-tenancy*. In particular, if the service is shared among consumers from different organisations, some of them might be malicious users or even competitors. If there is isolation insufficiency between co-resident users, malicious ones might attack their co-residents. For example, they can steal their

confidential data, analyse their behaviour, monitor their transactions, and interrupt their service by conducting a denial of service attack.

The other characteristic that poses a security challenge is *rapid elasticity* when combined with *self-service* and *automation*. There is a threat of attackers increasing the bills of other consumers by forcing them to request more resources (expand) for false reasons, fake requests, or incomplete requests of the service.

Roman *et al*. in [74] defined elasticity as "the degree to which a system is able to adapt to workload changes by provisioning and deprovisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible." There are a number of published research papers discussing the *automation* of elasticity in the cloud, which is when the cloud expands automatically without any human intervention from either sides (cloud's clients and cloud administrators). Today, the automatic elasticity feature is not activated in many cloud providers. However, we believe that it is essential and has to be activated with the development of the cloud when it becomes more mature; solutions should be developed to deal with the risks posed.

Cloud computing, according to the NIST, has three service models: software-as-a-service (SaaS), platform-as-a-service (PaaS), and infrastructure-as-a-service (IaaS). In the SaaS model, cloud providers offer their application to users through the network. Users can access the application using thin clients and browsers or program interfaces designed to communicate with the other applications hosted in the cloud.

PaaS model are offered for applications developers as a development environment in which to host and support developers with libraries, services, supported tools, networks, and storage. In IaaS, providers offer the needed infrastructure and resources to host consumers' machines (virtual machines). IaaS providers offer computing power, storage, networks, and any other supporting resources to host VMs. Each host in the cloud IaaS model is occupied by a number of VMs sharing the resources; the VMs are isolated from each other by the virtualisation layer. If the hosting service is offered to only one organisation exclusively, the deployment model in this scenario is a *private* cloud. The infrastructure in the private model could be either on-premises of the organisation or off-premises. The *public* deployment model is when the cloud service, infrastructure, and resources are offered to different organisations and individuals to share [122].

There is also a third deployment model: the *community* cloud deployment model. In this model, the service is offered to a specific community, like health care, where organisations belong to this community share the same cloud and the same concerns, policies, regulations, etc. [122]. The last deployment model is the *hybrid*, where more than one deployment model is used. The deployment models are bound to each other using technologies that enable data exchange and application portability [122].

Sometimes cloud providers offer mixed service deployment models to support each other, such as Microsoft Azure, which offers mainly PaaS models and supports it with an IaaS in a way similar to *value chains*, in which organisations deploy a series of activities to add more values to the offered service [123] and [142].

IaaS, PaaS, and SaaS all represent an abstract level, with IaaS being the lowest abstraction level. IaaS can provide a service to PaaS and SaaS; PaaS can provide a service to SaaS but not to IaaS; and SaaS cannot provide a service to PaaS or IaaS

[123].

2.3 Cloud Usage Patterns

There are three common scenarios for using cloud service. First, there is the simple scenario, in which a provider provides the service directly to end users. Second, the hybrid service is when the end user consumes different cloud services from different providers, concurrently. Third, the value chains are when different providers collaborate to add values to a service or when one provider deploys more than one cloud service model to add values to its end users' service [123].

2.3.1 Stakeholders

Stakeholders are the players in cloud usage scenarios. The first player is the cloud service provider, which could be either a native provider or a non-native one. A native provider is one that owns the cloud infrastructure, while a non-native provider rents the infrastructure from another cloud service provider. Non-native providers are also called *intermediary* providers because they consume cloud services and provide cloud services to end users. End users are also stakeholders and can be individuals, applications, enterprises, or governmental institutions [123]. There are also cloud partners, third parties, and service brokers.

In the public cloud, there is a problem of trust between different players. Consumers are not sure if providers will maintain the security of the cloud properly, have a disaster recovery plan, and protect consumers' data and VMs. Consumers cannot even be sure if providers are accessing their data or not. On the other hand, providers cannot be sure whether a consumer is innocent or a hacker, and whether they are using the the service properly or for malicious purposes.

Since there is a problem of trust between consumers and providers, regulatory compliance and service level agreements (SLAs) are needed.

2.3.2 Regulatory Compliance

Regulatory compliance is needed to ensure that providers are following the best security practices and deploying the needed security controls and processes to protect the infrastructure and consumers' data. Furthermore, regulatory compliance is essential to reducing the risk inherent in relying on cloud services.

2.3.3 Service Level Agreements (SLAs)

A service level agreement is the key accountability agreement between a cloud consumer and its provider to gain assurance of risk mitigation. It has to specify the minimum level of quality of services (QoS), such as level of availability, response time, and error-rate. For instance, response time is critical in some businesses (i.e. voice processing). Such businesses have to guarantee the minimum response time before moving the service to the cloud, which is specified in the SLA. There are two types of SLAs, internal and external. Internal SLAs specify the acceptable QoS between a consumer and a provider belonging to the same cloud, while external SLAs are between two different organisations.

Due to the flexibility of the cloud, there might be a future need for dynamic SLAs. Dynamic SLAs can change automatically with consumers' needs and requirements, unlike general SLAs that describe entire classes of services [35].

Still there is a problem of SLA compliance, assurance, and violation reporting. The performance and compliance (i.e. the response time or error rate) should be checked continuously to guarantee that the QoS level is maintained. Furthermore, there is a need for regulatory compliance evaluation, which is usually performed by a trusted third party. There is also a need for a continuous risk assessment to ensure continued acceptable performance in the face of new security threats. Security monitoring is a main part of the risk assessment and management process. Continuous monitoring is essential to detecting violations and to predicting violations before they occur. These tasks should be performed automatically without human intervention to suit the cloud model.

For availability, providers usually determine the minimum level of availability they guarantee in the SLA (i.e. Amazon guarantees 99.95% availability in a year). Downtime due to scheduled maintenance is not considered in the calculation of the overall downtime for the SLA.

2.4 Who Uses the Cloud

The best candidates to move to the cloud are businesses with an unsteady workload and varied demands on the IT services. For example, if a large amount of computing power is needed only once a week or just during Christmas time, then moving the service to the cloud might save money and effort, and require smaller IT teams. Businesses with unknown in-advance demand are also good candidates for the cloud [11]. New small-to-medium organisations can launch their services faster in the cloud with less initial capital expenses. The reduction of capital expenses due to the use of cloud computing might cause an increase in the operational costs. However, it could also cause the opposite (a reduction in the operational costs) if the business nature is suitable for cloud computing (i.e. has a highly varied workload or unknown in-advance demand). Businesses with limited technical resources (i.e. IT experts personnel) are also good candidates for cloud computing.

In contrast, large and old organisations with legacy infrastructure might find it hard to migrate to the cloud due to lack of interoperability. Furthermore, the cost of migration might be very high to some businesses (i.e. companies handling critically sensitive data) to the extent that cloud computing is not a choice. Regulatory compliance restrictions sometimes prevent some organisations from benefiting from the cloud. For example, U.S.-based cloud hosting usually has a problem with European and Canadian companies due to the Patriot Act, which gives the U.S. government the right to access hosted data; this leads to a potential privacy breach [18].

2.5 Cloud Providers

Providers for the public cloud should be ready with a massive infrastructure and massive pool of computing resources (networks, storage, servers, and services). They also have to be ready with the experience, tools, and IT personnel to manage

the resources. Experience in maintaining security and deploying security control are essential to protecting the cloud and its consumers. Providers are responsible for providing a secure multi-tenant environment. Access policies, data protection, authentication, identity management, access control, accountability, forensics support, policy integration, and privacy control all are essential to providing a trustworthy cloud service; providers have to be able to provide these security mechanisms. Large responsibilities and high demand of experiences allow only blue-chip companies, such as the very popular online retailer Amazon.com, to offer large-scale public IaaS cloud service.

2.6 Cloud Security Standards, Best Practices, and Guidelines

There are three major issues slowing businesses' adoption of cloud services: trust, interoperability, and the cost of migration.

To deal with the trust problem, some form of evaluation is needed to help consumers decide if they can trust the provider and help them calculate the risk introduced by moving to the cloud. Security standards are essential to provide this evaluation, which helps providers prove that they have sufficient security controls. Furthermore, government agencies require cloud services that can ensure security and compliance.

Standardising cloud environments is also essential to ensuring interoperability and ease of integration. There are many general IT security standards; however, *cloud security* standards are often missed. If the cloud is going to be thought of as a utility, an industry-wide cloud security standard has to be developed.

2.6.1 Cloud Security Alliance (CSA)

When the cloud was first introduced, non-profit organisation, Cloud Security Alliance (CSA) developed a cloud security best practices. Almost all major cloud providers (such as Amazon, Oracle, RedHat, and Salesforce) are members of the CSA.

The CSA published a report of nine top threats to cloud computing with recommendations and guidelines [9]. They also developed a compliance standard called *cloud control matrix (CCM)*, which provides standard security controls that can guide providers and help consumers in the assessment of the risks associated with a provider [42]. It also provides a detailed framework for cloud information security and covers the area of operational risk management, cloud threats and vulnerabilities, security control requirements, normalisation of security expectations, cloud taxonomy and terminology, and security measures.

2.6.2 National Institute of Standards and Technology (NIST)

The NIST's Special Publication 800-95 *Guide to Secure Web Services* is for web services in general, but is also useful for cloud computing [102]. They also have more recent publications designed specifically for cloud security. In addition to the NIST definition of cloud computing [122], the NIST published Special Publication

800-144 *Guidelines on Security and Privacy in Public Cloud Computing*, which covers public cloud security and privacy challenges, outsourcing recommendations, cloud threats, and technology risks and safeguards. The guidelines are designed for consumers to help them evaluate the security of providers [85].

2.6.3 ISO 27017 and ISO 27018

The ISO is publishing two cloud security standards in 2015: ISO 27017 and ISO 27018. The new standards are to apply 27001 and 27002 to cloud computing. They will also provide guidance to ensure that providers offer suitable security control and maintain the privacy of their customers. The standards also consider other ISO standards such as ISO/IEC 27031 on business continuity and ISO/IEC 27036-4 on relationship management [84] and [1].

2.6.4 IEEE P2301

IEEE project P2301 guide for *cloud portability and interoperability profiles* (CPIP) is a guide aiming to help cloud providers and consumers "in developing, building, and using standards-based cloud computing products and services" [83]. CPIP offers recommendations to providers and consumers in choosing standards-based applications, interfaces, file formats, operation conventions and other areas [83].

2.6.5 ENISA

In 2012 ENISA published a practical guide called *'Procure Secure: A Guide to Monitoring of security service levels in cloud contracts'* designed for the procurement and governance of cloud services. It provides advice regarding security monitoring in the cloud and potential indicators for transparency during service delivery [57].

2.7 Virtualisation

Virtualisation is the key enabling technology for multi-tenancy that plays a significant role in cloud computing. In virtual environments, there are three main components: hypervisors or virtual machine managers (VMMs), virtual memory managers, and virtual machines (VMs) [102].

The hypervisor is the software responsible for creating and managing VMs. Each VM might have a different operating system and, in a public cloud computing scenario, each of them might belong to a different organisation. Providing resources to each VM and maintaining the isolation between them are the hypervisor's main tasks. Some hypervisors interact directly with the hardware, this type of hypervisor is called *type 1* or *native* hypervisor. Other hypervisors work within the host operating system; such hypervisor is called *type 2* or *hosted* hypervisor [102]. Most of the recent hypervisors require CPU with virtualisation extensions (Intel VT or AMD-V) to perform well.

Virtual memory managers are responsible for managing memory requests made by the host operating system and VMs. Allocating pages, page tables, and page directory tables are some of the virtual memory manager's responsibilities.

A virtual machine (VM) is a piece of software that acts like a physical computer by running operating systems (guest OSs) and applications on them. VMs' operating systems and applications run on virtual devices that mimic actual physical devices. In virtualised environments, more than one VM can run simultaneously in the same physical host (multi-tenancy). They are constantly created, migrated, and terminated.

Providers can offer:

- computation resources such as CPU cores and speed, memory, and network rate,
- storage resources in different size and types,
- system services such as virtual router, DHCP, and DNS, and
- network service.

2.7.1 VM Lifecycle

To create a VM, the consumer has to select or upload an image (ISO file) and specify the CPU speed, number of cores, amount of memory, and disk space. The consumer also has to choose the zone (location) of its new VM, after which the host will create and run the new VM. Then, the client can run, stop, reboot, and terminate its VM. VMs can also be migrated from host to host depending on the cloud migration policy; however, clients should not be aware of this migration.

To move from a VM state to another, the cloud has to perform a number of operations and tests. For instance, to terminate a VM, the cloud has to track all containers in memory, interactions between VMs, and dependencies with other VMs in the cloud [102]. Consequently, each VM operation has a varying cost depend on the complexity and imposed risk of the operation.

VM migration is one of the most costly VM operations in the cloud. It is a very complicated task because VMs have to be migrated without interrupting the service or creating a noticeable latency; even the running memory has to be migrated while running. Migration operation also consumes a large amount of resources and increases the attack surface and security risk. VMs have to be protected while migrating in an environment that can be considered hostile. Because understanding migration operations are essential for this research, they are discussed in detail in chapter 5.

For security purposes, clients who own VMs hosted in the cloud are advised not to mix services in one VM. As stated by Krutz *et al*. in their book [102], "while contemporary servers and virtual machines are adept at multi-tasking many functions, it's a lot easier to maintain secure control if the virtual machine is configured with process separation. It greatly complicates the hacker's ability to compromise multiple components if the VM is implemented with one primary function per virtual server or device." The cost model in the cloud, usage-based pricing, supports this idea. Instead of creating one large VM with many functions, the client can create a number of small VMs, each with one primary function and still pay the same cost.

2.7.2 Virtualisation Security

Complexity and security are inversely proportional; more complexity means less security. Transferring real environments to virtual ones adds layers of complexity to systems. Therefore, virtualisation usually has a negative effect on security. The hypervisor code is usually small in order to limit the newly introduced attack surface; however, the threat and vulnerabilities still exist. Examples of attacks against hypervisors are Blue Pill (Xen attack), Cloudburst (exploit that allows VMware guest to escape to the host), SubVirt, and DKSM [138] and [117].

There are number of security concerns associated with virtualisation:

- virtualisation hides some of the network activities (traffic between co-resident VMs) from the network-based intrusion detection systems,
- the use of vulnerable images to create new VMs,
- the use of default VM images without changing the default passwords and configurations,
- security patch management (i.e. sleep VMs that require security patches),
- resource sharing among VMs (CPU, clock, memory, network, and hard disk) introduces the possibility of a number of attacks such as VM-to-VM attack, starvation attacks, and covert channel attacks,
- VM escape attacks (when a malicious VM escapes from the isolation and controls the host), and
- kernel rootkits in virtual environments, which are hard to detect with regular IDS because of the difficulty of extracting meaningful information from the available low-level data that are collected from virtual resources [91].

2.8 Cloud Architecture

The large-scale public IaaS cloud system can be divided into front-end and back-end. The front-end is the interface for consumers. Consumers can use different types of devices and thin devices to access the cloud. They can access the service through the Internet from anywhere in the world. To use the service, consumers might need special applications in addition to authentication credentials, such as passwords and encryption keys. If cloud consumers (clients who own VMs hosted in the cloud) are themselves service providers, there will be another layer - the consumers of cloud consumers' services.

Back-end is the cloud system that involves networks, physical servers, storage systems, and management machines. The cloud consists of a collection of regions. Under each region there is a collection of locations, and each location has a number of hosts. Consumers' VMs live in hosts and each VM has at least one operating system and a collection of applications and services. Third parties usually exist to provide services such as auditing, forensic analysis, and integrity checks.

2.9 Security Concerns

There is a lack of security monitoring tools in the cloud. Cloud providers usually deploy performance monitoring tools to provide administrators with the ability to

monitor the performance of servers and VMs to identify the areas of poor experience and improve them. These tools also help to ensure availability in addition to providing the means to charge consumers based on the resources consumed. However, performance monitoring tools cannot fulfil the requirements of security monitoring due to a lack of security monitoring and reporting built-in functions; additionally, they cannot inspect VMs' internal states [190].

The ability to detect and report malicious activities inside VMs is critical for securing the cloud by providing measurements and continuous risk assessments.

The regular security goals of confidentiality, integrity, and availability must be met in the cloud as in any other computing environment. Old threats exist in the cloud too, such as illegitimate modifications of data, eavesdropping, and searching and indexing on encrypted data that might reveal sensitive information to cloud administrators. Therefore, regular security defences have to be deployed and managed in the cloud. examples of such defences include firewalls, intrusion detection systems, data encryption, and authentication and authorisation mechanisms.

New threats are induced by the use of virtualisation and sharing physical resources among public cloud service consumers, such as VM-to-VM attacks and covert channel attacks [28] and [181]. Other threats induced by some of the cloud's main characteristics include massive scalability and maintaining the security of VMs during migration [167] and [180]. There are some threats that come from the nature of VMs, such as handling the security of offline VMs, VM snapshots, and VM rollback to a compromised state or affecting the freshness of cryptographic keys [28].

Data in the cloud travels long distances through the Internet, which increases the threat of data loss, eavesdropping, and unauthorised modification of data. Furthermore, there is the threat of different types of DoS attacks. Recently, new DoS attacks have been developed specifically to work on the cloud by misusing cloud features and characteristics [167], [164], [5], and [180]. Lastly, one of the main challenges facing cloud computing is coping with new service paradigms [148].

2.9.1 Security Monitoring

Proving the security state of the whole cloud and each of its components is a tough problem facing providers [35]. Monitoring is essential to meeting many security requirements and providing service assurance. By security monitoring, providers can collect information that helps them reflect the security status of cloud systems, discover vulnerabilities, know which security controls are weak or missed, and then react accordingly.

Main cloud components (i.e. network traffic, host systems, virtual machines, applications, and processes) have to be monitored. The data collected from monitoring these components are usually raw and low-level. Due to a mixing of services, a mixing of clients, and high data volumes in the public cloud, making sense of and extracting useful security information from these low-level data is a hard task. First, collecting and analysing data then extracting information and reporting (when needed) have to be automatic and without human intervention because manual interaction with data in such a tense mixed environment as the large-scale public cloud is not feasible. Second, the monitoring tools have to cope with high volumes of data without falling over. Third, monitoring tools have to work on the

fly, providing instant monitoring, detecting, and reporting. When designing cloud monitoring tools, providers should:

- assume that the public cloud environment is *hostile* [138],
- design monitoring tools that are tamper-resistant,
- consider not posing overhead,
- design tools with enough generality to monitor different VMs [191],
- consider coping with cloud special features (i.e. VM migration), and
- consider sharing information with consumers or third parties when needed [70].

To extract information from collected data, events from different monitoring tools must be correlated. However, maintaining central events analyser or co-relater in large-scale distributed flexible environments would not be efficient for many reasons: data volume will be excessively large; a signal point of failure will be created; a bottleneck will be created; sending data through the network to the central analyser might consume the bandwidth; and distances and large volumes might delay the process of analysing and responding. All these special requirements are imposed on new monitoring tools that are designed especially to work on cloud environments.

Although monitoring tools collect large amounts of raw data, there are still dark yet critical inaccessible areas which is the area inside VMs. Therefore, developing efficient VM monitoring tools can improve the security of cloud computing and reduce risk exposure. There is a gap in VMs monitoring area as shown in the state of art chapter, chapter 3, where we conducted a survey of available VM monitoring tools in literature.

2.9.2 Security Responsibilities

Security responsibilities differ according to the used service model. In SaaS, consumers are only application users. They are allowed (in some cases) to change certain configuration parameters of the used application. Consumers are responsible for securing and protecting their passwords, encryption keys, and any authentication credentials used for the SaaS service. Applications, operating systems, hosts, networks, and the underlying infrastructure are under the full control of providers [118] and [24].

In PaaS models, providers have control over operating systems, hosts, networks, and the underlying infrastructure, while consumers have full control over the application, including its code and the security [118] and [24].

In IaaS, providers have control over the virtualisation layer, network, and the underlying infrastructure while consumers have full control over their VMs (the used operating system, applications, services, and security controls). Providers have no control in the area inside VMs. Having limited control does not imply less responsibility. For instance, in IaaS, while providers have almost no control about what is inside VMs, they are still responsible if one of these VMs is compromised and starts attacking other co-resident VMs, the infrastructure, or the outside world. Having no control, makes security design and achieving security goals and assurance harder tasks for providers [118] and [24].

The mix between control and responsibilities in the IaaS model has created an unclear vision for third parties who are responsible for security monitoring and incident reporting [70]. For instance, the third party who is responsible of security monitoring has to decide to whom the incident should be reported; if a malicious traffic has been detected, should the third party inform the provider or the consumer? [70]. If traffic with signs of anomalies has been detected in the large-scale IaaS cloud network, it is hard to identify the true source of the malicious traffic because of resource pooling, the large number of VMs, and large volumes of data.

In some service models, and especially in IaaS, providers might need to enable their consumers to access data sources relevant to some incidents to help them diagnose security vulnerabilities, thus improving the security of their virtual machines [70]. This emphasises the problem of trust. Do providers meet their obligations on reporting security incidents that have an impact on consumers' virtual machines?

A number of solutions can be used to deal with this problem. First, security reporting and the access possibilities for data sources can be included in the SLA [70]. Second, a security system with an interface for incidents and events data exchange can be designed to automate the process of event monitoring and reporting, a system that considers consumers' rights to access some of the relevant data in a timely manner [70].

Furthermore, a trusted third party for security monitoring and reporting is usually suggested as a solution for trust problems. The last solution to be mentioned here is offering security as a service to consumers, which might imply allowing providers' security systems to have a better vision for what is inside VMs. This would provide better security by enhancing the ability to extract meaningful security information from low-level data, or bridge the *semantic gap*. However, providing better vision for providers might increase their legal responsibilities in addition to slowing the monitoring process, especially in large-scale clouds where there are hundreds of thousands or even millions of virtual machines.

2.10 Security Monitoring

Monitoring the cloud's overall security status, in addition to monitoring VMs' behaviour, is essential for providers to fulfil their security responsibilities (which include protecting the infrastructure, preventing VM-to-VM attacks, and preventing attacks on the outside world using the cloud network). Intrusion detection systems are the most widely used tools for security monitoring; they are discussed in this section.

2.10.1 Intrusions

An intrusion is any unauthorised activity on the network, data, or computer systems, i.e. unauthorised access, data theft, and denial of service attacks. Intrusions might lead to security incidents. A computer security incident, according to NIST SP 800-61 [146], is any "violation of computer security policies, acceptable use policies, or standard security practices," such as leaking sensitive data through a file-sharing service.

The NIST categorised incidents into five categories: denial of service attacks, malicious code (i.e. viruses), unauthorised access, inappropriate usage, and multiple components (incidents that contain two or more incidents) [146].

Incident handling:

One of the primary domains in the information security management process is incident handling. It is one of the main domains in ISO 27001 in addition to security policy, human resources security, and business continuity management [61]. Incident handling usually has the following steps: detection of possible incidents, analysis, containment, recovery from the incident, and continuous improvement [70]. Intrusion detection and prevention systems are useful for detection, analysis, and sometimes containment of incidents. There are different types of IDSs: host-based and network-based (or HIDS and NIDS), anomaly-based and signature-based.

Host-based IDSs are for monitoring computer systems; network-based IDSs are for monitoring network traffic. The anomaly-based detection method works by building a normal behaviour profile for the monitored host and then measuring any deviations from normality. If it exceeds a certain threshold, an alarm for intrusion will be raised [92]. In the signature-based detection method, intrusions are detected by comparing events with known signatures of intrusions [92].

Signature-based IDSs usually have low false alarms rate for known attacks, but they require accurate defined rules. The main disadvantages of signature-based IDSs are that they cannot detect unknown attacks and the difficulty of updating and adding new rules for detecting new attacks. Anomaly-based IDSs can detect novel attacks if they change the behaviour of the host. However, it usually suffers from high false alarms rate [92].

In this research we focus on monitoring VMs using HIDS based on anomaly detection methods. We concentrate on anomaly-based HIDS because it can detect novel attacks and does not require accurately defined rules, which are hard to create, hard to update, and require a deep understanding of the attack's technical details. Furthermore, as explained, anomaly-based HIDS has the ability to create a normal behaviour profile that represents the VM if it has a relatively consistent behaviour, as we will discuss in detail in chapter 3 and 4.

2.10.2 Anomaly Detection

To identify intrusions in hosts, an intrusion detection system is used to inspect the host's activities and detect ones that do not conform to the expected behaviour. The detection system has to be trained to be familiar with the host's normal behaviour (and sometimes abnormal behaviour) so it can recognise any deviation. Data collected from the host are normally used to train the detection system. The choice of training method is heavily affected by the available data for training and for detection.

The detection system has to go through several phases:

1. Collect data for training: collect live data, data generated by experiments, real systems, or import already collected data from other research. The accuracy and detection rates rely heavily on the accuracy of the collected data. Generating quality data for training is critical and experience in computer systems is needed to be able to build the system with the right configuration [128].

2. Train the classifier: the training could be supervised, semi-supervised, or unsupervised. In supervised anomaly detection, only labelled data are used for training (normal or abnormal); in semi-supervised, only normal data are labelled, while in unsupervised there are no labels based on the assumption that most of the data are normal and anomalies are rare [25].
3. Features and patterns extraction: in this phase, the collected data has to be processed to extract features of them that represent the data and to be used later for detection. The extracted features are then used to train the classifier. Many methods are available to choose and extract features. Some of them are based on statistical analysis and others on machine learning techniques or data mining approaches. The choice of method depends on the available data for training, system requirements, and the available resources.
   In statistical approaches, the classifier generates a normal behaviour profile representing the data statistically, and then the variances of the monitored new data from the normal profile are constantly generated. For detection, the system compares current usage to the saved profile and reports any statistically significant deviation (which might be a potential attack) as abnormal [134]. Examples of statistical approaches are given by Jyothsna *et al*. in [92]. They include the following: threshold metric (simply counting the number of events occurring in a specific period of time and raising an alarm if they are less or more than a specific threshold); Markov process (examining the state of the system at fixed intervals and calculating states probabilities); mean and standard deviation; multivariate model; and time series (time and value of observations are stored, then when a new observation arrives, the probability of its occurrence is calculated and an alarm of abnormality is raised if the probability is low).
   Machine learning techniques are based on *intelligent* algorithms that are used to learn and recognise complex patterns in data then make decisions. Machine learning algorithms normally focus on finding and extracting relationships from data [92].
   There are many mature machine learning algorithms, such as neural networks and the naive Bayesian classifier.
   In neural networks, there are a number of nodes that produce non-linear functions. The input of each node is either from the source or from another node. The general function is modelled by the network, which consists of a set of very complex inter-dependencies and the characteristics of nonlinear systems with feedback [92].
   In naive Bayesian, one scan of data is required to train the classifier. The classifier applies Bayes' theorem with the strong assumption that attributes are conditionally independent. Each attribute consists of a number of values and the classifier estimates the class-conditional probability of each value. For classification, the classifier computes the posterior probability for each class [60].
   The output of the classification function is either label or score. In label, each monitored activity is given a label of being normal or anomaly. In score output, each monitored activity is given an anomaly score. More details are in chapter 3.

In the next chapter (the state of art), we discuss from literature IaaS cloud computing security in general, hypervisor-based IDSs that are used to monitor VMs, and intrusion detection systems that are designed specifically to work in the cloud. We also survey cloud attacks (especially DoS attacks), the techniques they rely on, and the exposed vulnerabilities.

Chapter 3

State of the Art

> "It is a security nightmare and it can't be handled in traditional ways" [120] in the description of cloud computing.
>
> J. CHAMBERS, CISCO SYSTEMS' CHAIRMAN AND CEO,

This chapter will explore the literature that is relevant to cloud intrusion detection systems, host-based cloud intrusion detection systems, hypervisor-based intrusion detection systems, and detection systems in the cloud. We will also discuss the importance of intrusion detection systems in a large-scale public IaaS cloud. Furthermore, we will discuss in detail the special *requirements* for cloud monitoring systems that have to be met to work efficiently in our environment and will show how previous research failed to satisfy these requirements, which created a need to develop new detection systems that satisfy them. We believe we have accomplished this in the proposed systems. We developed two hypervisor-based detection systems to work in a large-scale public IaaS cloud. These new detection systems significantly close the gap in the literature that we discovered, as revealed in this chapter. We will also explore DoS attacks that are designed specifically to work in the cloud and introduce our novel architecture-based cloud DoS class of attacks.

Section 3.1 will discuss the security challenges faced by a large-scale IaaS cloud. Discussing these challenges is essential to our research because it leads to an understanding of the need for monitoring and determining the threats and requirements. In this section, we also explore the challenges of security control in the cloud, VM monitoring, denial of service threats, IDS taxonomy, IDS in the cloud, HIDS in the cloud, and HIDS considerations to suit cloud environments.

The literature on hypervisor-based IDS in general is discussed in section 3.2. Cloud intrusion detection systems are discussed in section 3.3. The last section, section 3.4, covers the literature on cloud attacks, which are specifically designed to work in cloud environments.

3.1 Cloud Computing Security

As stated by Yanpei *et al*. in [40], "security has emerged as arguably the most significant barrier to faster and more widespread adoption of cloud computing." Since 2009, many research papers and reports in academia, government organisations,

and industry have emphasised this fact such as in [11], [121], [165], [102], [40], and [163]. As a result, the security of cloud computing has become a main topic in most security conferences. A study in 2012 showed that about 35% of *r*esearch about cloud computing from 2007 to 2012 involved the security and privacy of the cloud [68], see figure 3.1.

Category	# of papers
Security & privacy	66
Infrastructure	46
Data management	15
Interoperability	12
Legal issues	9
Economic issues	9
Service management	8
Quality	8
Software	6
Trust	6
Other	5

Figure 3.1: About 35% of research in cloud computing from 2007 to 2012 was about security [68].

Most of the papers from 2007 up until now are about the security risks of using cloud computing, security challenges, and countermeasures.

Researchers have also presented numerous security benefits that can be delivered by using cloud computing, as Garrison stated in his book [67], by "removing local databases from unmanaged computers and from laptops that can be lost or stolen" and by taking "security out of the hands of what could be under-skilled IT workers in some instances." These benefits are mostly for small business, as stated by Krutz *et al*. in their book *Cloud Security: A Comprehensive Guide to Secure Cloud Computing*, [102], "Smaller companies with minimal IT departments often see a bigger security return by using cloud infrastructure offered by a public cloud service provider than large, better funded organizations with complex infrastructure. Large established companies have a bigger investment in traditional IT and its accompanying security architecture, whereas a newer company can more readily employ the security features offered by a CSP."

Availability is a main concern in cloud computing; providers periodically publish reports revealing service outage such as in [159]. Any service disruption event

can cause significant damage to the provider's brand reputation, affect consumers' productivity, and potentially cause the loss of existing customers. Moreover, the news of the event will spread quickly via IT news websites, magazines and blogs. Availability is one of the main factors defined by a service level agreement (SLA), in addition to the response time and error-rate [35]. Any service outage incident problem might cause an SLA violation. SLA violations directly affect the revenue of the cloud service provider (CSP) because of the penalty scheme for violations and indirectly by affecting the provider's reputation, as shown by the two famous Amazon S3 and EC2 incidents that affected availability in 2008 and 2011, see [161] and [160].

3.1.1 Security Control in the Cloud

Cloud security responsibilities are shared between providers and consumers;[1] security responsibilities are distributed differently in each cloud model [156]. In SaaS, the cloud is almost under the full control of CSP, which bears most of the security responsibilities; consumers are only application users and they are responsible for managing their authentication credentials properly and using safe application configurations and modes of operations [70] and [156]. In PaaS, applications are under the control of the consumers, and application security is one of their responsibilities. Providers are responsible for maintaining the isolation between consumers' applications and controlling the runtime environment and infrastructure [70] and [156].

In IaaS, consumers have full control over their virtual machines (VMs). They can freely choose the operating system and applications used; they also might choose a highly vulnerable application and never apply security patches. Providers have almost no control at all in this area, but they are still responsible for network security, protecting the infrastructure, and protecting VMs from each other [70] and [156]. A highly vulnerable VM in the IaaS cloud could affect the whole cloud; once this VM is breached, the attacker could use it to attack other collocated VMs, which may belong to other consumers from different organisations. The attacker may also attack the cloud infrastructure or perform a denial of service attack (DoS), which might cause a service outage and SLA violation. Providers have no control over the hosted VMs, but they are *still* responsible, which makes security harder to achieve and more complex, as stated in [181] "it is more error-prone for normal users to manage a whole virtual machine (IaaS) than just a single piece of software (SaaS). Therefore, security threats under IaaS model deserve more attention." Moreover, in the IaaS model, server administration commands (even critical ones) come through the network (possibly the Internet), which increases the security burden on providers [178].

Other security issues should also be considered in the IaaS model, as stated in [181], such as "multi-jurisdictional issue, redundancy checking, privacy-preserving calculations, and hardening virtualisation environment" [181].

In [163], researchers conducted a deep analysis of cloud security challenges; they based their research on the threats introduced by cloud security alliance (CSA)

[1] Cloud consumers are usually service providers themselves and they share their part of the security responsibilities with their consumers or users (the consumers of cloud consumers services)

[9]. The technologies used in the cloud are not unique. Therefore, it faces security threats that are similar to those of other computing environments. However, the collection of technologies and concepts has also created new threats and vulnerabilities [163]. The security concerns discussed in [163] are that cloud environments have heavy workloads that have to be processed and monitored. They are also multi-tenant environments, and providers have to protect tenants from each other, in addition to protecting the cloud infrastructure from malicious tenants [163]. Researchers have also suggested countermeasures to improve the security of the VMs and network in the cloud such as establishing "zones of trust in the cloud, the virtual machines must be self-defending, effectively moving the perimeter to the virtual machine itself" [163].

In [126], Molnar *et al*. divided the threats into two categories: threats created because of moving from owning to leasing and threats that exist because of resource sharing. They investigated a large number of threats and suggested countermeasures for each of them. For instance, under those resulting from leasing, there is a contractual threat, with cost-overrun attacks under this threat, where the victim consumers pay the cost of answering attackers' malicious (and fake) requests. This type of attack is discussed in more detail later, in section 3.4. As a countermeasure to cost-overrun attacks, Molnar *et al*. suggested allowing consumers to set boundaries for the VM expansion to control the cost (limit their elasticity)[126].

3.1.2 Security Challenges in the Cloud

The special combination of technologies and features in the cloud also introduces new security risks, leading to new security requirements. There is a need for new or improved security mechanisms in addition to regular security defences such as intrusion detection and prevention systems, encryption, firewalls, and access control policies and mechanisms [102] and [163].

In the cloud market today, providers offer many monitoring tools such as Amazon CloudWatch [172][2] and LogicMonitor [113][3]. Most of the tools offered provide the service of performance and resource consumption monitoring for VMs and cannot be used for security monitoring. As Zou *et al*. states, "performance monitoring typically provides means to charge cloud users based on the resources consumed by their VM instances", and they take "the whole VM as the monitoring target and does not consider the internal state of the VM, therefore applying existing performance oriented monitoring solutions is not enough for the requirements of security monitoring" [190]. Providers offer these tools to help consumers control the cost.

3.1.3 Special IaaS Cloud Characteristics, Requirements, and Threats

The most obvious cloud characteristic is the *high volume of data*; it is one of the main characteristics of the cloud that has a direct effect on the choice and design of security measurements and tools [163]. The reasons for having higher than usual amounts of data in cloud environments are as follows:

[2] Amazon CloudWatch is a tool that allows consumers to monitor their usage of cloud resources and offers load balancing [191].

[3] LogicMonitor is a tool to discover newly added or deleted VMs and monitors them [191].

- First, the utilisation of servers is increased to save power, which reduce the cost and the carbon footprint. Reducing cost by consolidating servers means more activities. Thus, more traffic is generated.
- Second, the cloud infrastructure and the hosted VMs are accessed by many parties concurrently, including providers (cloud administrators and systems), consumers, users of consumers' services, third parties and possible malicious parties (i.e. attackers, worms, and/or viruses). Having many parities accessing the service concurrently increases the amount of data flow to, from, and within the cloud network.
- Third, the migration feature of the cloud requires the transfer of large amounts of data; for example, in live migration, the VM has to be transferred to another host while running (without interrupting the service). Even the memory has to be transferred while running bit by bit along with network connectivity, which generates more traffic [16].[4]
- Fourth, the continuous and automated creation of snapshots and backups of VMs, which are usually stored in different geographical areas, as part of business continuity management, generates large volumes of data.
- Fifth, data and commands to and from VMs travel long distances through the network and most probably the Internet in the case of a public cloud; graphical outputs, mouse clicks, and even keyboard keystrokes come from long distances, which generate a greater data flow.
- Sixth, in the model of ephemeral VMs, the process of initiating these intense short-lived VMs, their tense workloads, and terminating them after a short period of time contributes to the expansion of the data volume in the cloud.

Having high volumes of data, especially if combined with the requirement of working on the fly, demand special security solutions, i.e. distributed security systems to be able to monitor and control the high volume of traffic. Fast automated monitoring and acting on the fly are essential to suit the dynamicity of cloud environments; for instance, a consumer can initiate a large number of VMs (i.e. hundreds), use them for an hour or two to accomplish a specific task (i.e. analyse data or generate reports), and then terminate them. Ephemeral VMs are popular in the cloud, and they require security systems that can be built and work on the fly.

Another cloud characteristic is the *openness* of the environment; a public cloud offers the service (automated provisioning) to almost anyone that can pay the cost, i.e. businesses, governments, individuals, and possible attackers [181]. Different clients initiate VMs with different operating systems and applications require flexible security systems with adequate generality to be able to work effectively in the diverse cloud environment [191]. Furthermore, the environment should be dealt with as hostile, as stated by Rhoton in [138] "applications must therefore be much more secure in a potentially hostile environment than they would need to be on a private network."

In addition, since all commands and data (even highly critical and sensitive ones) are coming through the Internet (in the case of a public cloud), strong confi-

[4]There are many approaches to performing live migration such as *pre-copy migration*, where memory pages are copied from the source to the destination without interrupting the work of the VM, and then transferring dirty pages. At some point the VM should be stopped for an ideally unnoticeable short period of time and transferred to the destination [26].

dentiality mechanisms are required. Encryption can protect data during transformation from being read by other users or attackers, but there is no guarantee that data are not being accessed by cloud administrators, especially since the data have to be decrypted at some point in the cloud to be useful [102] and [148].

Multi-tenancy is a defining attribute of the cloud to increase server utilisation; a number of VMs share the same physical host (co-resident), and each of them might belong to a different client. The cloud provider (in all of the cloud models) is responsible for maintaining the isolation between co-resident VMs [163]. Virtualisation is the technology that allows multi-tenancy. Although, popular virtualisation applications (hypervisors) on the market today, like Xen, VMware, and KVM, are continuously improving the isolation mechanisms between VMs, there are still leaks, as we will see later, in section 3.4. Multi-tenancy with different clients introduces new threats and vulnerabilities [28]; VM co-residency, as Vaquero *et al*. said "increases the attack surface and risk of VM to VM compromise" [163]. VM to VM compromise is also called inter-VM attack, and it usually happens through the shared resources, i.e. by using side channels [28] and [181].

Malicious VMs might attack other benign VMs, and they also might attack the cloud infrastructure. Some virtualisation features, such as rollback, can be abused and threaten VMs, and thus the cloud. For instance, a VM can be rolled back to a compromised or unpatched state [28]. Moreover, a rollback and snapshot might affect the freshness required in cryptographic protocols [28]. Another threat was suggested by Zhang in [181]. He called it a "Threat to Dormant"; he said that even if VMs are turned off, they are still available because they are saved in the cloud system, which is online most of the time. He stated that an "offline VM is not equivalent to an off powered computer at home" [181]. Thus, offline VMs can be attacked. In addition, attackers might copy them or even snapshot them and attack the copy offline to extract some data or information from them [28].

Another threat to the cloud is *VM escape*, where a malicious VM successfully controls the host or part of the host, if the VM can bypass the hypervisor and interact with the host directly. As Reuben said in [137], this might result in a complete "breakdown in the security framework of the environment." Elhage, in [53], published an example of a VM escape attack in a KVM hypervisor. They exploited a bug in KVM (CVE-2011-1751) [44]; KVM receives unplug requests from guest hardware and executes them before checking whether they support being unplugged or not. These unplugged devices might leave behind corrupt states or dangling pointers that allow attackers to execute malicious code, as presented in [53].

Although virtualisation has negative effects on security, some researchers have suggested using virtualisation to address security problems [132] and [66]. Many research papers and books have been published in this area [79], [132], and [66]. The properties of virtualisation that make it attractive for security are its isolation and flexibility. By using virtualisation, you can easily isolate infected virtual servers so they do not affect other virtual servers or the host, as Hoopes said in his book, *Virtualization for Security*, "isolate compromised portions and OS instances by denying them the very resources they rely on to exist. CPU cycles can be reduces, network and disk I/O access severed, or the system halted altogether. Such tasks would be difficult, if not impossible, to perform if the compromised instance was running directly on a physical host" [79]. Therefore, virtualisation increases the de-

fence system's ability to respond. Isolation has also been used by many researchers in building attack-resistant host-based intrusion detection systems (HIDSs). The most popular one is virtual machine introspection (VMI) [66], which we will discuss in detail in section 3.2.

The flexibility of managing and maintaining VMs is another appealing feature of virtualisation that can be used in providing security. Organisations can use virtualisation in disaster recovery by creating backup virtual servers instead of spending large amounts of money on physical backup servers that are idle most of the time. These backups are activated in the case of failover, which saves money, management effort, physical space, and power [79]. Virtualisation can also be used in sandboxing, honeypotting, and helping in forensic analysis.

3.1.4 Patch Management in the Cloud

Providers in that IaaS model have no control inside VMs, which makes patch management a great problem in the IaaS cloud. To maintain security, applications should be kept in updated states with the most recent patches. However, there is no guarantee that consumers are following through and updating [124]. The researchers in [124] suggested *virtual patching* as a solution to this problem. Virtual patching is a layer over the application to prevent the exploiting of known vulnerabilities [17]. It is like an application firewall that filter the application's *ins* and *outs* that are related to a known vulnerability until the patch is available and tested. It is only effective for known vulnerabilities that do not yet have patches or have untested patches. In contrast, we are here discussing users who fail to follow best practice security policies by ignoring the need to update their applications and operating systems with available patches. Patch management in the cloud is also a problem under different circumstances:

- if a VM is offline for a long period of time and is thus outdated,
- if a VM is rolled back to a state before patching,
- if an unpatched VM is cloned to create an image that is used in future to create other VMs, and
- if an image (i.e. in a public repository) is in an unpatched state.

Kang and his colleagues in [59] investigated the problem of patching in the cloud and designed a system called offline patching scheme (OPS) for patching. OPS searches for out-of-date images and patches them (even the offline ones). However, it targets *images* rather than *VMs*. Users with unpatched VMs are not discovered by this system. Regarding offline images, they patch them in three ways: mounting the image and then patching it; inserting the updating script into the image to be run when the image is booted next time; or by using the *Script Rewrite Method*, which is a method to analyse the patches and apply only the essential part, see [59] for the details. As stated, this might solve part of the problem: the problem with unpatched images, but not unpatched VMs.

3.1.5 VMs Monitoring

To deal with malicious or unpatched VMs, there is a requirement for monitoring VMs from inside. If these out-of-date VMs are breached, attackers might attack

other VMs (inter-VM attacks), attack the hypervisor (VM escape attacks), attack the infrastructure, and/or attack other parties outside the cloud, which might cause legal liability for the provider. As stated in [191], "to accurately observe, distinguish and report the malicious and benign behaviours inside VMs is a critical stage for providing measurements to secure the clouds." Some researchers have suggested installing agents inside VMs for monitoring, such as in [13]. This leads us to question the acceptable level of instrumentation in VMs.

Installing an agent or a piece of code inside VMs can lead to a number of negative effects. First, collecting and analysing detailed data from inside all the VMs might affect the efficiency of the used security systems; large amounts of data will be collected, which will increase the power and time needed to analyse them. This might prevent the security system from working on the fly and live detection which are essential in the cloud. Second, legal and contractual restrictions might limit the permitted level of intrusiveness. A high level of intrusiveness might also cause an increase in the providers' accountability. Moreover, an SLA sometimes requires the absolute minimum amount of surveillance from providers. Allowing providers to install code in consumers' VMs presents another problem involving trust in the cloud.

Should consumers trust providers? Are there any guarantees about the security of the service? Do provides comply with security standards? In reality, consumers distrust cloud providers because there are currently no trust-supporting mechanisms in the cloud, as stated in [191].[5] To deal with the trust problem, third party auditing might be required. However, as stated in [148], the "privacy of data from a third party auditor is another concern of cloud security." Allowing providers or third parties to enter VMs will emphasise the trust problem and increase the threat. Therefore, providers should reduce the level of instrumentation within VMs to the minimum possible. There is a need to reduce the instrumentation while simultaneously verifying the security status of the cloud, which are necessities for both providers and consumers.

3.1.6 The Problem of Infected Images

Images are used to create new VMs. These images are usually stored in *public repositories* or the *Cloud App Store*.[6] A public repository allows users to share and exchange images; any user can upload an image (publisher) and other users might use it to create their own VMs (consumer).

Sharing images imposes security threats to the cloud, see figure 3.2 (this figure is from [28]). First, a publisher might upload an infected image with malware or a Trojan horse. Usually, providers scan the repository for viruses and rootkits to reduce the threat. However, this does not eliminate it completely [28]. Second, the publisher might (by mistake) upload an image that contains sensitive information, i.e. cryptographic keys or passwords [28]. Bugiel *et al*. in [28] showed that the threats introduced by publicly shared images are still serious and no longer theoretical.

[5] In Zou *et al*.'s paper, [191], they discussed the problem of trust regarding the provided quality of service (QoS). However, if there is a problem related to trust, it might affect the security, in addition to performance monitoring.

[6] Cloud App Store is the name of Amazon's public repository

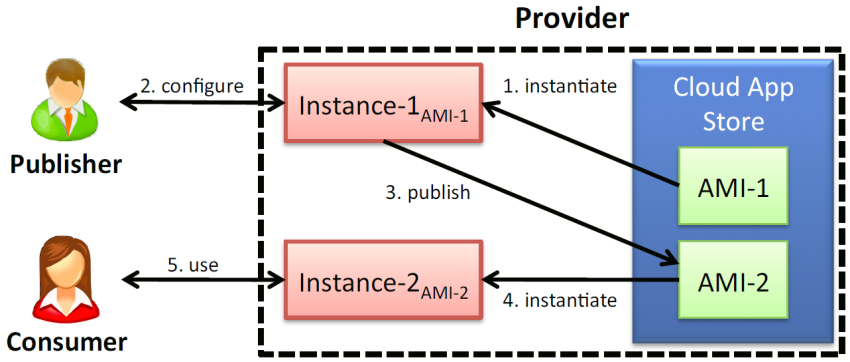

Figure 3.2: Uploading infected images in shared repositories [28].

3.1.7 Vendor Lock-In

Another security concern is locked in to a provider. For example in the case of provider bankruptcy, if there are no data interoperability standards among providers, consumers will be in a data lock-in state [148]. Garrison, in his book [67], suggested managing this problem contractually, in addition to choosing a well-known provider. He also warned that providers might increase the cost if they have a lock on the consumer business.

3.1.8 The Threat of Denial of Service

Three main features of a public IaaS cloud make it more vulnerable to DoS attacks and distributed DoS attacks (DDoS) than a private IaaS cloud or regular on-premise IT environments. The first is the openness of the public IaaS cloud architecture; the service is offered to everyone through the Internet. Because of this openness, malicious consumers can easily gain access to the cloud to perform DoS attacks. Furthermore, a DoS attack can be performed on the consumers' side or the providers' side; an example of a consumer side DoS attack is blocking the consumers' access to the cloud service or blocking their users' access to a cloud-based consumer service. An example of a provider side attack is disturbing the cloud service or affecting the fairness when distributing resources.

The second feature that makes the public IaaS cloud more vulnerable to DoS attacks than private cloud is the multi-tenancy of the VMs belonging to different clients sharing resources. Attackers can easily become neighbours and share resources with benign consumers. They might then penetrate them and steal their data or resources. Isolation between VMs is provided by the hypervisor; despite the effort to improve this isolation, many attacks in the literature succeeded in discovering and exploiting vulnerabilities in the isolation layer in different hypervisors (see section 3.4).

The third feature that emphasises the risk of DoS attacks in a public IaaS environment is the on-demand elasticity. We found from the literature that the misuse of this feature occurs by performing economic-loss attacks, where innocent consumers are forced to pay more for fake reasons, i.e. the economic denial of sustainability attack (EDoS) which is discussed in [77], [152], and [167]. Another example

of an economic-loss attack is when attackers' VMs gain more resources by stealing them from their innocent neighbours (co-resident VMs), i.e. a resource freeing attack (RFA), which is discussed in [164]. The details of these attacks are in section 3.4.

These attacks affect the availability of the service, although their goal is financial gain rather than disturbing the service. For instance, if the targeted consumer has inactivated the *automatic* elasticity (to fix the cost) it will turn into a regular DoS attack, as discussed in [180]. These attacks can also be categorised as starvation attacks, which are DoS attacks. Regular DoS attacks, i.e. Smurf attack and SYN flood attacks, are also threats in the cloud.

To be able to detect DoS attacks and monitor the behaviours of consumers' VMs to detect and prevent any penetration of cloud defences by malicious consumers, malicious cloud administrators, and/or malicious outsiders, intrusion detection systems (IDSs) are required. An IDS is responsible for monitoring, inspecting, and generating early warnings of any suspicious activities. Nowadays, an IDS is considered a main security tool in network and computer systems. It is an integral part of an incident response policy and is widely used to detect DoS and DDoS attacks.

3.1.9 IDS Taxonomy

An intrusion detection system (IDS) and intrusion prevention system (IPS) are two different security tools that are closely related to each other. An IPS is sometimes called an *active IDS*, whereas the regular IDS is a *passive IDS*. An IDS reacts to intrusions by raising alarms, while an IPS reacts more actively, i.e. disconnecting suspicious connections. An IPS is an IDS with an extension, which is the active reaction [188]. Here, intrusions could be unauthorised access, malware, data theft, and/or DoS attacks.

An IDS is used to detect intrusion attempts, whether they succeed or not, before, while, or after the attempt [80]. After detection, the IDS should generate an alarm, which is supposed to lead to an investigation, followed by a decision to dismiss or react to it. The investigation, decision making, and response processes could be manual or automated [80]. Furthermore, some IDSs can be used to collect forensic evidence [87]. The data collected by an IDS can be sent for analysis in a central IDS (centralised IDS) or can be analysed at the same place as the collection (distributed IDS) [188].

IDSs can be categorised in several ways. Mian, in [188], drew a tree to show different IDS categories, see figure 3.3.

IDSs can also be classified based on the methods used for detection: signature-based and anomaly-based. In a signature-based IDS, the system scans traffic or data for known attack patterns. In an anomaly-based system, the system creates a normal behaviour profile for the traffic or data and detects any deviation from it. The nature of an anomaly-based IDS makes it more suitable to detect new intrusions and attacks such as zero day attacks [188].[7] For anomaly-based IDSs, many approaches are used to create normal behaviour profiles, including data mining ap-

[7] Zero day attacks are attacks against unknown vulnerabilities (not known to the public but found by attackers). Defending against this type of attack is hard because the vulnerability is not known. Thus, it cannot be detected by antivirus applications and cannot be patched [22].

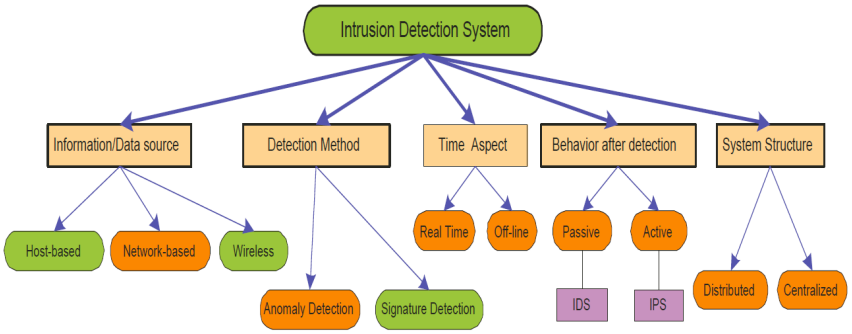

Figure 3.3: IDS Taxonomy [188].

proaches and statistical approaches. More information about anomaly-based IDS approaches is given in chapter 4.

The most popular IDS types are host-based IDS (HIDSs) and network-based IDSs (NIDSs). An HIDS provides protection inside a server by monitoring its activities, i.e. log files, system calls, and/or critical files [188]. A NIDS protects the network by monitoring the traffic [188].

The literature reports less popular types of IDSs such as mobile agent-based IDS.[8] The idea of a mobile agent IDS involves dispatching an agent to hosts to examine the data instead of collecting them using a central IDS. These data might have large volumes, which consume high bandwidth if the data are being sent through the network to the central IDS [145]. There are many other ideas for Mobile Agent IDSs. However, it is not a good candidate to perform security monitoring in the cloud due to problems with reliability, scalability (it cannot deal with a large amount of data and a large number of VMs), and lack of attack-resistance [95].

3.1.10 IDS in the Cloud

Kholidy *et al.*, in [95], said that a "traditional NIDS and HIDS cannot identify suspicious activities in a cloud environment." A NIDS faces numerous challenges in the cloud. The first is the problem of monitoring and analysing encrypted traffic, knowing that all the traffic between consumers and their VMs and between nodes in cloud environments are encrypted [80] and [95]. Second, co-resident VMs might communicate within the host, and a regular NIDS will not be able to monitor in-host communications [73].[9] There is also a common NIDS problem when implemented in switched environments [102]. Switched environments will not allow NIDS sensors to inspect all the traffic because switches work based on connections, and the sensor will only receive packets addressed to it (unlike a hub that echoes

[8] Mobile agents are small, autonomous, and intelligent pieces of code (programs) that have the ability to travel, learn, communicate within the network or between hosts to perform specific tasks set by the agent developer [65].

[9] This is especially true in a case where co-residency is a feature offered by providers to help consumers that have time critical businesses, i.e. financial business. These consumers can request to host their VMs at the same physical host to reduce communication time, and they pay more for this service. Furthermore, virtualisation sometimes allows communication between co-resident VMs through sharing the clipboard and memory, if needed to increase efficiency [91].

every packet). In his book [102], Krutz discussed a number of solutions to deal with the problem of cloud NIDSs in switched environments. The first solution is configuring the switch to span the data from a specific port to the IDS port. However, he said, "unfortunately, some switches cannot be guaranteed to pass all the traffic to the spanned port, and most switches allow only one port to be spanned at a time" [102]. The second solution is deploying hubs between switches, between routers and switches, and between servers and switches. This solution will guarantee that data will reach IDS sensors for inspection. However, as Krutz argued, this solution "spells the beginning of the end for the switched network, and removes the benefits of a switched solution" [102]. Furthermore, as Jie *et al*. concluded in [73], "the communication port of the switch may be changed when virtual machines are migrated to a different physical server. Therefore, traditional NIDSs are no longer applicable in intrusion detection on the IaaS data centers" [73].

A NIDS suffers from more problems, including the problem of attacking the NIDS sensor itself and the problem of coping with large volumes of data in the cloud. With an increase in the data volume, the NIDS might be saturated, which usually heavily affects the IDS performance. It will require more time to store and analyse data, and some packets might be discarded, which will generate high error rates [102]. Moreover, in a recent analysis of the current cloud computing security measurements in [49], researchers said that the regular network defences in the cloud will not be able to detect intrusions, but an indication of their existence will be derived from anomaly detection systems or other intelligence sources. These are the challenges facing a NIDS in monitoring the cloud network. However, intrusions inside hosts are outside the NIDS scope. Thus, inter-VM attacks and VM to infrastructure attacks cannot be detected by a NIDS [80] and [102]. An HIDS is required to cover this area. An HIDS and NIDS are complementary systems. Cloud NIDS restrictions and challenges emphasise the importance of using an HIDS in the cloud.

3.1.11 HIDS in the Cloud

An HIDS can be anomaly or signature-based. A signature-based HIDS is effective for known attacks that have known signatures. Many antivirus applications can be considered to be signature-based HIDSs [80]. An anomaly-based HIDS is used to detect deviations from the normal host behaviour, and it is effective for unknown attacks or attacks that have unknown behaviours.

We found that there are three different models of HIDSs available that can be deployed in the cloud:

- Model 1: an HIDS that is implemented in the cloud host to monitor its activities (critical file changes, critical system calls, log files, application activities, and system integrity); this is a regular HIDS.
- Model 2: an HIDS that is implemented in the hosted virtual machine. It collects data from virtual resources but deals with them as real resources; VMs users (clients) see it as a regular HIDS.
- Model 3: a hypervisor-based IDS that is implemented in the cloud host to monitor its tenants (VMs). This IDS is isolated from the monitored VM, which

make it more attack-resistance and gives it relatively good visibility. The IDS sensor in this type of HIDS can be deployed directly on the host system itself (in the virtualisation layer) or in a dedicated VM. More about this type of HIDS is in section 3.2.

3.1.12 HIDS Considerations to Suit Cloud Environments

For an HIDS to work in the cloud, we conclude that it should consider the following:

- cope with high volumes of data in the cloud,
- work on the fly,
- avoid a "single point of failure" [95],
- "detect intrusion initiated from within a VLAN" [91],
- protect the IDS sensor itself from tampering or being disabled in the case of compromised hosts or compromised VMs [95] and [91],
- monitor all the VMs and their hosts in the cloud,
- minimise the level of intrusiveness without requiring installations or modifications inside VMs (for legal and efficiency reasons as stated in section 3.1.5),
- "have a flexible architecture to be applied to several cloud architectures" [95],
- "consider different service models and user requirements" [95],
- adapt to the scalable and distributed nature of the cloud, i.e. VM migration [95],
- does not require any prior knowledge from inside VMs,
- flexible and not centralised,
- does not require any interference to be able to work effectively (automated), and
- detect unknown novel attacks.

The dilemma in cloud security monitoring is that IaaS providers should deploy detection systems that are able to perform efficiently with little or no knowledge about what is inside customers' VMs. Providers should not require any knowledge about a VM's structure, memory content, OS, and applications [70].

To cope with the scalable and distributed nature of the cloud, researchers suggest using distributed IDSs (DIDSs) such as in [166], [71], and [95]. Kholidy *et al.* stated in [95] that "only a distributed strategy may be appropriate."

In literature, we found that regular HIDSs are faced with three main challenges:

- First, the monitoring tools suffer from *low attack-resistance* due to the fact that they are installed in the host being monitored; once the host is compromised, the attacker can simply disable the monitoring tools [104], [115], and [148]. Virtualisation is often used to solve the problem of low attack-resistant by isolating the HIDS from the monitored host, moving the monitored host to a VM and installing the HIDS in the host, such as in [66].
- Second, the monitoring tools might start *dropping data when overloaded*. This problem is emphasised in an environment with a high data volume such as a cloud. To cope with high volumes of data, a distributed IDS is often used [166].

- Third is the challenge of bridging *the semantic gap* between the collected data and the effect that this data has, or will have, on the monitored system (extracting useful information from the data) [110]. HIDS usually suffers from the semantic gap problem; however, the problem becomes more severe in a hypervisor-based IDS. The semantic gap in this type of HIDS is between the hypervisor and guest VMs [111]. Many approaches are used to deal with this semantic gap, such as *Trap and Inspect* and *Checkpoint and Roll Back* [66], [12], and [110]. In both approaches, the IDS sensor is located in the host or in another dedicated VM, and to provide the needed information, it injects a piece of code in the monitored VMs (owned by consumers) [12], [110] and [66]. This solution might be acceptable in a private cloud, or in a trusted virtualised environment, but not in a public cloud where the environment should be dealt with as hostile, and a lack of trust exists among parties. Further information about the approaches and solutions to the semantic gap is discussed in chapter 4.

As stated in section 3.1.11, there are three types of HIDSs: a regular HIDS in the host to monitor the host, regular HIDS in consumers' VM to monitor the VM, and hypervisor-based IDS in the host or a dedicated VM to monitor the consumers' VMs. The first and second types of HIDSs are regular ones and do not meet the requirements for an HIDS in the cloud. A hypervisor-based IDS has good potentiality for meeting the requirements and tackles the dilemma of efficiently monitoring consumers' VMs with little or no knowledge about what is inside them. A hypervisor-based IDS has high attack-resistance due to the full isolation from the monitored VM. It also has the required power to control VMs because the host runs with the highest operating system privileges (Ring 0) which allows it to clone, suspend, and restrict VMs [110]. Working with a high privilege mode gives the hypervisor-based IDS the capability of prevention in addition to detection. Furthermore, a hypervisor-based IDS clearly has better visibility than a NIDS [110], because it can at least inspect all the *ins* and *outs* of every hosted VM. For these reasons, more research has recently been done on hypervisor-based IDSs. However, it still has to deal with the semantic gap problem. We think that a hypervisor-based IDS might also help providers gain a better sense of the overall security picture of the cloud. It might also help them to detect any malicious VM before it starts attacking the infrastructure, other VMs, and/or other parties outside the cloud. By monitoring VMs, the provider will be able to evaluate the security situation of each VM, which will help in forming feature decisions regarding renewing or modifying contracts with some consumers, and/or notifying them to improve the security condition of their VMs.

3.2 Hypervisor-based IDS

A glimpse of the use of virtualisation in security was provided by Goldberg in 1974 [69]. He started his paper by saying:

"Virtual machines have finally arrived. Dismissed for a number of years as merely academic curiosities, they are now seen as cost-effective techniques for organizing computer

systems resources to provide extraordinary system flexibility and support for certain unique applications."

He suggested using VMs to improve software reliability; in that relatively innocent old digital world, by software reliability, he meant not allowing errors in multiprogramming applications and operating systems to affect other operating systems and applications. The virtualisation feature used here to provide security was *isolation*.

After that, most of the research on virtualisation was about using it in server consolidation, testing applications, fault tolerance, and the use of legacy applications [27] and [29]. In 2001, a hypervisor-based IDS was suggested by Chen and Noble in their paper *"When virtual is better than real"* [39]. They suggested moving the IDS from a real machine to a virtual one as a solution to the low attack-resistant problem. They also suggested an IPS called *clone-based intrusion prevention*, which works by cloning the monitored VM and then testing the detected suspicious events on the copy rather than the original to decide whether to allow these events to reach the original VM or not. The virtualisation features they used were isolation, portability, and easy control (clone, encrypt, and migration).

The challenges discussed in [39] were the performance overhead and semantic gap. Chen *et al.*, in [39], said that to fully bridge the gap, "one must re-create this information in some form." In the proposed model, VMs can be accessed by the monitoring tool to collect data that are not acceptable in a cloud model. Chen and Noble said that three services will take advantage of moving to virtualisation: "secure logging, intrusion prevention and detection, and environment migration" [39]. They argued that trusting the hypervisor is similar to trusting a real processor because hypervisors are small and provide few and simple services. They concluded by remarking that VMs "offer the potential for improving both intrusion prevention and intrusion detection" [39], which proved to be true in the following research.

A year after this, another paper was published [52] with an application called *ReVirt*. This applied the same idea of moving to a VM to isolate the monitoring tools from the monitored host, which makes them more tamper resistant. However, the security tool here was not an IDS but a *system logger*. They targeted two main problems in regular system loggers: the low attack-resistance in the case of a compromised system and the lack of completeness to fully understand the occurring incidents. ReVirt's main job is to monitor and log all events in the VM. Thus, to overcome the lack of attack-resistance problem, ReVirt converted the host to a VM; and to overcome the lack of completeness problem, it used techniques from the backup and recovery field, i.e. "checkpointing, logging, and roll-forward recovery" [52]. By using these techniques, *ReVirt* can replay the whole VM execution bit by bit. However, this has a high cost on the overall performance. In ReVirt model, VMs are accessed by the monitoring tools in the host which is an unacceptable level of intrusiveness for cloud environments.

One of the most popular hypervisor-based IDSs is *virtual machine introspection (VMI)*; in [66], Garfinkel and Rosenblum designed VMI to monitor and detect anomalies. The system was designed to improve regular HIDSs by making them achieve better attack-resistance. They converted a real server into a VM and monitored this VM. The IDS was also isolated from the hypervisor and deployed in a

separate VM dedicated to monitoring. VMI monitors hardware state such as memory pages, register contents, and I/O device flags. It requires prior knowledge and data from within the VM, i.e. the operating system structure and implementation, to deal with the semantic gap. Therefore, different operating systems require different policies and use different libraries. Researchers also added some response abilities to the IDS (intrusion prevention abilities), which allow it to suspend, resume, reboot, and checkpoint the monitored VM. The paper described in detail the relationship between the hypervisor and the monitoring VM. The proposed system also required modification to the used hypervisor (VMWare) to be able to collect data from VMs and communicate with the IDS. The VMWare was modified and tested. However, the researchers claimed that the same modification could be applied to most of the available hypervisors. After modifications, the IDS can access the internal states of the VM and examine them. For instance the IDS can perform queries such as, "give me a list of all the processes currently running on the system, or tell me all the processes which are currently holding raw sockets" [66], and "show me the contents of virtual memory from x to y in the context of the login process, or display the contents of task structure for the process with PID 231" [66].

Bryan, in [132], criticised VMI because it provided a very low-level memory view, which caused difficultly in extracting any meaningful information from the collected data. Furthermore, we think that the level of intrusiveness of VMI is unacceptable in public cloud environments for efficiency and legal reasons, as we explained in section 3.1.5. Most of the hypervisor-based IDSs that came after Garfinkel *et al.*'s were based on VMI, such as the hypervisor-based IDS proposed in [88].

The systems we developed and tested in this research were hypervisor-based IDSs that monitor VMs from outside and isolate the IDS from the monitored VMs to make it as tamper resistant as VMI. VMI converts a real machine to a virtual machine for monitoring purposes, while in the cloud, using virtualisation to allow multi-tenancy is essential and not a choice.

The main difference between our systems and VMI is that we comply with the security monitoring requirements of a large-scale public cloud. Our systems consider detection in real-time, building small profiles to reduce the performance overhead and reduce the building and detection time. Small profiles are also essential because, in the case of VM migration, the profile also has to be migrated with the corresponding VM. Furthermore, we require no prior knowledge, require no instrumentation within VMs, and monitor system calls only; in contrast, of VMI operates by observing hardware states such as physical memory pages and registers based on a prior knowledge of the VM structure. VMI also uses a modified version of VMware, whereas we utilised plain KVM.

Payne *et al.*, in [131], designed another hypervisor-based IDS that provides active monitoring, while VMI provides passive monitoring. In passive monitoring, the monitoring mechanism is not able to stop the attack before it happens, it only scan what is already happening. In active monitoring, the execution of the monitored system will be interrupted if a certain condition occurs, and the control will be passed to the security tool [131]. A hook was placed inside the VM. If the execution reached the hook, it would be interrupted, and the control would be passed to the security system. The execution inside the monitored VM would be trapped and transferred to the IDS VM for analysis. For that reason, their system is an IDS

and IPS.

Laureano *et al.*, in [104], implemented an idea similar to VMI. The HIDS was separated from the monitored host to protect it from tampering. They modified the hypervisor User-Mode Linux (UML) to be able to extract data (i.e. system calls) from inside the monitored VMs. In general UML is not a competitive hypervisor in the market however they used it for testing the suggested architecture, and because it is open source, it can be modified. The communication between the hypervisor and monitored VM internal processes was through named pipes. To register system calls, sliding window from [63] was used. In the learning phase, all the processes with correspondent users were recorded to generate an access-control list (ACL). In the detection, the sequences of system calls were checked, in addition to the processes and users in the ACL. An idea similar to VMI was applied with the improvement of adding an ACL and more concentration on response mechanisms. We think that, in Laureano *et al.*, there was a lack of technical details and descriptions of the methods used.

Another hypervisor-based system is *HyperSpector* from [101]. HyperSpector detection system, that is a HIDS and a NIDS, is deployed in a VM dedicated to monitoring. It monitors network traffic *to* and *from* the monitored VM, the integrity of the file system, and the behaviours of processes in the monitored VM. To be able to collect these data from inside, the VM maps the monitored VM file system and processes to the IDS VM. Therefore, the hypervisor can access VMs and is aware of their interior. They also assumed the ability to work as part of a distributed IDS architecture [101]. If HyperSpector was used in the cloud, it would threaten the privacy of consumers and would require a high level of intrusiveness.

Jin *et al.*, in [89], developed *VMFence* which is a distributed intrusion prevention system for distributed virtual environments and prevention implies detection. The system was not dedicated to cloud environments. However, we are convinced that it has great potential to work efficiently in the cloud because it was designed for distributed and highly flexible virtual environments with the consideration of multiple hosts and multi-core CPUs. VMFence provided network monitoring services only (NIDS). It captured all network traffic to and from each VM by monitoring the virtual bridge in the host and without installing any code inside the monitored VMs or requiring any prior knowledge. In this system, there is a dedicated monitoring VM that runs numerous detection processes, each of which is responsible for monitoring only one VM. With the arrival of any VM (new or migrated), a new process is launched. There is also a main process that analyses traffic and distributes it among its children (detection processes) based on the MAC address. Each detection process has detection rules that are continuously updated to suit the corresponding VM. These rules (configuration file) are migrated with the VM in the case of migration. The prevention involves updating firewall policies in the monitored VM and/or notifying the administrator to suspend or terminate the VM under attack. The system was implemented and tested on Xen and was shown to provide a low drop rate, high performance, and the ability to respond in real-time. The system was designed to work in highly flexible environments: environments that support migration. Furthermore, it can cope with high data volume. However, the researchers did not discuss one of the most important challenges facing a NIDS in such environments: dealing with encrypted traffic. In addition, the detection

rate was not covered in the research.

3.3 Cloud Intrusion Detection Systems

Since 2009, in academia, many security systems have been designed to work specifically in cloud computing environments. Dastjerdi *et al.*, in [45], proposed a mobile agent-based IDS architecture for cloud environments. The system has four components:

1. An IDS control centre (IDSCC) which manages and controls all the mobile agents in each subnet of VMs.
2. Investigative mobile agents (IMA), which are responsible for collecting evidence from within VMs in the case of incidents for further analysis and auditing. They also perform the task of correlating data to detect distributed attacks.
3. A mobile agents agency, which is a piece of code installed in each VM that is responsible for hosting and executing mobile agents and protecting the underlying VM.
4. Static agent detectors (SAD), which are installed inside VMs and are responsible for monitoring the VMs, collecting data, i.e. log files, in addition to other tasks. In SAD, different types of IDSs can be used, i.e. any NIDS or any HIDS. Furthermore, the SAD can be designed to be application specific to reduce the CPU load and the amount of data collected.

As we can see, the previous architecture requires the installation of code inside consumers' VMs, and mobile agents have to enter the VMs. For these reasons, we believe that this architecture might suit private clouds but is much too intrusiveness for public cloud environments. We also doubt the efficiency of using a mobile agent-based IDS in extremely tense environments such as a public cloud (they are tense in terms of the workload and high flexibility). Jin *et al.*, in [89], also criticised the use of a mobile agent-based IDS in virtualised environments by stating that "the configuration and management of mobile agent platforms on numerous hosts in a large-scale distributed system is extremely hard and it is difficult to assure correctness" [89]. The public cloud is not just a *large-scale distributed system* but is also an open environment, with automated provisioning for almost everyone and different clients with different operating systems and applications in an environment that should be dealt with as hostile, as we discussed in section 3.1.3.

In [140], a distributed network and host-based IDS architecture were developed to work in cloud environments. There are three components in the system. The first component is a central management unit to manage and control all the IDS sensors and perform correlations between alerts. The second component consists of IDS sensors attached to consumers' VMs to access, collect, and analyse data from within these VMs. These sensors can be any type of IDS sensor (HIDS and/or NIDS). Consumers are allowed to configure their IDS sensors, explore detected attacks, and choose the countermeasure. The sensors are connected to the central

management unit. The last component consists of IDS sensors to monitor cloud hosts and cloud infrastructure. They are also connected to the same central management unit. The alerts are designed using the intrusion detection message exchange format (IDMEF).[10] Providers can check alerts in the central management units which make the detection of large-scale attack relatively easy because of the correlation between the alerts raised from different sensors. Providers can also automate countermeasures, i.e. shutdown hosts in case of attack, which converts it to an active detection system (prevention system). The management units are allowed to read and modify the configuration of all the sensors under their control (VMs sensors and hosts sensors). Because the sensors can access and collect data from within the VMs, again this system might suit the private cloud but not the public. It also has a single point of failure, which is the central management unit. Furthermore, having a central management unit makes it unsuitable for tense environments such as the cloud.

Mazzariello *et al.*, in [119], designed and tested two different scenarios for a cloud-specific NIDS. The first NIDS was deployed close to the cluster controller (a server that manages a collection of node controllers, each of which controls a collection of hosts).[11] Researchers in this paper tested the proposed architecture using Eucalyptus (cloud management software), snort (a signature-based NIDS), and session initiation protocol (SIP) against flooding attacks targeting hosts.[12] The second scenario involved deploying a NIDS for each physical server. The attacks were successfully detected in both scenarios. However, the IDS in the first scenario was heavily overloaded, which might give attackers the chance to deliberately overload and thus disrupt the detection before starting the real attack. In Mazzariello *et al.*, the researchers only tested the detection of a DoS attack, but did not mention how to deal with encrypted traffic, which is the main problem facing the use of a NIDS in the cloud, in addition to dealing with large volumes of data without affecting the efficiency.

An IDS log cloud analysis system (ICAS) is a system developed in [178] to collect all the IDS log files and analyse them to provide user-friendly and useful attack reports for administrators to read and react accordingly. The main problem that this system solves is dealing with extremely large IDS logs, which are rapidly increasing in cloud environments. The Hadoop MapReduce algorithm was used to analyse IDS logs and generate reports.[13] The inputs of ICAS are IDS log files collected from

[10] The purpose of IDMEF "is to define data formats and exchange procedures for sharing information of interest to intrusion detection and response systems and to the management systems that may need to interact with them" [47].

[11] A cluster controller is a dedicated machine that controls a location (or an availability zone in Amazon terms); this location is a collection of node controllers, where each node controller manages a number of physical servers and each physical server hosts a number of VMs. The terms *cluster controller* and *node controller* are used by Eucalyptus, which is an open source private cloud software for building cloud services.

[12] "Session initiation protocol (SIP) is an application-layer control (signalling) protocol for creating, modifying, and terminating sessions with one or more participants. These sessions include Internet telephone calls, multimedia distribution, and multimedia conferences" [141].

[13] "Hadoop MapReduce is a software framework for easily writing applications that process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. A MapReduce job usually splits the input dataset into independent chunks, which are processed by the map tasks in a completely parallel manner.

different IDS sensors. These logs are then analysed using the MapReduce algorithm and the outputs are reports containing attacks graphs. The proposed system was tested using different NIDS sensors, including Snort, IDP8200, and NK7Admin, to collect logs.

In [14], researchers proposed a hypervisor-based IDS that was not designed for cloud computing. However, we believe that it has good potential to work in the cloud because it requires neither prior knowledge from inside VMs nor the installation of any code inside them, which satisfy the most challenging cloud IDS requirements. The detection system was deployed in the host virtualisation layer. Only information available to the hypervisor was collected and used for detection. However, modifications to the hypervisor were required to be able to extract the required data during runtime. To implement a prototype of the suggested IDS they used a VirtualBox hypervisor, which is open source so can be modified. The extracted data included the I/O accesses, page faults, translation look-aside buffer (TLB) flushes, and control register updates. The semantic gap was bridged using data mining algorithms; they tested two methods: the distance-based K-Nearest Neighbour and the density-based Local Outlier Factor algorithms. The collected data were transformed into features using statistical techniques. Then, the features were used by the data mining algorithms to build a normal behaviour profile for the monitored VM, which could be used later for detection purposes (by measuring the deviation from normal). Only normal data were used to train the classifier. The proposed IDS was tested using 300 different malwares, and it provided a 93% detection rate and 3% false alarm rate. Two approaches were used to categorise the features: storing sum of events and the relationship between pairs of events. They reduced the feature space by using feature subset selection. We believe that this hypervisor-based IDS is a very strong IDS candidate that has the potential to work efficiently in the cloud. This IDS is further discussed later in the chapter 4.

In [148], a multithreaded cloud-specific NIDS architecture was designed to deal with various challenges. The first involved the problem of trust in the cloud when a consumer VM is breached or has a data loss incident, and the provider is aware of the incident but chooses to keep the consumer unaware of the incident for different reasons. The second is the problem of the high volume of data in the cloud. The third is the problem of the low attack-resistance of HDSs. They suggested a distributed multi-threaded cloud IDS that is monitored by a third party. When the IDS detects an intrusion, it raises an alarm to the third party, which in turn analyses the alert, generates a report, notifies both the provider and consumer, and sends them recommendations to deal with the detected intrusion. The researchers in [148] stated that a NIDS is more suitable for deployment in the cloud than an HIDS because an HIDS suffers from low attack-resistance and cannot deal with a high volume of data. They deployed the NIDSs outside hosts, i.e. close to switches or routers, where the suggested NIDS was signature-based. They also assumed that the third party was able to analyse NIDS alerts and generate useful security information and recommendations from them. We believe that using a third party to deal with the trust problem in the cloud might be a good solution. However, a

The framework sorts the outputs of the maps, which are then input to the reduced tasks. Typically both the input and output of the job are stored in a file system. The framework takes care of scheduling tasks, monitoring them, and re-executes the failed tasks"[64].

centralised IDS service might not be able to cope with the scalability and flexibility of the cloud. It represents a single point of failure. Furthermore, we question how realistic it is to assume that the third party is able to generate efficient security information and recommendations just by analysing the NIDS alerts. The research also did not mention how to deal with encrypted network traffic. Lastly, we do not think that a NIDS is an alternative to an HIDS, but is complimentary, and both of needed in cloud environments.

Another solution was proposed by Jie *et al.*, in [73], to solve the problem of trust in the cloud between consumers and providers. The researchers tried to help consumers to be aware of the security status of their VMs. The solution they suggested is a user-oriented distributed HIDS and NIDS. The system consists of two components; the first one is a management server, called a 3D-IDS server, which is deployed in the user's machine outside the cloud (off-premises server). This server is responsible for receiving alerts from different IDS sensors and analysing them to generate security status reports. The second component consists of sensors that are installed by the consumer in each VM under its ownership. These sensors are HIDSs, NIDSs, and system logs.[14] In our opinion, these sensors have a thorough view of all the activities and traffic inside VMs. However, they suffer from a low attack-resistance; they can be neutralised by attackers if the VM is compromised. Furthermore, sending system logs and sensor alerts though the network (which is the *Internet* in the case of a public cloud) will have a negative effect on the cloud's overall performance by increasing the bandwidth consumption because of the probable large volume of data. Moreover, if the analysis of the collected data is conducted in the VM sensors themselves (which was suggested by the researchers), they will require more computation power, which will increase the cost because the cloud cost model is pay for what you used. Lastly, we believe that, from the user's point of view, this IDS architecture is a regular distributed HIDS, which is an old problem, not the new security challenge introduced by cloud computing.

Alharkan *et al.*, in [8], also suggested a user-oriented distributed IDS like in the previous research. The suggested IDS is an on demand service provided to consumers. It is an IDS-as-a-service (IDSaaS), with the pay-for-use cost model. It is a NIDS because it monitors the traffic between VMs. It has three components; the first is the IDSaaS manager, which performs all supervision tasks and manages all of the sensors. The second is the IDS core (sensors), which monitors the traffic in the private subnet. Many identical copies could be made from the sensor to cope with the high volume of data and avoid a single point of failure. The third component is the LoadBalancer, which is responsible for distributing traffic among the sensors. This paper was just a simple proposal for an IDSaaS. It is not clear how consumers will use the system and many details are missed.

Kholidy and Baiardi, in [95], developed a cloud intrusion detection system architecture called CIDS, which stands for cloud-based IDS. CIDS is both an HIDS and NIDS using signature and anomaly-based detection methods. It is a distributed detection system with no central coordinator to suit the scalability of the cloud and to be able to cope with large data volume. Each host should have two databases, an audit system, and an alert system. The databases are used by the CIDS, one

[14] They considered three different types of sensors, which is the reason for calling it three-dimensional IDS. However, we consider the collection of system logs to be part of an HIDS.

by the signature-based IDS and the other by anomaly-based IDS. The audit system monitors message exchanges between hosts, log system events, and collect data coming from within VMs. The alert system is used to communicate alerts to other hosts in the cloud in the case of incidents, either directly or through the node controller. It also provides an application called the report producer with the required data to generate a readable report for cloud administrators. The report producer is located in a dedicated machine called *the scheduler* machine, which is accessed by cloud administrators to read incidents reports. The scheduler machine collects alerts from VMs and NIDSs, which are deployed in physical or virtual switches inside the cloud virtual network. Each VM should have a sensor that is responsible for collecting logs from inside the VM, audit data, user actions, and commands, and acts as an HIDS. CIDS runs using different detection models. The first is when hosts have to exchange consumer audit data to obtain complete audit data for the current consumers in each of them.[15] Another detection model is when hosts exchange consumers' audit data with the use of a neural network method. The last model is when every host works independently and there is no need to exchange audit data. The proposed system is scalable and flexible with no single point of failure. Even the scheduler machine is not a single point of failure because as they said, "a cloud runs several copies of the scheduler node with a fault tolerance technique provided by the middleware to backup the processing data" [95]. It is not clear how the NIDS will deal with the problem of detection in encrypted traffic. Lastly, the level of intrusiveness is high because they install a piece of code inside each VM, which exports too much *sensitive* data that could lead to trust problems, legal problems, and performance problems.

Zou *et al*., in [191], designed a cloud monitoring framework and implemented it on Xen and Openebula.[16] The framework is used to monitor different VMs with different operating systems, with almost no intrusiveness (no need to install any code inside VMs), but they require prior knowledge about the monitored VM's operating system version. It also deals with the trust problem between providers and consumers: first, by separating the monitoring functionality from the VMM by deploying them in a specific VM (monitoring VM) to reduce the services provided by the VMM and thus reducing the attack surface. Second, the integrity of the monitoring VM is maintained by utilising the trusted computing technology. Furthermore, an independent trusted computing base (TCB) was established for each monitored VM. Lastly, the researchers suggested putting the monitoring VM under the control of a trusted third party.

The suggested framework has three main components. The first component is the management VM (MVM), which is used by cloud administrators to control and manage other VMs. The second component is the monitoring VM, which has a piece of code called the monitoring driver. The monitoring VM might have several monitoring drivers corresponding to different types of operating systems in the monitored VMs. With the arrival of a new VM, a correspondent monitoring driver will be loaded. The monitoring driver is used to process the low-level data

[15] It is not clear to me why they want to maintain all of the audit data for each consumer in each host. However, I think the reason is to allow the data to be used in the case of migration or in a case where one consumer has more than one VM; although they did not mention migration in the paper.

[16] OpenNebula is an open source cloud computing management system that is used to manage VMs in distributed environments[133].

collected from the corresponding monitored VM. It has the needed information to understand the low-level data and bridge the semantic gap (more about this part of Zou *et al.*'s research is discussed later in the chapter 4, section 4.4). The third component is an event sensor that is deployed in the VMM and is responsible for collecting low-level data coming from the monitored VMs. It collects data about VMs from registers and transfers the data to the correspondent monitoring driver. The collected data are only system calls coming from VMs. The event sensor also monitors privilege VMs (not just consumers VMs). To be able to monitor privilege VMs and intercept events, a modification of the VMM code was required.

One of the main goals of cloud services is reducing the need for human intervention to the minimum possible. Some research papers have relied heavily on users to supply some data for the IDS, such as in [130]. Onoue *et al*. proposed a system to monitor VMs from outside. Another VM dedicated for monitoring was used to monitor VMs. They also used a modified version of the hypervisor (Xen) in the host machine. The system installed in the monitoring VM intercepts system calls from certain processes that are specified by the VM user ahead of time. In the monitoring VM, there should be a security policy that specifies what is allowed for certain processes and what is not, and how to react if a violation occurs. This policy also should be provided ahead of time by the user. The user is also required to provide knowledge about the structure of its operating system kernel, such as the addresses of several routines in the kernel space corresponding to the system call entry and exit. This knowledge is generated automatically for the supported operating system (Linux only) using a tool during the process of installing the operating system. All the required information depends on the version of the used kernel. Most of the tests in this paper involved the performance. We are not sure how the system performs from the security side. We think that the main obstacles here are the requirements from the user. Deciding what processes need to be monitored and a security policy for each process are very complicated tasks. However, some of the techniques used here are useful for converting a research laboratory HIDS into a real-world system.

In [36], researchers developed a system called SYRINGE for monitoring virtual machines. The main goal is protecting the detection system from being neutralised or tampered with in the case of a compromised host by separating the detection system from the monitored host. The monitoring application in SYRINGE was placed in a separate VM called a security VM (SVM). SYRINGE is allowed to invoke functions from inside the monitored VMs (VMs' operating systems) using a technique called function-call injection, which allows the SVM to invoke functions in the monitored VMs by interrupting their execution and manipulating the contents of their CPU and memory. We consider the system's ability to invoke functions from inside the monitored VMs to be a high level of intrusiveness that is not acceptable in a cloud environment.

3.4 Cloud Attacks

Many cloud-specific attacks developed in academia have been published recently, such as in [167], [139], [53], [187], [19], [164], [183], [184], and [180]. As explained in section 3.1.8, public cloud computing is vulnerable to DoS attacks due to its

openness and shared resources. It is vulnerable to different types of DoS and DDoS attacks. Some of the published cloud attacks were implementation-based, whereas, others were architecture-based. Examples of implementation-based cloud attacks are the attacks introduced in [187] and [53]. In [187], researchers discovered a vulnerability in the Amazon EC2 version of the Xen hypervisor that can be exploited by malicious co-resident VMs, which allow them to monopolise about 98% of the CPU time and affect the performance of other consumers, thus violating the SLA.[17] This type of attack is called a *Theft-of-Service* attack. They tested and proved the severe damage from this attack, notified Amazon about the vulnerability, and Amazon patched it.

The other implementation-based attack is *VM escape*, reported in [53], which we discussed in section 3.1.3. Architecture-based DoS attacks (compared to implementation-based) represent a more significant threat to the development of the cloud. Therefore, they have more weight in our research.

As we discussed in section 3.1.8, any economic-loss attack can be considered to be a DoS attack. The first architecture-based DoS attack is the *economic denial of sustainability* attack (*EDoS*). The idea for this attack was first suggested by Hoff in his blog in 2008 [77]. The attack targets the cost model in the cloud(pay-for-use) and the *on-demand elasticity* feature. Attackers enlarge the bill of the targeted cloud consumer by requesting the provided service using fake or incomplete requests. Due to these fake requests, the victim will consume more resources and might also respond to the increased demand by expanding, and thus paying more money to the provider. This attack and its mitigation were discussed in [152], [167], and [177]; however, we believe that there has still been no deep analysis of the EDoS class of attacks in the literature.

The second cloud attack is the cross-VM attack discussed in [139]. This attack aims to extract sensitive information from the targeted innocent VM, such as cryptographic keys. It requires the attacker VM and targeted benign VM to live in the same cloud host (co-residency). Then, it uses a side channel to extract the required sensitive data or information.[18] To increase the possibility of co-residency, they suggested using techniques such as finding and analysing the cloud placement algorithms (the algorithm used by cloud management machines to distribute VMs among hosts, more about this in chapter 5, section 5.1.1). They also suggested collecting network features such as the structure of IP addresses to be able to map the infrastructure and help achieve co-residency. Furthermore, a brute force strategy can be used for co-residency. In a brute force attack, the attacker initiates a large number of VMs and terminates the non-co-resident ones. Many techniques can be used to check the success of co-residency. Some of these techniques are network-based such as network probing, time-to-lives (TTLs), small packet round-trip times, and numerically close internal IP addresses. Other techniques are based on using a covert channel to perform co-residency checks. They also proved that a malicious VM can monitor the workload of a co-resident VM through time-shared caches. The attack was tested in the lab and also in Amazon EC2.

Many defence strategies were suggested in the same paper to defend against

[17] Amazon uses a modified version of Xen that is concealed.

[18] In a side channel, an attacker obtains secret information by monitoring the activities and hardware status of the targeted system [189].

the cross-VM attack, such as keeping the internal structure of the cloud and the placement policy confidential. They also suggested blocking all possible side channels. Furthermore, they said that "the best solution is simply to expose the risk and placement decisions directly to users" [139]. We found that the first solution (keeping the placement policy confidential) is security through obscurity, which is a risky proposition. The second solution (block side channels) is almost impossible because it is hard to prove that all side channels are blocked; there are no guarantees. The third solution (expose the risk to users) is only transferring the problem. The more acceptable solutions from the security prospective are, for example, improving the detection mechanisms to cover this attack and strengthening the hypervisors' isolation techniques.

Bates *et al.*, in [19], developed a traffic analysis attack that works in cloud environments, in addition to any other virtualised environment. In this attack, the attacker successfully measures the workload and extracts more accurate traffic information about the victim VM using a malicious co-resident VM, without relying on side channels, as in the previous attack. The malicious co-resident VM injects a watermark signature into the network traffic of the victim co-resident VM. First, the malicious VM (which is a regular web server) uses co-resident watermarking to confirm co-residency. Then, the attacker from outside the cloud initiates a single TCP session with both the malicious and victim VMs. After that, the attacker measures the workload of the victim by observing the ratio between the throughput of the two TCP flows. Since the two VMs live in the same server and the two flows usually use the same network path, they are affected by the same conditions, i.e. network congestion. Therefore, the effect of these conditions can be ignored because it does not change the ratio. The inserted watermark can threaten the resource isolation between VMs. This attack technique was evaluated in the lab and in a production environment. A relatively wide covert channel (4 bits per second) was successfully established and also tested as a prove of co-residency using network-based techniques. They used a Kolmogorov-Smirnov distribution test to examine the inter-packet delays between VMs. In the experiment, they proved co-residency in less than 10 s and injected the watermark successfully within 2.5 s.

Resource freeing attacks (RFAs), were introduced in [164] by Varadarajan *et al.*. They tested different techniques to improve the performance of the malicious VM by forcing other innocent co-resident VMs to release resources which make them available to attackers to use (steal the service). For instance, attackers analyse the targeted VM to find a performance bottleneck application or service on it. They then saturate the bottleneck so the victim is forced to suspend other used resources, which makes them available to the malicious VM to use. They implemented the attack in the lab and in Amazon EC2. In the lab, they successfully increased the malicious VM performance by about 60%, and in Amazon EC2 it was increased by about 13%. The performance of the targeted VM was also heavily affected with a degradation in performance of about 80%. This attack also, as in previous attacks, required co-residency and a co-residency check. They used a brute force technique to attain co-residency. For the co-residency check, they used a network-based technique involving the packet round-trip times of network probes and a cache-based covert channel technique. The main contribution of the paper was finding free-up resource strategies. The researchers also suggested some mitigation strategies such

as using dedicated servers to prevent co-residency, in addition to recommendations to strengthen the isolation between VMs and improve the scheduling mechanism.

As can be seen, many cloud-specific attacks are based on exploiting vulnerabilities in the resource isolation between VMs. The isolation and resource distribution are the main job of the hypervisor. However, research has proven that the isolation and resource distribution in virtualised environments need to be improved [164].

Covert channels are widely used in cloud attacks (due to the multi-tenancy feature and isolation insufficiency). Some research papers such as [173], [112] and [176] have attempted to create a covert channel in virtualised environments. Wu *et al.*, in [173], designed a covert channel dedicated to cloud computing environments. It is based on a memory bus not on shared caches as most of other covert channels. The researchers argued that cache-based covert channels are not efficient in cloud environments for many reasons. The first reason is CPU cores do not share any caches. The second reason is the hypervisor completely flushes the L1 cache at context switches (when the resource user has changed). The third reason is the VMs use virtual processors, which are frequently migrated from a physical processor core to another by the hypervisor. The covert channel designed by Wu *et al.* was implemented and tested in the lab and in Amazon EC2. In the lab, it successfully transferred about 700 bps with an error rate of 0.09%, while in Amazon EC2, it transferred about 100 bps with an error rate of 0.75%. They also designed a protocol to use the suggested covert channel, and it was used in their experiments. More information about covert channels is provided in chapter 5.

For security assurance, some cloud providers offer a dedicated hosting service to their consumers for extra money, which eliminates the threat of inter-VM attacks and any covert channel leakage. Zhang *et al.*, in [183], proposed a system that verifies the exclusivity of using the host resources for consumers who request dedicated servers. It is necessary to verifying the exclusivity because, as discussed in section 3.1.5, there is a trust problem between providers and consumers. The system is called HomeAlone, and it uses an L2 memory cache-based side channel to check the exclusivity. The same suggested side channel was also used in [184] for attacking innocent co-resident VMs and extracting cryptographic keys from them.

3.5 Conclusion

As we can see from the literature, there is a need for an efficient cloud HIDS that is able to monitor VMs without requiring any prior knowledge from inside them or installing any code in them. This HIDS should also be able to cope with large volumes of data, should be flexible enough to suit VM migration, and have no single point of failure. It also has to have adequate generality to be able to deal with different types of VMs and should also be able to perform on the fly and without disrupting the service or causing an SLA violation. We also found that hypervisor-based IDS has great potential to work efficiently in the cloud. In the next chapter, we propose two new hypervisor-based monitoring and detection solutions that are designed to work in large-scale public IaaS cloud environments and satisfy their requirements.

From the literature on cloud attacks, we found that there is usually a generic theme in most of the available architecture-based DoS cloud attacks. They require

co-residency, co-residency checks, and the use of covert channels for communication or stealing data or resources. We used the same theme to develop our new cloud DoS attack, as discussed in chapter 5.

Chapter 4

Host-Based Virtual Machine Monitoring

The providers of public IaaS cloud services are required to protect their infrastructure, prevent VMs from attacking each other, and prevent malicious users from attacking the outside world using the cloud. Furthermore, providers need information about the security status of each consumer to maintain the security of the whole cloud and deal with the security ignorance of consumers by notifying them, isolating them, charging them extra money, or even terminating their contract if the situation requires it after checking the legal status (i.e. if a breach of contract has been detected).

To fulfil these security requirements, there is a need to monitor all of the hosted VMs to detect any malicious attempts and be aware of the security status of each consumer and the whole cloud. Intrusion detection systems (IDS) are usually used to continuously monitor and detect any suspicious traffic or events.

As discussed in section 3.1.10, a network-based IDS in the cloud suffers from numerous issues. First, it cannot monitor encrypted traffic, and almost all the traffic in the cloud is encrypted. Second, it cannot detect attacks between co-resident VMs (inter-VM attacks). Third, it also cannot inspect all the traffic in switched environments, and cannot cope with a high data volume, which is one of the main characteristics of the cloud.

Regular HIDS in the cloud also has the problem of coping with a high data volume, in addition to suffering from low attack-resistance (when the attacker successfully compromises the machine, he or she can disable the monitoring application). Furthermore, the main problem facing regular HIDS in the cloud is that to monitor VMs efficiently, the detection system needs to access them and collect data from within; these data are used to bridge the semantic gap and generate useful information about the security status of VMs. Signs of intrusion are extracted from the data collected from inside VMs.

From chapter 3, we understand that there are numerous special requirements for cloud detection systems. These can be summarised in seven points:

1. dealing with large volumes of data;
2. monitoring VMs without accessing them or installing code in them, with the minimum level of intrusiveness;
3. requiring no prior knowledge about VMs;
4. high attack-resistance so consumers or attackers cannot disable the monitoring system;
5. being able to work on the fly (fast training, monitoring, and detection);

6. being able to migrate with the corresponding VM in the case of migration, which means the IDS has to be small, autonomous, and mobile; and
7. being completely automated and not requiring any human intervention to suit large-scale public IaaS cloud environments, which are automated, flexible, and tense.

These requirements must be considered in a new cloud-specific intrusion detection system, in addition to the regular host-based intrusion system (HIDS) requirements, which are an acceptable overhead, a minimum false positive rate, and an acceptable detection rate. Furthermore, maintaining a balance between false positives and false negatives is essential.[1] In addition, there is the requirement of being able to detect novel attacks, especially in open environments such as the cloud.

In this chapter, we propose two new intrusion detection systems that satisfy these requirements to a high degree. The new detection systems were designed and tested in the lab. They are a type of hypervisor-based intrusion detection system. As proven in the literature and discussed in chapter 3, the best HIDS candidate to work in the cloud and monitor VMs is the hypervisor-based IDS. There are two types of hypervisor-based IDSs: anomaly-based IDS and signature-based IDS. A signature-based IDS detects malicious events by comparing them to known and pre-determined attack patterns. It requires prior knowledge and human intervention, and cannot detect novel attacks. Therefore, it is less appropriate for a cloud environment in comparison with an anomaly-based IDS.

In a hypervisor anomaly-based IDS, the monitoring system is used to build a character or a normal behaviour profile for each of the hosted VMs and detect any deviation from normal as an anomaly. To build the profile and detect anomalies, the IDS should go through several stages:

1. selection of learning methods and features to build the classifier,
2. collection of data to be used for training and testing,
3. training of the classifier using the collected data to create the normal behaviour profile, and
4. testing of the classifier to measure efficiency and effectiveness.

To select the learning method, we need to first decide the source of the data to be used to build the behaviour profile of the monitored VM. We chose to monitor all system calls invoked by the VM and used them to build the normal behaviour profile that represents the VM; the reasons for using the the system calls as the main source of data and the literature on their use in an HIDS are discussed in section 4.1.

After selecting the source of data (system calls), we then built a small cloud environment with the minimum acceptable setup for production environments, as suggested by the Eucalyptus system administrators guide [56]. The details of the setup and experimental tools and how we generate normal and abnormal traffic, in addition to the methods we used to analyse the system calls to get useful security information and bridge the semantic gap, are discussed in section 4.2.

[1] False positives occur when normal events are falsely classified as anomalous; false negatives occur when malicious events are falsely classified as normal. The balance between false positives and false negatives is the same as the balance between under-generalisation and over-generalisation.

We collected data (normal and abnormal) that we used to build the two new hypervisor-based IDSs that we proposed. The first one was built using the BoSC, which was first suggested in [93]. The first reason for choosing BoSC is its simplicity (it is almost the simplest method we found in the literature). Because this study is one of the first in this area, it is more reasonable to start with a simple method and improve or change the method when needed. The second reason for choosing BoSC is that it ignores the order of the system calls and only consider their frequencies; maintaining the order of events in a virtualised environment is a hard task, and an IaaS cloud is a virtualised environment.

The second system is built using the hidden Markov model (HMM) representation method. This detection system was designed to monitor ephemeral VMs: VMs that run for a short period of time to accomplish a specific task and then terminate (which is a popular usage model in the IaaS cloud). Collecting data from ephemeral VMs, using the data to build representative normal behaviour profiles, and then using these profiles to detect any change in behaviour are challenging tasks given that the VM will only be *on* for a short period of time. Time is critical in the process of training the IDS to monitor these VMs and detect any anomaly. We decided to use a representation method that provides a high detection rate with the minimum time even if it consumes more resources, if not time. We found from the literature that HMM satisfies our need [170] and [128]. HMM is known to provide a very powerful model to capture the structure of sequential data. Therefore, we used it to build the second detection system.

The new detection systems could not be tested in a real productive environment (for legal and security reasons); they had to be tested either in a virtualised environment or in a real environment in the lab. We decided to test the systems in the best available choice, which was a real environment in the lab. We built an IaaS cloud in the lab with the minimum acceptable setup suitable for productive environments, as suggested by the Eucalyptus system administrators guide [56]. We host a number of VMs in our cloud with activities on them; the hosted VMs ran an ERP (enterprise resource planning) application and performed regular ERP tasks. After that, data were collected (normal non-malicious data) from each of the VMs and used to build the normal behaviour that represented each of the VMs (training the classifiers).

To generate abnormal data, we ran abnormal activities in the VMs; abnormal data were then used for testing the detection systems. The idea was to change the VMs' behaviours and test whether the change in behaviour could be detected by the classifiers. Any change in behaviour should have been considered to be a sign of abnormality in VMs because we assumed that the VMs had consistent behaviour. To generate abnormal data we designed an attack scenario based on stressing the CPU to trigger a migration order for false reasons to misuse the migration and overcommitment features of the IaaS cloud. That was the initial test of the attack that was later further developed and demonstrated in chapter 5.

We tested the new detection systems using normal and abnormal data. The detection rate, accuracy rate, and false positive rate were used to measure the efficacy of the proposed systems, which are widely accepted criteria for measuring the efficiency of HIDSs [93]. We also calculated the required time, storage, and size of the representative normal behaviour profile. To determine whether the anomaly

signals were strong enough, we used hypothesis testing.

The details of the new monitoring systems and their performances in the lab are discussed in section 4.3.

Section 4.4 is dedicated to discussing other monitoring systems that are very close to the proposed systems. There are three very strong candidates: VMM-IDS by Azmandian *et al*. in [14] and two systems proposed by Zou *et al*. in [174] and [191]. In this section, we discuss the details of these systems, evaluate them, and demonstrate the difference between them and our systems.

The last section before the conclusion is section 4.5, which covers possible attacks against the proposed systems; then the conclusion and future work are discussed in section 4.6.

The initiation of the idea:

Forrest and her team [63] developed an anomaly-based intrusion detection system that relies on monitoring the system calls invoked by root processes based on the assumption that root processes have consistent behaviour and any change in behaviour might be an indication of an abnormality. We noticed the similarity between root processes and the VMs hosted in the IaaS cloud: both are expected to have consistent behaviour and any change in behaviour could be an indication of an abnormality.

Why are the VMs hosted in a cloud expected to have consistent behaviour?

First, the VMs hosted in an IaaS cloud are limited in the services and applications used. This assumption is relative to regular servers that may aggregate multiple services on a single instance. Regular best practices for IaaS cloud services would argue against such mixed workloads being deployed in a single VM for security and performance reasons.

- For security reasons, clients who own VMs hosted in the cloud are advised not to mix services in one VM. As stated by Krutz *et al*. [102], "while contemporary servers and virtual machines are adept at multi-tasking many functions, it's a lot easier to maintain secure control if the virtual machine is configured with process separation. It greatly complicates the hacker's ability to compromise multiple components if the VM is implemented with one primary function per virtual server or device."
- For performance reasons, as stated in [11], "companies with large batch-oriented tasks can get results as quickly as their programs can scale, since using 1000 servers for one hour costs no more than using one server for 1000 hours. This elasticity of resources, without paying a premium for large scale, is unprecedented in the history of IT." The cost model in the cloud, usage-based pricing, supports this idea. Instead of creating one large VM with many functions, the client can create a number of small VMs, each with one primary function, and still pay the same amount of money.

Second, in general, servers usually tend to perform the same series of tasks repeatedly, which provides a sound data set for training.

If VMs have consistent behaviour, an anomaly-based HIDS is a good candidate to build a normal behaviour profile that represents each VM in the cloud and detects any change in behaviour as a sign of abnormality.

4.1 Learning Methods and Features Selection - System Call HIDS

Different methods can be used to design the anomaly-based IDS (to build the classifier and detect anomalies). The choice of method depends on the environment requirements and the available data for building the normal behaviour profile and indicating attacks. For instance, slow methods cannot be used in environments that require detection on the fly (fast training and fast detection).

4.1.1 Available Data for Training

Each VM hosted in an IaaS cloud has two communication channels. The first one is between the VM and the host, which is occupied with system calls and signals. The second channel is between the VM and the outside world through the network using packets (see figure 4.1).

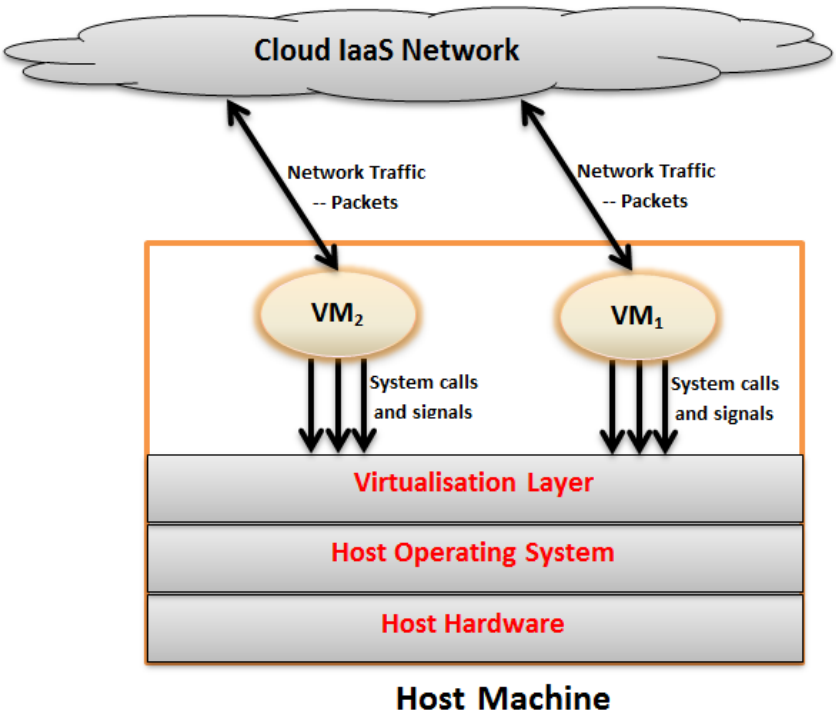

Figure 4.1: VMs communicate with host using system calls and signals and with outside world using network packets.

Network packets in the first channel can be monitored using a network-based intrusion detection system (NIDS) (which is not covered in this research). The second channel between VMs and the host, which is occupied with system calls and signals, can be monitored using a system call-based HIDS. System call-based HIDSs are reported in the literature to be effective, and they are discussed next.

4.1.2 Literature on System Call HIDS

Monitoring system calls and using them to build an HIDS is not new. Usually shell commands, audit events, and/or system calls are used to build an anomaly-based HIDS [92]. We monitor system calls for detection because they are almost the only available data we can collect from VMs without accessing them or installing any code on them. Furthermore, previous research proved that detecting anomalies by monitoring system calls provided high efficiency and effectiveness [63], [93], and [128]. Any activity inside VMs requires invoking system calls, which are sent to the host and can be analysed to detect anomalies. The main problem here is bridging the semantic gap; system calls are low-level data and extracting useful information from them is a hard and tricky task.

Another problem in system call-based IDS is its inability to detect attacks not reflected by system calls and attacks that do not change the behaviour of the monitored VM. This problem becomes an issue if the IDS only monitors one application or process in the VM, and the attack is performed using another application to exploit errors of omission in the attacked application, such as race conditions, opening files without appropriate safeguards and checks, and leaving temporary files with critical information. These errors in the monitored application are usually exploited by another application, and thus they might not cause a change in the behaviour of the attacked application. In our case, we monitor the whole VM as a black box, including the *malicious* and *victim* applications. These will most probably be reflected in system calls, unless it is a 100% network-related attack, which should be detected using NIDS. As we stated in section 3.1.10, *an HIDS is not an alternative to the NIDS*; they are complementary systems. There are types of attacks designed specifically to avoid detection by a system calls-based IDS. These are discussed later in the chapter, in section 4.5.

Jyothsna *et al*. in [92], provided a thorough survey of anomaly detection methods. They categorised them into five types: statistical-based methods, computer immunology, user intention identification, cognition-based, and machine learning-based methods. Statistical-based methods include the Markov process, operational, multivariate, statistical moments, time series, and univariate methods. Finite state machines, description scripts, and expert systems methods can be subcategorised under cognition-based methods. Lastly, the machine learning category comprises Bayesian networks, generic algorithms, neural networks, fuzzy logic, and outlier detection.

System calls are sequences of observations that are made over a period of time. Thus, they can be considered to be a time series model and can be analysed using statistical, machine learning, and data-mining methods. Each system call might have arguments and/or return values. Some of the used methods to build the classifier are simple; they do not contain any probability calculations and ignore arguments and return values, which reduce the complexity of the method and allow it to operate faster, i.e. sliding windows (stide) [63], RIPPER [105] and bag of system calls [93]. Other methods consider arguments for building the classifier and detecting anomalies [157] and [128]. The main reason for ignoring arguments and not using probability calculations is to reduce the cost (reduce the overhead on the system) [62].

Mutz *et al*., in [128], compared methods that considered arguments with other

simple methods and proved that the previous ones are efficient and more accurate (see table 4.1).

Table 4.1: Comparison between different methods of learning extracted from [128].

	FN	FP
Stide system from [63]	3	208
Bag of system calls from [93]	3	116
K-Nearest neighbours	2	179
Cluster-based estimation	12	158
Mutz *et al.*'s system	0	39

As we can see from the table, Mutz *et al.*'s system, which is an argument-based system, [128], gave the best results; it gave 0 false negatives and 39 false positives. The second best system is the bag of system calls (BoSC), which gave 3 false negatives and 116 false positives.

Mutz *et al.*'s system classified system calls using Bayesian networks and scoring instead of a simple threshold-based scheme. If the root node of the Bayesian network registers a 50% or more probability of abnormality, then the system call is classified as malicious. Regarding the overhead, the system did not affect the performance of the host during regular usage. However, when the rate of invoking system calls was increased (heavy use of the host), the performance degraded; the CPU load was 40% when the detection system was disabled and 58% when the system was enabled. To improve the performance, only a chosen subset of all invoked system calls was monitored. Then, a profile was created for each system call to register its normal arguments, in addition to a set of procedures (a function) to decide whether or not an argument was an anomaly, using a probability calculation; arguments with low probabilities were classified as anomalies. The set of procedures and evaluated features of the arguments were designed based on the designer's knowledge of known attacks. Known attacks were analysed to decide which arguments and features to consider. Examples of the features considered were the argument string length, string character distribution (normal frequency of characters), structural inference (i.e. grammatical analysis of the argument) and token finder (define a set of possible alternatives for the argument).

The proposed method increased the CPU load by 45% in the case of a heavy load host. It also required prior knowledge and an analysis of known attacks to select the detection features. As discussed in section 3.1.3, the cloud is a heavy-load environment; in addition, working on the fly is a main requirement. Furthermore, the detection system in the cloud should be fully automated, involving no human intervention, and no prior knowledge.

Simple method:

The time-delay embedding (tide) by Forrest *et al.* [63] is the simplest of the methods for system call anomaly detection. A sliding window technique was used to register system calls. Researchers tested window size of 5, 6, and 11. To explain a sliding window, see figure 4.2; if the window size is 6, a sequence of 10 system calls will generate 5 sequences.

Forrest *et al.*, in [63], monitored only the root processes of two applications (sendmail and lpr) and created a normal behaviour profile for each of them using

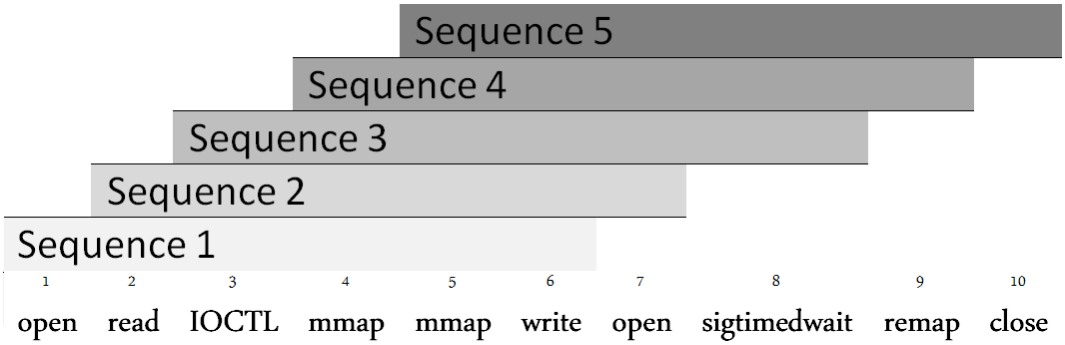

Figure 4.2: Sliding window of size 6 generates 5 sequences of system calls.

sequences of system call. They ran malicious activities on the monitored system, which generated traces for suspicious system calls; these sequences were collected and registered using a sliding window. Then, the registered sequences were compared with a database of normal system calls, and the number of mismatches was counted. The reason for monitoring only the root processes is that these show *relatively consistent behaviour* [63]; so it is possible to create a representative profile for them. In addition, the root processes have full access to the host, which make them a serious threat to it in case of exploitation.

Forrest *et al.*'s system was tested in an experiment. Enough data were generated to train the classifier, with additional data (malicious and innocent) for testing it. The experiment showed that by monitoring system calls *only*, malicious activities can be detected reliably.

Forrest *et al.*'s research in [63] is the first research to suggest using system calls to build normal behaviour profiles for processes to use in comparisons for anomaly detection. They based their research on two assumptions. The first assumption is that programs have consistent behaviours, which generate a pattern of consistent sequences of system calls. Consistency is essential for building a steady profile that represents the monitored process. The second assumption is that during an attack or errors, unusual sequences of system calls will be generated. System calls arguments were ignored for performance reasons.

The same researchers extended their research in [78] by fixing the window size to 10 and using *Hamming distance* as the matching rule to generate anomaly signals; the method was called sequence time-delay embedding (stide). To test the new system, data were collected from real environments located in the *Artificial Intelligence laboratory* of the *Massachusetts Institute of Technology* and in the *Computer Science Department* of *the University of New Mexico*. Multiple sources of data were used to show the effect of changing the environment on system call sequences. The data were first analysed for false positives, and then were used to create the normal behaviour profile.

After that, malicious traffic was generated by running malicious scripts, which were sunsendmailcp, syslogd, a decode alias intrusion, sm565a, sm5x, and forwarding loops error on the sendmail application. The proposed system successfully detected intrusions and errors and generated strong anomaly signals. Researchers also found that changing the window size had almost no effect on the

detection rate. Regarding the effect of changing the environment on system call sequences, they found that data from two different environments had created the same database pattern, and only the database size varied.

The experiments of Forrest *et al*. in [63] and [78] provide the foundation of most system call-based anomaly detection systems. Their promising results promoted more research in the area. The data they collected (sequences of system calls) were used later by many other researchers to test different system call-based detection methods.

Another study similar to that of Forrest *et al*., with modifications, was conducted by Kosoresow *et al*. in [100]. Kosoresow *et al*. used a sliding window to build the normal behaviour database and used deterministic finite automaton (DFA) with its equivalent language to describe the patterns of system calls. The DFA was constructed using macros with sequences of system calls with different sizes; so the length of the system call sequences was not fixed, and the macros were selected manually.

The detection system based on a machine learning algorithm was introduced in [106]. Data collected by Forrest *et al*.'s were used to test the new detection system, which was based on a machine learning classifier. Normal and abnormal data were used to train the classifier, which was different than most anomaly detection systems, where only normal data are used for training, and abnormal data are only used for testing. The classifier was based on RIPPER, which is a classification rule learning algorithm introduced by Cohen in [43]. The output of RIPPER is a set of *if-then* rules for normal traces; an example is, *if the third system call is 'lstat' and the fourth is 'write', then the seventh should be 'stat'*. A list of normal patterns was built using sliding windows with sizes of 7 and 11. Then, intrusion traces were scanned, and all the traces not found in the normal list were recorded as abnormal. The detection system was tested using new normal and abnormal traces, which were not used before in the training phase. A sequence of system calls was labelled as malicious if the majority of these were classified as abnormal. Furthermore, a machine learning algorithm was used to generate a rule that predicted system calls to define the normal correlation among system calls. For example, the algorithm predicts the middle system call in a normal sequence of size 11. If the system detects a system call that is different than the predicted one, it raises an alarm for an anomaly. The confidence value was used to express the strengths of intrusion signals. Lee *et al*., in [106], found that malicious sequences of system calls generated larger percentages of abnormal system calls than the normal sequences.

Some system calls-based anomaly detection systems have response techniques, [21] and [150]. In [21], only system calls with a high threat (privileged system calls) were monitored. Researchers targeted buffer overflow-based attacks as an example. However, the concept can be generalized. Their main contribution was the detailed analysis of Linux system calls. System calls were categorised into 9 categories (see table 4.2), and 4 levels of threats were created to classify system calls (see table 4.3).

Then, system calls were grouped according to their threat level; examples of the groups are shown in table 4.4.

Only system calls with a high threat level were monitored. Analysing and categorising system calls may help reduce the overhead on the system performance

Table 4.2: System calls were categorised into 9 categories by Bernaschi *et al*. in [21].

Group	Functionality	Group	Functionality
I	File system, devices	V	Communication
II	Process management	VI	Time and timers
III	Module management	VII	System info
IV	Memory management	VIII	Reserved
		IX	Unimplemented

Table 4.3: Four levels of threats were used to classify system calls by Bernaschi *et al*. in [21].

Threat level	Description
1	Allows to get full control of the system
2	Used for a denial of service attack
3	Used for subverting the invoking process
4	It is harmless

Table 4.4: System calls grouped according to their threat level by Bernaschi *et al*. in [21].

Threat	System call	Group	System call	Group	System call	Group
1	open	I	link	I	unlink	I
1	chmod	I	lchown	I	rename	I
2	creat	I	mknod	I	umount	I
2	brk	II	kill	II	ioperm	II

by monitoring only a small subset of system calls, rather than monitoring every invoked system call. Tables (4.2, 4.3, and 4.4) were made by Bernaschi *et al*. in [21].

The same idea of categorising system calls depending on the threat level (and monitoring only high threat ones) was tested in our research using the novel monitoring system we proposed, as detailed later in section 4.3.2.

In [150], researchers developed a system called *process Homeostasis* (*pH*). The pH system monitors the system calls invoked by every process in the system, and when an anomaly is detected, it responds by delaying or aborting the suspected system calls automatically. pH was tested on a Linux machine by adding an extension to the kernel to give the ability of detecting and responding. Somayaji *et al*., in [150], concluded that with the response mechanisms *on*, the system performance was heavily affected.

Mutz *et al*., in [128], conducted a comparison between different types of system call anomaly detection systems, the best performance was for the hidden Markov model (HMM)-based IDS, and the second best performance was for the bag of system calls from [93]. The bag of system calls IDS (BoSC) provided a better detection rate than other simple methods in addition to being far less complex in comparison with the HMM-based IDS. Kang *et al*., in [93], aimed to provide a detection system that has better performance and accuracy than *tide* from [63]. The bag of system calls (BoSC) method was derived from *bags of words modelling*, which is a popular

representation method in speech recognition. In BoSC, the order of system calls is ignored, and only their frequencies are considered. Researchers in [93] proved that the frequencies of system calls were enough to build a normal behaviour profile for processes and detect anomalies reliably. The system was tested using data from *MIT Lincoln Lab* from [109]. For clustering the distribution of normal and abnormal system calls, they used two-means clustering and a one-class naive Bayes algorithm. The proposed system provided promising results, a high detection rate and low false positives, see table 4.5 (this table is from [93]).

Table 4.5: Testing results for bag of system calls system from [93].

Program	Accuracy	Detection Rate	False Positive
UNM live lpr	99.28	100.00	1.29
UNM synthetic sendmail	80.3235	40.00	16.76

The main advantages of the system are as follows:

- only small size databases were needed (because only frequencies were stored without the order),
- fast learning,
- less memory was needed, and
- fast detection, which provided the ability to work on the fly.

Kang *et al.*, in [93], stated that by maintaining the order of system calls, the size of the database will increase exponentially; therefore, they ignored the order of system calls. We used a similar strategy in one of the detection methods we proposed. The details are presented later in the chapter, section 4.3.1.

Tandon *et al.*, in [157], also conducted a comparison between system call anomaly detection systems. The first one was a sequence-based IDS that ignored arguments, whereas the other one considered arguments, return values, and error status. A rule learning algorithm called LERAD was used to specify the monitored attributes of arguments. By considering arguments, the detection rate was better, but with a very high cost in time. One of the reasons for the increase in the time overhead was the complexity of the attribute learning algorithm LERAD [157]; it gave each rule a probabilistic score, which was used to calculate the degree of abnormality for system calls. The other reason for the increase in the time overhead was the enriched features.

Another comparison between different methods was conducted by Warrender *et al*. in [170]. Methods such as *stide*, *t-stide* (which is stide with a frequency threshold), *RIPPER*, and *HMM* were compared. In an experiment, HMM gave the highest accuracy, with the highest computational demand. Furthermore, researchers found that a sequence size of 6 provided the best results for stide and RIPPER. Warrender *et al*. stated that sequences of system calls are "regular enough for even simple modelling methods to work well" and "that weaker methods than HMMs are likely sufficient" [170].

In the next section, we present alternative methods to perform cloud security monitoring based on system calls, as well as a hypervisor-based HIDS to monitor VMs as black boxes. These methods also satisfy the special requirements of cloud-based monitoring systems.

4.2 Alternative Detection Methods

In this chapter, we propose two novel detection systems that we designed and tested in the lab. These monitoring systems are able to detect anomalies in VMs hosted in a public IaaS environment without additional instrumentation or any prior knowledge about VMs. The system can also work on the fly and deal with a large amount of data. We believe that they fulfil the special requirements for a cloud HIDS that were discussed in the state of the art chapter, chapter 3.

We monitored the channel between VMs and the host, which is occupied with system calls. All the system calls passing this channel were collected and used to create a normal behaviour profile representing the normal behaviour of the monitored VM; then any deviation from the normal behaviour profile was detected as an anomaly.

To create the normal profiles and detect anomalies using system calls only, we tested two methods. The first method is the bag of system calls (BoSC) [93]. The literature shows that it provides high accuracy and low overhead. Another reason for choosing BoSC is that it ignores the order of system calls and only considers their frequencies, which requires smaller databases and allows it to be migrated with the corresponding VM in the case of migration. Furthermore, cloud computing is a distributed environment and maintaining the event time within such an environment is a complicated and hard to achieve task. Therefore, relying on the order of the system calls for monitoring might not be reliable and is not recommended. The chosen method should also be simple and have low complexity to be able to work on the fly. In this regard, BoSC is a simple method that is not based on any machine learning algorithms or complicated probability calculations; it only uses frequencies.

The second method is the hidden Markov model (HMM). We used HMM to build a classifier that works with *ephemeral VMs*, which are VMs that are created and live for short periods of time (i.e. an hour or two) to perform a specific task and then terminate. Ephemeral VMs are usually used for testing purposes or analysing large amounts of data within a short time. These short-lived VMs have different detection requirements, where time is the most critical resource. For this reason and for this scenario, we used HMM, which is fast to build and has a high accuracy rate, but a higher computational demand.

The phases needed to build the detection system are as follows:

- *collecting* the data,
- *training* the classifier using the collected data and the chosen method (BoSC or HMM). The output of this step is a profile (database) representing the normal behaviour profile of the monitored VM, then
- *testing* the detection system by generating malicious traffic and measuring the classifier's ability to detect it.

We built an IaaS cloud environment in the *Penetration Test Laboratory* of the *Information Security Group*, *Royal Holloway University of London*, to collect data and test the designed detection using BoSC and HMM. Our setup in the lab has the minimum requirements for the real production of a cloud environment. The description

of the experiment, modification of the tested methods to suit our environment, and different phases to build the classifier are discussed next.

4.2.1 Experimental Design

In this section, we describe the setup of the laboratory-based IaaS configuration that was used to collect the data set. We built an environment similar to the Amazon elastic compute cloud (Amazon EC2) service, which is an IaaS cloud service and one of Amazon's web services (AWS). The tools used to build the experiment were as follows:

Eucalyptus: This is a cloud management application used to build and manage the IaaS cloud. It is an open source software under GPL. Its main purpose is to create and manage the cloud computing IaaS components [55]. The software was originally developed in 2007 as a university research project in the *Computer Science Department*, *University of California, Santa Barbara*. The attempt was to build a cloud environment similar to AWS to help researchers explore, develop and examine the IaaS cloud; two years later, a commercial version of Eucalyptus was established. We chose Eucalyptus as the cloud management application because it is similar to the cloud management application used by Amazon EC2 (which is currently the most popular IaaS provider on the market), and it is compatible with Amazon's AWS APIs (Application Programming Interface) which allows the use of any code designed for Amazon AWS without modifications [55]. It is also free, open source, and has detailed documentation for installation. Nowadays, some of the most popular cloud providers (i.e. Ubuntu and Dell) are using Eucalyptus as the main cloud management application [55].

Ubuntu: This is a Linux operating system, which was used in our cloud as the main operating system. Ubuntu is open source, flexible, and well documented, which were the main reasons to support our usage of this tool. The used kernel version was 2.6.32.

Kernel-based Virtual Machine (KVM): We used KVM as the hypervisor (virtualisation technology), which allowed multi-tenancy in our cloud. The choice of the hypervisor is critical and might affect the collected samples and the reliability of the results obtained from the analysis. In today's market many of the main players in cloud computing are using KVM as their main virtualisation technology, such as Ubuntu, Red Hat, and Dell UEC SE Cloud Solution. KVM is an open source full virtualisation solution that requires the support of a virtualisation extension in the host hardware. It can run over an unmodified operating system. KVM was included in the Linux mainstream from kernel version 2.6.20 and upwards as a loadable module *kvm.ko* [158]. One of the main features of KVM related to the process of building a detection system is that it deals with each VM as a single process. For instance, the host in figure 4.3 hosted three VMs, each of which was a single process with a unique process ID (PID). The first VM had the PID 1358, the second VM had the PID 1532, and the third had PID 1334; each of these processes had many children (threads) working under it. This feature was essential because the designed detection system monitored each VM as a single process.

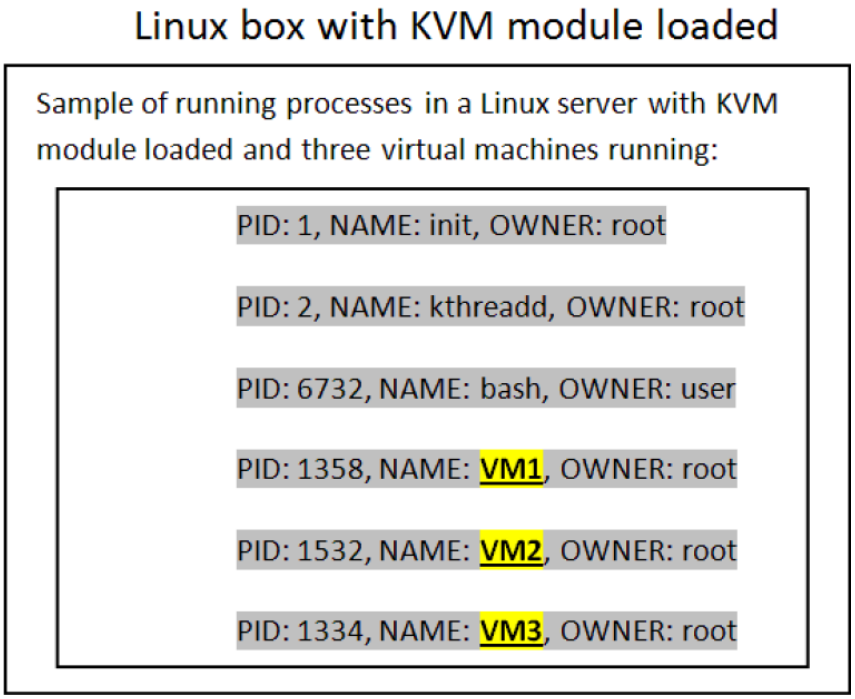

Figure 4.3: Each VM is represented by a single process in host machine.

4.2.2 The Setup

The minimum setup for an Eucalyptus-based cloud environment can be done by using a single machine for all the IaaS components, which are the cluster controller (CC), cloud controller (CLC), Walrus[2], storage controller (SC), and node controller (NC). However this deployment is not recommended by the Ubuntu Enterprise Cloud documentations for production use, and it has a number of registered bugs [51]. Furthermore, we found from users experience that it is not reliable. A more advanced setup was recommended. We used a setup called *single cluster setup*, which contained two machines, one for the CC, CLC, Walrus, and SC, and the other for the NC. The machine that had the CC, CLC, Walrus, and SC was called the *front end (FE)*. This setup is called the *single cluster setup* because it includes only one cluster, which is the minimum. However, a real live production cloud usually has more than one cluster, and each cluster has more than one node. The single cluster setup, which we used, is enough for a small cloud production environment; it satisfies all the requirements to run a stable IaaS cloud service.

Single cluster setup:

A single cluster setup can be deployed in different ways. Two examples of this setup are suggested by the Eucalyptus system administrators guide and can be found in [56]. The first example is shown in figure 4.4 (this figure is from [56]) and it shows the simplest configuration for a single cluster setup, where all the machines are connected directly to the public network.

[2]Walrus is the cloud storage equivalent to Amazon S3 (Simple Storage Service). It offers persistent storage to IaaS cloud users, i.e. it stores VM images and users' account information; it is also responsible for managing the consistency of the stored data [99].

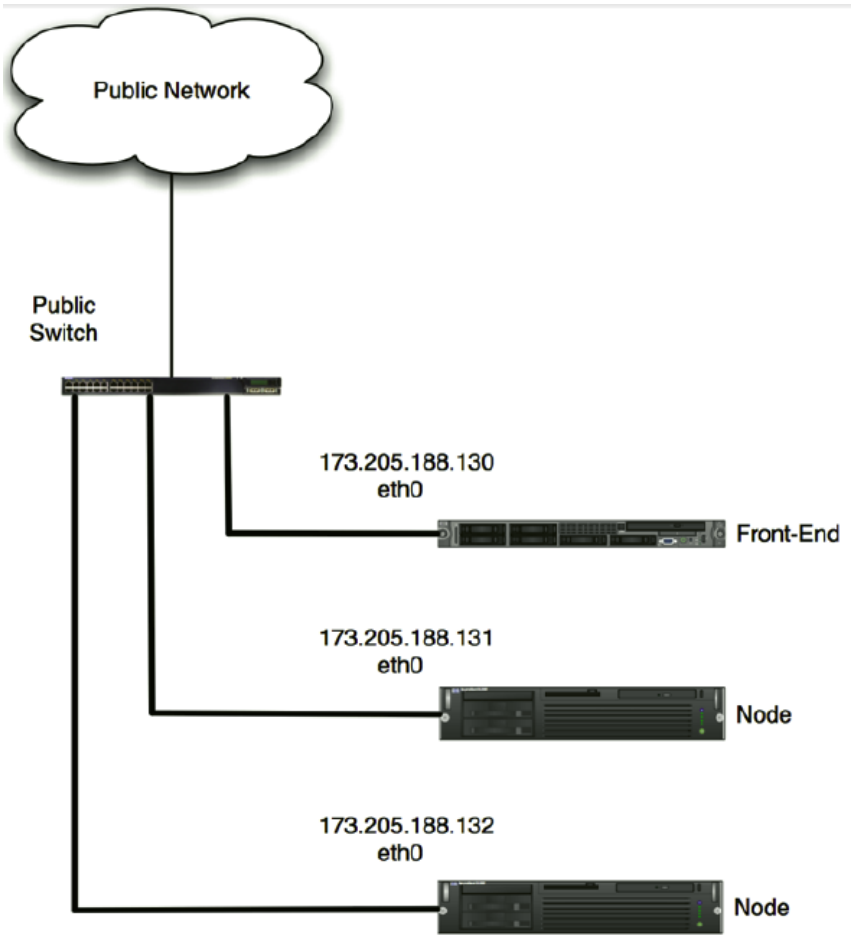

Figure 4.4: Single cluster setup where all machines are connected directly to public network [56].

The second example is shown in figure 4.5 (this figure is from [56]); in this setup, node controllers are isolated in a private subnet, and the FE server is the gateway. This is the deployment we used in our research; see figure 4.6 for our setup.

Details of machines:

Our setup is shown in figure 4.6; we used three main machines (Server1, Server2, and Client1), a router, and a switch.

Server1 is the FE server, which contained the CC, CLC, Walrus, and SC. The used operating system in Server1 is a Linux kernel 2.6.32-28-server. It has four Intel processors (1.86 GHz), each with vmx capability (virtualisation extension), 64 bit width, and 1 GB of memory. It also has two Ethernet ports: eth0, which is connected to the public network through a router, with an IP address of 192.168.10.3, and eth1, which is connected to the private switch and has the IP address 192.169.20.1.

Server2 has the NC, which is connected to a private switch and uses Server1 as a gateway. The used operating system in Server2 is a Linux kernel 2.6.3 -28- server. It has four Intel processors (1.86 GHz), each with vmx capability, 64 bit width, and 1 GB of memory. It also has two Ethernet ports: eth0 and eth1. However, only eth0 is used. The IP address for Server2 is 192.168.20.2. For this server, having more

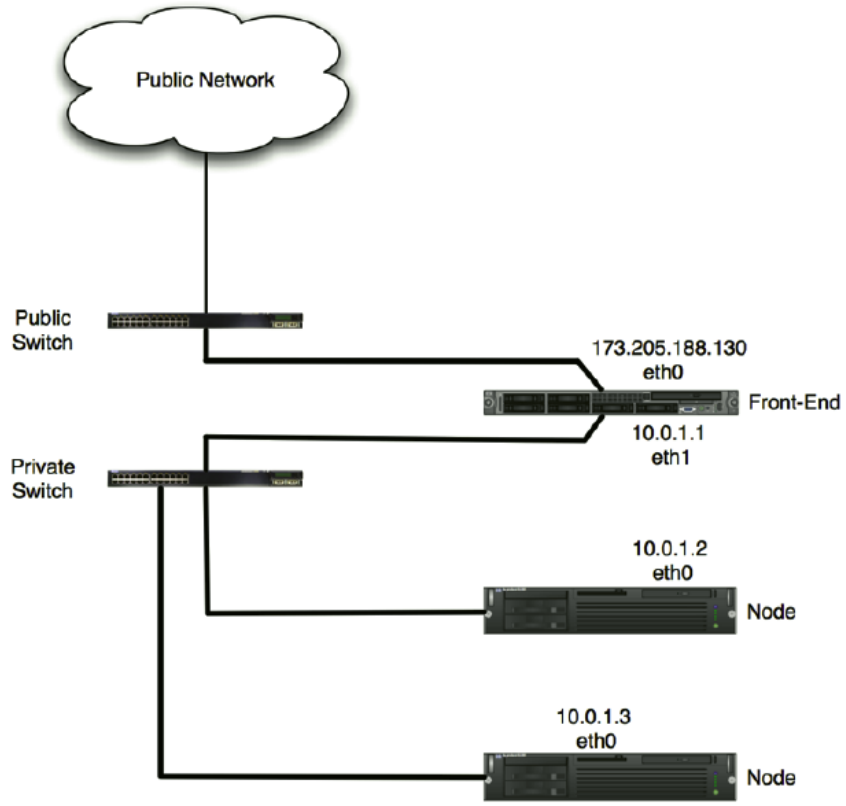

Figure 4.5: Single cluster setup where controllers are isolated in private subnet and FE is used as gateway [56].

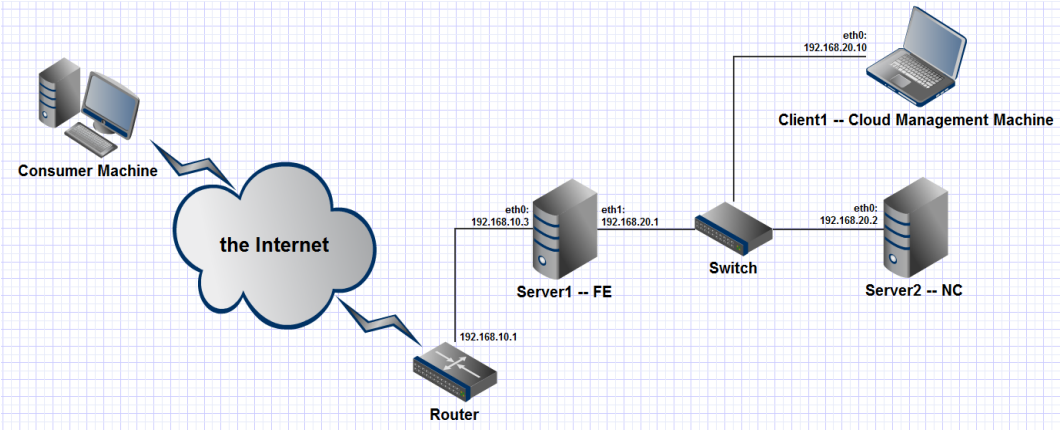

Figure 4.6: The experiment setup.

than one processor may affect the sequences of system calls. However, forcing the VM process to use only one processor is unrealistic because in a real-world cloud, servers have large numbers of processors to be able to host several VMs at the same time. VMs will move from one processor to another, which might affect the order of the collected system calls; this increases the importance of using a representation method that ignores the order of the system calls, as BoSC does.

Client1 is the cloud management machine. It is used to create and ping VMs, and uses secure shell (SSH) to access them.[3] It is also used to create and store access keys and for other cloud management tasks. This machine is an HP laptop with a Linux kernel 2.6.38-11-server. However, the specifications of this machine are irrelevant; any machine can play this role. The only requirement is the installation of the Euca2ools software, which is a command-line tool to interact with web services and send an API compatible with the Amazon EC2 service [50].

Cloud IaaS components:

An Eucalyptus-based cloud IaaS has five major components: the cluster controller (CC), cloud controller (CLC), Walrus, storage controller (SC), and node controller (NC), [56].

The cluster controller (CC) performs the following tasks:

- connects the NC with the CLC,
- allocates new VMs to hosts (NCs) based on reports received from hosts and other environmental parameters,
- answers CLC queries about load,
- gathers information about virtual machines and schedules their execution,
- manages virtual machines networks, and
- participates in the enforcement of the service level agreements (SLAs) as directed by the CLC.

The cloud controller (CLC) performs the following tasks:

- resources management (in which cluster a new VM should be allocated),
- communicates with the CC,
- provides a GUI to the management machine, and
- manages the CC, SC, Walrus, and network via the Amazon EC2 API and web-based interface.

Walrus performs the following tasks:

- allows consumers to store data permanently using REST and SOAP API compatibles with Amazon S3,
- stores VM images,
- allows consumers to access their VMs, images, and stored data, and
- manages the consistency of the stored data.

The storage controller (SC) performs the following tasks:

- allows the creation of snapshots of volumes and stores them in Walrus,

[3] The secure shell (SSH) is a protocol that provides encryption, authentication, and integrity protection to secure remote login over an insecure network [179].

- allows the creation of persistent block devices that can be attached to the instance file system (cannot be shared across instances), and
- provides raw unformatted block devices that can be used as any block device such as creating a file system on top of it.

The node controller (NC) performs the following tasks:

- hosts and controls virtual machines, which include:
  - checking the authenticity of users' requests to start instances,
  - fetching images from Walrus,
  - creating a virtual network interface,
  - executing instances, and
  - terminating instances.
- manages virtual network endpoints,
- takes instructions from the CC about how to manage the host OS and hypervisor, and
- responds to the CC queries about availability.

Eucalyptus networking modes:
In Eucalyptus, there are four networking modes: system, static, managed-NOVLAN, and managed. The *system* and *static* modes are the simplest and are not recommended for production environments because they do not offer the following essential services, [56]:

1. security groups, which is a firewall that can filter both ingress and egress traffic from VMs [171],
2. elastic IPs, which refers to the dynamic assignment of IP addresses to VMs, and
3. isolation of network traffic between VMs.

Managed-NOVLAN and *managed* modes offer security groups and elastic IPs. However, only the managed mode offers network traffic isolation between VMs. Therefore, we chose the managed mode for our deployment [56] because it is the normal choice for a public IaaS, where different VMs owned by different clients are sharing hosts and other cloud resources.

Virtual Machines:
In each NC, a limited number of virtual machines can be run concurrently. We can choose between a small, medium, large, and xlarge VM size. These differ in memory size, disk space, and number of CPUs. The maximum numbers of VMs in our deployment can be found using a command from Euca2ools, which is *euca-describe-availability-zones verbose*, see figure 4.7.

With 0 VMs running in the system, the number of *free* is equal to the number of *max*, i.e. free=4 and max=4. Before starting a new VM, we have to check the availability, so that the capacity and power of the new VM can be decided.

We initiated three VMs with small, medium, and x-large sizes. Each VM had two IP addresses: a public and private. The range of public IP addresses are between 192.168.10.200 and 192.168.10.2003; while the range of private IP addresses are between 10.10.1.2 and 10.10.1.5. The public IP addresses allow consumers to

```
client1@client1:~$ euca-describe-availability-zones verbose
AVAILABILITYZONE  cluster1  192.168.10.3
AVAILABILITYZONE  |- vm types  free / max  cpu  ram  disk
AVAILABILITYZONE  |- m1.small  0004 / 0004  1  192  2
AVAILABILITYZONE  |- c1.medium  0004 / 0004  1  256  5
AVAILABILITYZONE  |- m1.large  0002 / 0002  2  512  10
AVAILABILITYZONE  |- m1.xlarge  0001 / 0001  2  1024  20
AVAILABILITYZONE  |- c1.xlarge  0001 / 0001  2  1024  80
```

Figure 4.7: Maximum allowed number of concurrent VMs in same host.

access their VMs from outside the cloud network, while private IP addresses are used for the virtual subnet.

Applications in VMs:

One of the most popular uses of IaaS cloud services is for middleware applications. Therefore, in the VMs under examination, we ran an ERP application,[4] which satisfied the following criteria:

1. multi-tiers,
2. heavy weight transactions,
3. free or has a trial version, and
4. open source to be able to check the source code if needed.

Any multi-tier ERP application with heavy transactions could be chosen. After examining the available applications in the market, we selected *Tryton*. It is a three-tier application platform under the GPL-3 license (free and open source). It is written in Python and use PostgreSQL as its database engine [162]. It requires at least two VMs to work: one of them is the client and the other is the server. It has the ability to generate reports for the company who initiated and owns the VM and saves information databases.The modules chosen in Tryton were *timesheet management* and *inventory management*. Tryton deployment has 3 parts: the Tryton client, Tryton server, and PostgreSQL database. The Tryton client is used by users to access the server and to add, delete, modify, search, and generate documents, and to print reports. The PostgreSQL database is in the Tryton server and holds all of the persistent data. The only way to use this application is through the Tryton client, which has a graphical interface; the Tryton server cannot be accessed directly. We created three machines: two were Tryton clients and one was a Tryton server.

VMs specifications:

VM1 - Tryton client:

- public IP: 192.168.10.201,
- private IP: 10.10.1.3,
- name: i-5CD30A7F,

[4] ERP stands for enterprise resource planning and it is type of management systems (application) which aim to bring different functions within an enterprise onto a single system [129].

- operating system: Ubuntu 10.04 LTS - Lucid Lynx (amd64),
- kernel version: 2.6.32-21,
- size: c1.small 192 ram, 2 GB disk, 1 CPU, and
- PID: 27992.

VM2 - Tryton server:

- public IP: 192.168.10.203,
- private IP: 10.10.1.5,
- name: i-410F07B0,
- operating system: Ubuntu 10.04 LTS - Lucid Lynx (amd64),
- kernel version: 2.6.32-21,
- size: m1.medium 256 ram, 5 GB disk, 1 CPU, and
- PID: 4669.

VM3 - Tryton client:

- public IP: 192.168.10.202,
- private IP: 10.10.1.4,
- name: i-3F1C080F,
- operating system: Ubuntu 10.04 LTS - Lucid Lynx (amd64),
- kernel version: 2.6.32-21,
- size: c1.xlarge 2 GB ram, 80 GB disk, 2 CPUs, and
- PID: 14029.

For each VM, there should be a KEYPAIR, with private and public keys. These keys are used to validate a user's identity when logging into VMs via SSH. The keys are injected into the VM to allow only authorised users to access the VM using SSH, which usually does not require passwords. The public key is stored in Eucalyptus, and the private key is stored in a file such as *mykey.priv* and printed as standard output. The same KEYPAIR can be used for several VMs if required.

4.2.3 Data Collection

The detection system has to be trained using sufficient quality data to be able to build a representative normal behaviour profile for each VM and then detect any deviation from normal. We have to have sufficient data for training and also for testing the detection systems.

Many researchers in the area of HIDS have tested their techniques on data imported from other research (they did not collect the data themselves). This was because, as Mutz stated in [128], generating quality data is a problem in itself. In our situation, no quality data were available from previous research. Thus, we had three choices to obtain the required data:

1. Collect the data from an IaaS environment that is in production, such as Amazon EC2.
2. Collect the data from an IaaS environment that was built in the lab, with an attempt to make it as close as possible to an environment in production.
3. Collect data from a simulated environment.

The first choice was not possible for legal and security reasons (mistakes or errors could affect the cloud network in production and cause service disruptions, lost data, or other security threats). The third choice would have been the easiest, but we avoided it in an attempt to generate less artificial data. We chose to build an IaaS environment in the lab that was close as possible to a production environment. First, we used a configuration that is recommended for production, as discussed in section 4.2.2. Then, we used a Eucalyptus cloud management application that is designed to be similar to Amazon EC2 and is used in real IaaS clouds [55]. Furthermore, when we used the applications inside the monitored VMs, we added randomness factors to avoid generating spurious similarities among data sets and to mimic realistic usage patterns, as will be explained in more detail later in section 4.2.3.

The data quality affected the accuracy of the results; any misconfiguration or missed details could affect the reliability of the findings. For example, when collecting samples we had to be conscious of the tendency of some applications and operating systems to buffer data before sending them to the CPU, memory, and/or I/O devices. If this critical point was missed, and data for a short time period were collected while there was some buffering, the collected data would not have been accurate and might not have reflected the real activities occurring in the VM. As a consequence, any analysis with this data would have been unreliable. To overcome this problem, we collected data for a long period of time and compared them with data acquired within a short time period to determine whether there was any buffering in our system. This point is mentioned just to show how all the details of the experiment were vital.

Despite all of the considerations we took into account, we know that a real productive environment will be different. However, this is the best we could do at this stage until one of the IaaS cloud providers adopts the project or applies the proposed ideas. There is also another popular problem in the area of HIDS, classifier drafting, which is a large topic and left for future research.

Declaring the source of the data and the details of the collection are essential to evaluate the accuracy of the findings in order to either accept or reject these. In our case, as no suitable corpus of data was available, we built the required environment in the laboratory to generate the data.

After setting up the experiment as described in section 4.2.2, we collected all the system calls that were invoked by VMs being monitored. To collect the system calls, we used the Linux *strace* tool; it intercepts and records all invoked system calls by the chosen processes (which were VMs in our case). As explained in section 4.2.1, each VM in the system was represented by a process that had many threads. The *strace* tools was used to monitor the main process and all the threads under it. Each of the processes and each of the threads under them had a unique process identifier (PID). Different PIDs were assigned to the threads and processes which gave the monitoring system better insight into the VM activities.

Tracing tool: To collect the system calls invoked by the monitored VM, we used the *strace* tool. According to Linux's man page, *strace* "intercepts and records the system calls which are called by a process and the signals which are received by a process. The name of each system call, its arguments and its return value are printed on standard error or to the file specified with the -o option" [116]. We ran

strace in the NC machine with the option *-f*, which monitors the targeted VMs, in addition to monitoring any other process cloned from this main process. The option *-f* is useful to maintain the sequences of system calls. This is the description of the *-f* option on Linux's man page.

"Trace child processes as they are created by currently traced processes as a result of the fork(2) system call. The new process is attached to as soon as its PID is known (through the return value of fork(2) in the parent process). This means that such children may run uncontrolled for a while (especially in the case of a vfork(2)), until the parent is scheduled again to complete its (v)for (2) call. If the parent process decides to wait(2) for a child that is currently being traced, it is suspended until an appropriate child process either terminates or incurs a signal that would cause it to terminate (as determined from the child's current signal disposition)" [116].

We found that the VM processes in the experiment used the clone system call to create new threads. It is also important to mention here that not all the system calls are recorded by *strace* reliably. There is a possibility that a small fraction of system calls will be missed, depending on the host load patterns. It is difficult to know the exact percentage of missed calls, but we have to consider that some of those can be lost in the analysis.

We used the option *-o* to save the tracing results to a file to be analysed later. *Strace* with the *-f* and *-o* options generates a file for the targeted process. Each line of the file started with the *PID* and then the system call that is invoked by this PID; figure 4.8 shows some lines from our sample as an example.

```
4669 read(7, 0x7fff59ca8e70, 512) = -1 EAGAIN (Resource temporarily
unavailable)
4669 select(54, [7 10 12 14 15 53]. []. []. {1, 0}<unfinished ...>
4676 <... ioctl resumed>, 0) = 0
4676 lseek(9, 0, SEEK_END) = 5398069248
4676 writev(50, [{"\0\34\304#\200:\320\r-
R\6\22\10\0E\20\0004\36s@\0@\6<\203\n\n\1\2\300\250"..., 66}], 1) = 66
4676 ioctl(6, 0xffffffffc008ae67, 0x7f6937cc5b60) = 0
4676 ioctl(11, 0xae80, 0)               = 0
4676 ioctl(6, 0xffffffffc008ae67, 0x7f6937cc5c10) = 0
4676 ioctl(11, 0xae80, 0)               = 0
```

Figure 4.8: Sample of data collected using *strace* tool.

The Tryton ERP application is business management software that consists of a suite of integrated applications (modules) used to perform tasks such as accounting, invoicing, inventory management, and sales management. To generate normal traffic, we ran the Tryton system using two modules: *timesheet management* and *inventory management*. We used these modules for normal tasks such as filling in data on forms and spreadsheets, generating reports and exporting them to LibreOffice applications[5], printing reports as PDF files, and saving data to a hard disk. Using different combinations of these services, we designed more than 30 scenarios to

[5]LibreOffice is a free and open source office suite that embeds several applications such as word processing and spreadsheet applications.

generate sufficient traffic in the monitored VMs and then randomly moved among these scenarios. We monitored a Tryton client VM during regular activities, subsequently extracting system call traces.These scenarios were automated by a scripting mechanism (Python scripts and the Dogtail open source automation framework). Then, the system was exercised by them. To avoid generating spurious similarities among data sets and to mimic realistic usage patterns, the move from one scenario to another and the intervals within scenarios were performed randomly.

We ran these scenarios while collecting system calls using *strace*. After collecting a large amount of system call traces, we divided them into epochs of 1 GB.

Analysing system calls: Our experimentation revealed that only about 25% of Linux system calls were used by our VMs. Therefore, to save space and computational time, we only considered the used system calls and covered the rarely used ones by a new entry added to the list of used system calls called 'other'. This trick reduced the data analysis time and also reduced the time required to build the normal behaviour profiles in the *BoSC* and *HMM* methods, as we will see in section 4.3. We used a small amount of collected data to generate the list of used system calls, and then added the entry 'other' to the list. Each distinct system call in the small sample was an item in the *list of used system calls* variable. We also added the item *'other'* to the list to cover new system calls that were not in the small sample when they arrive in the future (in the training or testing phase).

KVM is dealt with by the Linux host as a *character device*, which is managed using the *IOCTL* system call.[6] Each Linux device is represented by a file and has two numbers (major and minor) and a node name to identify it.[7]

From LANANA, we found that KVM is registered as a char device with major number 10 and minor number 232 with default location /dev/kvm. /dev/kvm has a file descriptor with number 5.[8] There are more file descriptors; for instance, kvm virtual cpu (vcpu) has the fd 11. Each file (device) also has a magic number at the beginning of it to identify the file type. The KVM device magic number is 0xae. *IOCTL* system calls are used to manage VMs, and it has the following format, see figures 4.9 and 4.10. The untyped memory pointer is represented by dots because it could lead to an unlimited amount of data. [9].

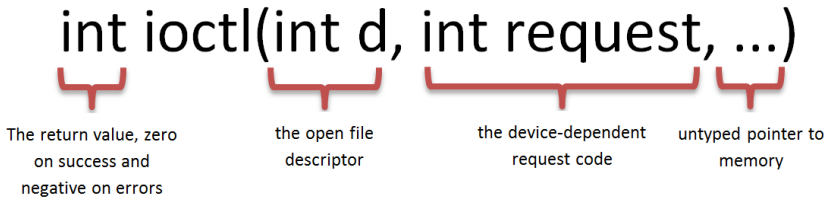

Figure 4.9: IOCTL system call format from Linux man page.

[6] *IOCTL* stands for input/output control and is used to manage all the devices in Linux.

[7] LANANA (Linux Assigned Names and Numbers Authority) is the authority that manages device names and numbers in Linux. Their website is http://www.lanana.org/.

[8] To check the file descriptor for any KVM device, go to any of the VM processes; for example, for VM2 go to /proc/27992/fd and then run the command ls -l or by using lsof command.

[9] The structure of IOCTL in our system can be found in /Usr/src/linux-headers- 2.6.32-28/include/asm-generic/ioctl.h.

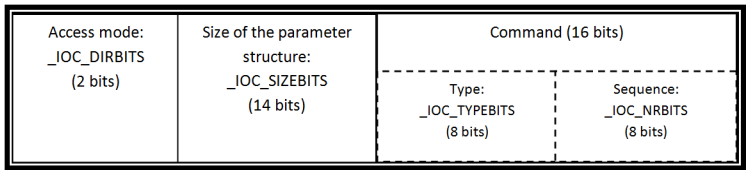

Figure 4.10: IOCTL system call structure in system.

There are four access modes in the *IOCTL* system call (_IOC_DIRBITS $2bits$), as follows:

- 00 = NONE ($\_IO\_NONE$ = no data transfer).
- 01 = WRITE ($\_IO\_WRITE$ = write operation).
- 10 = READ ($\_IO\_READ$ = read operation).
- 11 = READ and WRITE ($\_IO\_READ || \_IO\_WRITE$ = read and write operation).

The size of the parameter structure (_IOC_SIZEBITS 14 bits) is architecture dependent, and it represents the size of the user data. It is useful for error detection and backward compatibility tasks. Type (_IOC_TYPEBITS 8 bits) is the magic number, which is equal to 0xae in KVM. Sequence (_IOC_NRBITS 8 bits) is the KVM command number.[10]

It is essential to understand the structure of the *IOCTL* system calls to analyse the collected data. The following are two examples from our sample for *IOCTL*.

Example 1: IOCTL(11, 0xae80, 0) = 0

Example 2: IOCTL(5, 0xae03, 0x10) = 1

In example 1, the first argument is 11, which is the file descriptor for KVM vcpu. The second argument is 0xae80. 0xae is the KVM magic number, and 80 is the property of the KVM command representing the command KVM RUN. A list of KVM commands and their meanings can be found in Linux in the file kvm.h.

In example 2, the first argument is 5, which is the file descriptor for /dev/kvm. The second argument is 0xae03. 0xae is the magic number and 03 is the KVM CHECK EXTENSION to get the size for mmap (vcpu fd).

It is clear that the IOCTL commands in the previous examples requests absolutely different actions from the kernel. Therefore, we consider each IOCTL system call with the first and second arguments as a different system call. Therefore, "IOCTL(11, oxae80," and "IOCTL(5,oxae03," are registered as two different system calls in the list of distinct system calls.

How much data should we collect?

For testing purposes, we collected a large amount of data, more than 100 GB of sequences of system calls generated from normal activities on the monitored VM. However, the target is to use the minimum possible amount of data for training to be able to build the normal behaviour profile as fast as possible to suit the dynamic feature of the cloud. We stopped the training process when a consistent normal behaviour profile was created. To decide if a profile is consistent enough or not, we used a method similar to that previously reported by Warrender *et al*. in [170].

[10]Different commands can be found in this file /usr/src/linux-header-2.6.32-28/include/kvm.h.

According to their report, "one way of establishing a consistent measure of how much training data to use across several programs is to set a target for the slope of this growth curve. Once the rate of encountering new sequences drops below some preset value, we say we have enough data with which to build our model of normal."

Data generated from normal behaviour:

We automated the process of generating data (the normal usage of the application) followed by the collection of the generated data using the *strace* tool. We traced two processes (Tryton client VM and Tryton server VM). However, we could not trace the two processes concurrently because the *strace* tool can monitor only one process at a time. Therefore, we monitored the client first and then the server.

Approximately 60 hours for the Tryton client and 12 hours for the Tryton server were needed for collecting the data required for training.

List of training samples:

- 12 hours of normal usage traffic for VM3 (Tryton client). The size of the sample is about 28 GB.
- 24 hours of normal usage traffic for VM3 (Tryton client). The size of the sample is about 45 GB.
- 24 hours of normal usage traffic for VM3 (Tryton client). The size of the sample is about 37 GB.
- 12 hours of normal usage traffic for VM2 (Tryton server). The size of the sample is 411 MB.

The difference in the sample sizes is because we intended some randomness in the automation code to allow the program to run differently each time. Furthermore, the Tryton client processes the majority of the work and therefore generates more traffic.

For testing purposes, we also had to collect abnormal sequences of system calls generated from abnormal activities on the targeted VM. The target was to build a detection system that is able to detect abnormal activities in VMs, whether these activities are malicious, errors, or intended changes of behaviour, any *change of behaviour* whether it is legitimate or malicious. Therefore, to generate the abnormal testing traffic, the VMs' behaviour could be changed either legitimately or by an attack.

Generating malicious traffic

To generate anomalous system calls, an attack should be conducted against the monitored VMs. There are several ready-to-use attacks in the wild; however, performing an IaaS specific attack with the assumption that the attacker is conscious of the cloud environment and the used technologies is more valuable. We designed a DoS attack that targeted two of the main features of the cloud: 'migration' and 'over-commitment'. We started by writing a very simple piece of code to stress the CPU of the monitored VM and collect the generated sequences of system calls during the stress test. The code was not malicious (regular CPU stress testing), but if there was coordination between the VMs to increase the demand and trigger migration for false reasons, the effect on the cloud would be harmful. We started with a very simple malicious scenario just to change the monitored VMs' behaviours. The CPU utilisation was increased by running non-malicious stress test in the VMs. A

regular stress test was chosen because, unlike worms and viruses, it would not be captured by a network-based IDS, would not be blocked by firewalls, and there was no need to download any code from the Internet, because the code was in the operating system package (which was Ubuntu based on Linux 2.6.35).

The stress test code was as follows:

```
stress.sh
#!binsh
cd usrsrclinuxsource2.6.35
time make
done
```

Before running the stress test code in the VM, the host processors registered an idle time of approximately 96%; after running the stress test in one of the VMs, it fell from 96% to approximately 48%. This was a result of running the stress test in only one of the two VMs. The large drop in the host CPU idle time encouraged us to search more in the area of host over-commitment and continue developing the attack to become a sophisticated cloud-specific DoS attack, referred to as a Cloud-Internal Denial of Service (CIDoS) attack; the details of the sophisticated version of this attack are shown later in chapter 5.

4.3 Detecting Anomalies in VMs Through System Call Analysis

We tested two methods to build the system calls hypervisor-based IDS, which were the bag of system calls (BoSC) from [93] and hidden Markov model (HMM). The proposed systems help IaaS providers achieve the following. First, they detect any change in a VM's behaviour to raise a security alert, send notifications to the correspondent client, and/or trigger investigations about the reasons for the behaviour changing. Second, they help providers to evaluate a VM's security status by combining information about each VM from the HIDS and NIDS. The evaluation of each VM's security status helps providers to make future decisions and take actions, i.e. to notify consumers to improve their machine's security and/or change the contract terms and conditions or even charge security ignorant consumers an extra amount of money for the risk they pose to the provider's network and infrastructure.

The general hypothesis for this chapter is that a normal behaviour profile can be built for IaaS-hosted VMs by monitoring the sequences of system calls invoked by them.

The consistency of the VMs' behaviours

In 1996, Forrest *et al*. stated in the paper "A sense of self for unix processes" [63], that when building the first system calls anomaly detection system, they monitored only root processes because these usually have relatively consistent behaviours. Their research was based on two assumptions. The first assumption was that programs have consistent behaviours, which generate a pattern of consistent sequences of system calls. Consistency is essential for building a steady profile that represents the monitored process. The second assumption was that during an attack or errors, unusual sequences of system calls will be generated.

Their research inspired us to develop new cloud monitoring solutions based on similar assumptions because of the similarity between VMs hosted in the IaaS cloud and root processes; we assumed that VMs hosted in the IaaS cloud are also limited in the used services and applications.

This assumption was relative to regular servers, which may aggregate multiple services on a single instance. Regular best practices for IaaS cloud services would argue against deploying such mixed workloads in a single VM for security and performance reasons, as we discussed in detail at the introduction of this chapter.

This assumption leads us to assume that VMs have relatively consistent behaviours; and consistency is essential to be able to build a normal behaviour profile that represents the monitored VM accurately. Furthermore, in general, servers usually tend to perform the same series of tasks repeatedly, which provides a sound data set for training. This assumption drives us to the further assumption that in the IaaS cloud, any change in VM behaviour should be treated as a sign of abnormality. This is how the idea of the proposed monitoring solutions started.

Assumptions: We based the building of the normal behaviour profile representing cloud VMs on three assumptions:

1. VMs generate sufficient data to train the classifier in a timely manner.
2. VMs are not initially malicious and hence can provide a valid initial training data set. In the context of our research, this assumption is acceptable; if the monitored VM is malicious from the beginning, this means either the client who initiated this VM is an attacker or the VM was hijacked the moment it started. These two possibilities are not the security problems we investigate here; they are rather related to authentication mechanisms and the process of initiating new VMs in the cloud, and outside the scope of this research.
3. VMs are limited in the used services and applications. This assumption is relative to regular servers that may aggregate multiple services on a single instance. Regular best practices for IaaS cloud services would argue against such mixed workloads being deployed in a single VM for security and performance reasons, as described in section 2.7.1. This assumption leads us to assume that VMs have relatively consistent behaviours; and consistency is essential to be able to build a normal behaviour profile that represents the monitored VM accurately. Furthermore, in general, servers usually tend to perform the same series of tasks repeatedly providing a sound data set for training. This assumption drives us to the further assumption that in the IaaS cloud, any change in VM behaviour should be treated as a sign of abnormality.

4.3.1 Detecting Anomalies Using BoSC

4.3.1.1 Hypothesis

The hypothesis behind the first detection system (BoSC) is that a normal behaviour profile can be built for VMs hosted in an IaaS environment using sequences of system calls, and a *strong enough* anomaly signal is generated when comparing abnormal sequences of system calls with the normal behaviour profiles. The normal behaviour profiles are built using a bag of system calls representation method, which ignores the order of system calls and only relies on frequencies.

4.3.1.2 Training the Classifier Using BoSC

We trained and evaluated the new hypervisor BoSC-based HIDS using the data sets obtained from the experiment. BoSC was originally proposed by Kang *et al*. in [93]; it retains only the frequency of system calls without relying on the ordering (or adjacency) of individual events. We registered sequences of system calls using a sliding window technique.

First, we created a list of distinct system calls using a small sample of the collected data. Then, we added the item 'other' to the list to cover rarely used system calls. The generated list of system calls was used to structure a database containing the normal behaviour profile of each VM. Using Python, we wrote an algorithm to train the classifier using BoSC and create the normal behaviour profile. A series of 1 GB epochs were the input of the algorithm; the first one was used to create the table representing the VM, and the following epochs were used to update the table until reaching a consistent version that represented the normal behaviour profile of the VM. The data were read from the epochs line by line, where each line represented a system call.

To measure the consistency of the generated table and thus decide when to stop training, we calculated the distance between the table before and after each update, as we will see later in the section.

The frequencies of each system call in the sequence were counted, and if the same frequencies were found in the database, the corresponding frequency field was increased by one. If the frequencies were new, they were not found in the database and had to be added as a new field with a frequency of one.

The length of the sequences should be determined before running the algorithm. The length size 6 is usually recommended to provide the best performance [78]. We tested length sizes of 6, 10, and 20, and then compared their results to determine the effect of changing the length. The output of the algorithm was a stable normal behaviour database representing the monitored VM. This database was used later for detection.

Profile stability:

If the database of normal behaviour registered a slight change for several continuous epochs, then we concluded that the classifier was sufficiently trained and ready for use.

To calculate the change in the classifier before and after each epoch, we defined metrics based on the difference in the frequencies of sequences.

To stop training and announce the classifier as trained, we needed to satisfy two conditions: (1) the difference in the frequencies of sequences had to be less than a specific threshold, and (2) this difference had to be under that threshold for at least two sequential epochs.

$S_1 < T$ and $S_2 < T$

- S_1: The difference in frequency for the previous epoch
- S_2: The difference in frequency for the current epoch
- T: Threshold value = 200 for sequence size 10 (chosen by experiment)

By applying this formula we found that only 6 epochs were needed to train the classifier for a sequence size of 10.

4.3.1.3 Evaluating the Classifier

After training the classifier and creating the normal behaviour profile for the monitored VM, the next step was testing the efficiency of the classifier. We ran sequences of anomaly system calls and observed whether the classifier would recognise these as abnormal, with an acceptable cost (time and space) and efficiency and efficacy.

We wrote a simple detection algorithm for classification; it compared the data generated for testing with the normal behaviour profile of the monitored VM and calculated the difference in the frequencies of the sequences.

Efficacy analysis:

The system was efficient if the abnormal system calls were recognised, and the system generated a strong enough anomaly signal. We could say that the anomaly signal was strong enough if the difference between Q_1 and Q_2 was greater than the threshold T.

T is equal to 382, and it is calculated using the following equation: $(T = |B - A|/2)$, where A is the average difference in frequencies for normal samples, and B is the average difference in frequencies for malicious samples.

Q_1 is the difference in frequencies between the normal behaviour database of the VM and normal epochs.

Q_2 is the difference in frequencies between the normal behaviour database of the VM and malicious epochs.

This equation $(H_0 : Q_1 - Q_2 > 382)$ is true if malicious epochs register a large difference in frequencies in comparison with normal epochs, when compared with the VM normal behaviour profile.

The problem of deciding whether the generated anomaly signal is strong enough or not, can be formulated in terms of hypothesis testing as follows:

1. The null hypothesis: $(H_0 : Q_1 - Q_2 > 382)$
   $(H_1 : Q_1 - Q_2 <= 382)$ (one sided hypothesis)
2. Assume (H_0) is true
3. The difference in frequencies follows an approximately normal distribution, see figure 4.11. We considered 64 samples, and then grouped them depending on the frequency difference into 6 groups. We found that a low difference of frequency and very large difference in frequencies were closer to zero, as shown in the figure.
4. Level of significance $(\alpha = 0.001(99.9\%)$ confidence level)
5. Find Z scores: $(Z_\alpha = Z_0.001 = -3.09)$
6. Find the region of rejection RR which is a set of values less than or equal to $(\alpha) : ((RR <= -3.09))$
7. Collect samples
8. For each epoch, we extracted the difference in frequencies for sequences and calculated some statistics, as shown in table 4.6.
   $(SD = \sqrt{\frac{\sigma_1^2}{N_1} + \frac{\sigma_2^2}{N_2}} = 35.5)$

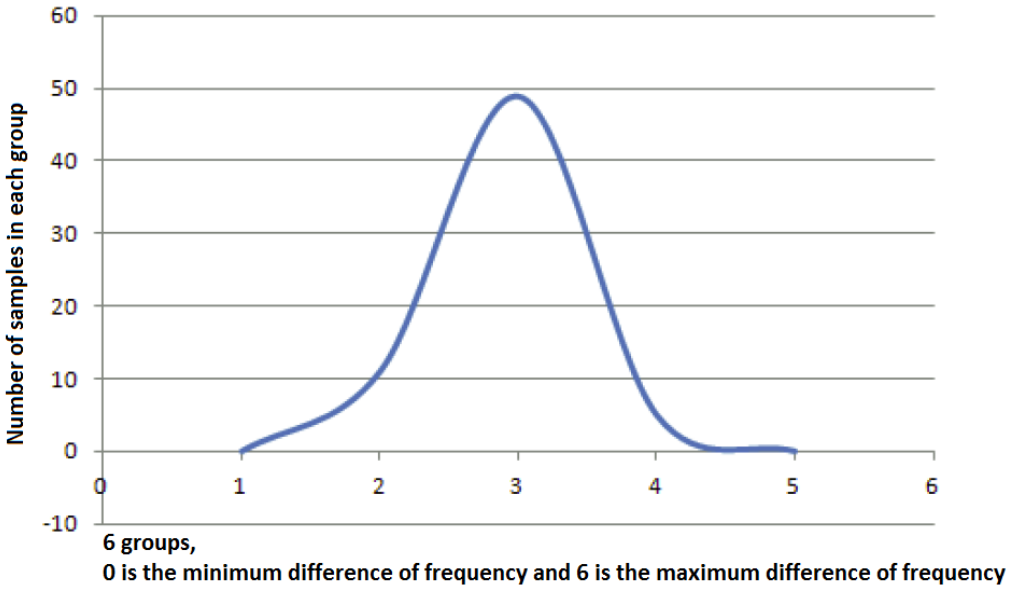

Figure 4.11: Difference in frequencies for sequence size of 10 follows normal distribution.

Table 4.6: Statistics calculated from samples for BoSC-based classifier.

Normal	Malicious
N_1 = 8	N_2 = 8
μ_1 = 68	μ_2 = 860
σ = 3	σ = 100

$(\Delta = 382)$ (the value in the null hypothesis)

$(\mu = \mu_2 - \mu_1 = 793)$ the new centre

$(ME = Z_\alpha * SD = Z_0.001 * SD = -110)$

Range: 683 to 902

$(Z = \frac{\mu - \Delta}{SD} = 12)$

9. Draw a conclusion: the test statistics Z = 12 is not in the RR so we retain the null hypothesis.

Widely accepted criteria for measuring the efficiency of an HIDS are the accuracy, detection rate, and false positive rate [93].

Accuracy $= \frac{\text{\# of } TP + \text{\# of } TN}{\text{\# of Input Sequences}}$

Detection Rate $= \frac{\text{\# of } TP}{\text{\# of } TP + \text{\# of } FN}$

False Positive Rate $= \frac{\text{\# of } FP}{\text{\# of } TP + \text{\# of } FP}$

TP = true-positive, FP = false-positive, FN = false-negative, and TN = true-negative.

The results for sequence size 10 are shown in table 4.7

Complexity analysis:

Table 4.7: Results with sequence size of 10. The anomaly signals are very strong, which generate 100% accuracy for the BoSC-based classifier.

Sequence Length	Accuracy	Detection Rate	False Positive Rate
10	100.0%	100.0%	0.00%

The time complexity is of the utmost importance in a cloud-based intrusion detection system. A computationally efficient algorithm that can be applied in almost real-time to train the classifier and perform the classification is required. In this section, we calculate the computational complexity of the used algorithm, along with the space consumption.

The used algorithm has a linear time complexity of $O(n)$, where n is the number of lines in the input file. The average number of lines in one epoch is $n = 11743281$, and there is a loop of n. Inside this loop, we have an array of 25 digits (number of distinct system calls), where each digit represents the frequency of this system call in a group of 6 consequence system calls (for sequence length 6). This array is added to the database of normal behaviour if it does not already exist. If it already exists, its frequency in the database is increased by one. The sliding window technique is used to receive new system calls and drop old ones.

In the worst case scenario, there are 70 instructions in each iteration of the loop. The sequence length is x_2, and the number of rows in the database is x_3. The equation for the number of instructions in our algorithm, in the worst case scenario, is $n(2x_2+70)+x_3$. If x_2 is 6 and x_3 is on average 2534, then the number of instructions using this formula is 82. For space consumption, the normal behaviour database has 3419 rows (for sequence size 6) and two columns, one of the char type and the second of the integer type. The database is relatively small, and only a few lines are added to it with each new epoch. We needed 5 epochs to train the classifier. Thus, our algorithm has a relatively low cost, which makes it suitable for real-time environments, and the generated database is small, which allow it to be migrated with the corresponding VM.

4.3.1.4 Discussion

Our results prove that it is possible to build a normal behaviour profile for a whole VM via system calls monitoring using the bag of system calls representation method. We successfully created an identity, and a sense of self, for the targeted VM. The used mechanism also successfully detected anomalies and generated strong enough anomaly signals. Three different sequence lengths were tested, sizes 6, 10, and 20. Table 4.8 shows a comparison between these. Size 6 is the best if we need a very fast detection system; while size 10 provides the best accuracy and false positive rate, but with a slightly higher consumption of time than size 6. Size 20 is not efficient because its accuracy and false positive rate are equal to those of size 10, but it has higher computational time demands in comparison to sizes 6 and 10.

The training algorithm has linear complexity, which makes it fast enough to be applied in real-time environments. The other algorithm is the detection algorithm, which is very simple and short; the bottleneck of this algorithm is the database. The algorithm's speed depends on the chosen sequence length; with an increase in

the sequence length, the consumed time increases significantly (see table 4.8). For instance, with a sequence size of 6, the detection algorithm produces the results almost instantly.

Table 4.8: Results with sequence lengths of 6, 10, and 20 for BoSC-based classifier.

	SIZE 6	SIZE 10	SIZE 20
Accuracy	98.36%	100.0%	100.0%
Detection Rate	100.00%	100.0%	100.0%
False Positive Rate	11.11%	0.00%	0.00%
Consumed Detection Time by Units	32	324	2492

Other interesting observations were found. First, the anomalous system calls generated using the stress test code had bigger arguments than normal ones, which made the 1 GB epoch of malicious data have fewer lines than that of the normal data. Malicious data also registered more new system calls than normal data, along with system calls under the 'other' category that had never been seen in the training data. Second, malicious data also registered as more unfound in the database sequences (new sequences) than normal data.

As we can see, the proposed system satisfies the special requirements for a cloud host-based IDS shown in the previous chapter, chapter 3. Because the used representation method is simple, it can deal with a large amount of data. The system can monitor VMs with almost no intrusiveness and requires no prior knowledge about VMs. The detection system is separate from the monitored VM. Therefore, it has a high attack-resistance. It can work on the fly due to fast training, monitoring, and detection. It is also small and autonomous. Therefore, it can be migrated with the corresponding VM. Lastly, it does not require any human intervention, which is essential for a large-scale cloud. However, there are other problems in dealing with ephemeral VMs, as discussed below.

4.3.2 Detecting Anomalies in Ephemeral VMs

The method proposed in section 4.3.1 (BoSC) is a frequency-based representation, which is simple and requires no probability calculation. By using this method, the classifier generated strong anomaly signals with high detection rates. However, we found that about 6 GB of data were needed to train the classifier, with 1 GB for testing. This is a relatively large amount of data, which is acceptable in long term VMs, especially given that a high volume of flow data is one of the prominent characteristics of cloud environments. However, for ephemeral VMs, there is a need for a classifier that can be built faster, even if it consumes more computing resources, if not time.

Requiring more resources to build this classifier, rather than time, is acceptable in the cloud, where resources such as computational power are not scarce because it is the main service offered by the cloud. Hence, with these requirements in mind, we modified our first classifier by changing the machine learning approach and the part of the data being analysed to reduce noise. We built a hidden Markov model (HMM)-based classifier.

HMM is a sophisticated representation method, which usually provides a high detection rate and low false positives, but has a high computational demand [170]. There are many research papers about how to improve HMM for anomaly detection by reducing the power usage (such as in [81], [155] and [80]); however, in this research, we tested the regular HMM.

4.3.2.1 Hypothesis

The hypothesis is that a normal behaviour profile can be built for ephemeral VMs hosted in an IaaS environment using limited number of system calls sequences, and a strong anomaly signal can be generated when comparing abnormal sequences of system calls with the normal behaviour profiles that are built using an HMM representation method.

There are two main requirements to be satisfied by the proposed system. First, the required amount of data to train the classifier should be kept to the minimum to be able to build the profile for ephemeral VMs within an acceptable time frame. Second, the model should provide acceptable accuracy, detection, and false positive rates. There should also be a balance between efficiency and efficacy, in addition to satisfying the special requirements for cloud security monitoring tools discussed in section 3.1.12.

One of the methods we used for pre-processing the collected system calls is to categorise them based on their threat level, as in [21]. Another method is to only monitor specific threads, rather than the whole process representing the VM.

We want to build an HMM-based classifier that generates a matrix representing the normal behaviours of VMs and detects any change in behaviour.

4.3.2.2 Modelling VM's Normal Behaviour Using HMM

HMM is known to provide a very powerful model to capture the structure of sequential data. It generates two sequences of symbols, where the first is observable, and the second is hidden. The hidden states can be discovered only through the observable states. The probabilities of transitioning from one hidden state to another and from a hidden state to an observable state are represented using two probability density functions. The model is denoted as $\lambda = \{A, B, \pi\}$.

HMM symbols:

- A: the transition matrix of size N*N that stores the probabilities of transitioning from one state to another and is row stochastic (the sum of each row is equal to 1).
- B: the observation matrix of size N*M that stores the probabilities of an observable event occurring given that the system is in a certain hidden state, which is also row stochastic.
- π: the initial state probabilities of each state being the start of the sequence. It is of size M and is row stochastic.
- T: length of the observation sequence (the number of observations taken).
- S: S_1, S_2... S_N is the set of hidden states with N items.
- O: O_1, O_2... O_M is the set of observable states which is the set of used distinct system calls with M items.

The collected system call traces of the targeted VM are considered to be the HMM observation sequence, and each system call is an observation symbol. Before building and training the classifier, we need to first specify the size of the HMM. The HMM consists of two finite sets of states, hidden states set S with N items and an observable events set O with M items. It is a common practice in HMM anomaly detection classifiers based on system calls to choose a size for N equal to the size of M, which is equal to the number of distinct system calls [76]. In our case, the number of distinct system calls is 25. The trick of using the 'other' item in the list of used system calls makes it possible to cover the rarely used ones and reduce the number of states in the HMM model, which significantly reduces the computation time.

After specifying the size, HMM becomes the stage for pre-processing data to train and then test the classifier. We used three samples: two were normal and one was malicious. The first normal sample was used to train the classifier, while the second and third were used for testing purposes.

4.3.2.3 Training Steps for HMM Classifier

To train the HMM classifier, we went through the following steps:

- Step 1: we prepared normal samples for training.
- Step 2: we processed the samples by removing errors, incomplete system calls, and IOCTL system calls, which had a missed first or second arguments. The data were pre-processed using three different methods:
  - Method one: we considered all system calls that were invoked by the monitored VM. This method is the main method; however, we will show the results of methods two and three in the results and discussion sections.
  - Method two: we considered only IOCTL system calls and system calls with threat level one from the list of used system calls. The threat level of each system call was imported from [21], but later modified depending on the research environment.
  - Method three: each VM was represented in the system with one process, but this process had many threads or children processes. In this method, only the main threads were considered. We defined the *main thread* as the thread that initiated the IOCTL system calls because in our environment (KVM-based virtual environment), an IOCTL system call was the most critical system call for the reason that VMs communicate with resources in the host kernel using IOCTL.
- Step 3: we took a small part of a normal sample and used it to create the list of used distinct system calls, and then specified the value of M, which was 25 for methods one and three and 8 for method two.
- Step 4: we transferred system calls to vectors of numbers; for instance system call $'ioctl(11, 0xae80,'$ was represented by number 10, and the system call called $'other'$ was represented by number 24. The list of numbers represented the items of the observable states O.

- Step 5: we converted all the system calls in the training sample to the corresponding numbers and divided them into groups of k, using a sliding window, $k = 6$.
- Step 6: we determined the number of sequences of system calls used to train the classifier. Since this classifier targeted ephemeral VMs, the number of sequences used for training had to be kept to the minimum. The number of sequences for training and testing were chosen by experiment, as shown later in the section.
- Step 7: after determining the number of sequences to train the classifier, we trained the classifier using the Baum-Welch algorithm, which is an expectation maximisation algorithm, to find the unknown parameters of HMM. The inputs of the algorithm were sequences of system calls of size 6, and the output was a trained classifier to be used later in the testing phase.
- Step 8: end the training process.

Testing the classifier

To test the classifier, we used 1500 traces (1000 were normal and 500 were malicious) collected in the experiment discussed in section 4.2. We input these to the classifier and then calculated the log-likelihood of each trace, and also calculated two sets of values, Set_1 and Set_2. Set_1 was the difference between the log-likelihood of 500 normal traces TR_1 and 500 abnormal traces TR_2. Set_2 was the difference between the log-likelihood of TR_1 (the same normal traces used in Set_1) and another 500 normal traces TR_3. To label traces as normal or malicious and generate the anomaly signal, two thresholds were designed. The first threshold T_1 was used to decide if a trace was normal (represented by 0) or a mismatch (represented by 1). The value of this threshold was chosen by experiment.

For each entry in TR_1 and TR_2, we calculated the difference in log-likelihood and compared it to T_1. If the difference was less than T_1, the value was set to '1', and the trace was registered as a mismatch, otherwise the value was set to '0', and the trace was registered as normal.

The second threshold T_2 was used to decide if a chunk of traces was normal or malicious, which determined the final decision to send an anomaly signal or not. Each chunk of traces contained 10 traces. If a chunk of traces registered over 4 mismatches, it was considered to be a malicious trace, and an alert was raised. However, we found that about 50% of malicious chunks registered 10 out of 10 mismatches, which meant all the traces in the chunk were malicious. If the chunk had 4 or less mismatches, it was considered to be a normal chunk. However, about 60% of the normal chunks had 8, 9, or 10 out of 10 normal traces in them. Dividing traces into chunks helped to make the classifier works on the fly using a small amount of data for detecting anomalies. When the classifier started processing a chunk and it registered 5 mismatches, there was no need to continue processing the rest of the chunk because 5 was enough to label it as malicious. In this case the classifier could ignore the rest of the chunk and move to the next one. The choice of T_2 and the number of traces in each chunk were defined by experiment.

The amount of data used to train the classifier was relatively small. We used about 780,000 system calls to train the classifier, which was about 19 MB of raw data

containing system calls, along with their arguments and return values. For testing 1500 samples were used each of them is of size 150 KB. Each sample contains 100 chunks of size 1.5 KB (10 traces $\approx$ 600 system calls). In 50% of the samples, the classifier was able to decide if a chunk was malicious or not by checking only the first 75 bytes of the chunk; in half of the samples (50 chucks, 75 KB) we can decide by checking only $\approx$ 38 KB.

4.3.2.4 Evaluating the Classifier

After training the classifier using the HMM method, the output of the training algorithm was the normal behaviour profile representing the monitored VM. The next step in this process was testing the efficacy of the classifier.

Efficacy analysis:

The problem of deciding whether or not the designed HMM-based detection system generates strong enough anomaly signals could be formulated in terms of hypothesis testing as follows:

Q_1 is the number of anomaly signals in malicious samples in comparison with normal sample, and Q_2 is the number of anomaly signals in normal samples in comparison with another normal sample.

1. The null hypothesis: $H_0 : Q_1 - Q_2 > 5$

   $H_1 : Q_1 - Q_2 <= 5$ (one sided hypothesis)

2. Assume H_0 is true

3. The difference in anomaly signals is based on the difference in the log-likelihood and follows approximately normal distribution, see figure 4.12. We considered 9710 samples and then grouped these depending on the difference of log-likelihood into 13 groups. We found that a low difference of log-likelihood and very large difference of log-likelihood were closer to zero, as shown in the figure.

4. Level of significance $\alpha = 0.001$ (99.9% confidence level)

5. Find Z scores[11]: $Z_\alpha = Z_{0.001} = -3.09$

6. Find the region of rejection RR which is a set of values less than or equal to $\alpha : (RR <= -3.09)$

7. Collect samples

8. Extract the difference in anomaly signals for sequences for each chunk and calculate statistics, as shown in table 4.9. N is the sample size, and μ is the average.

   $SD = \sqrt{\frac{\sigma_1^2}{N_1} + \frac{\sigma_2^2}{N_2}} = 0.320$

   $\Delta = 5$ (the value in the null hypothesis)

[11] Z Score is a statistical measurement represent the relationship of a score to the mean in a group of scores.

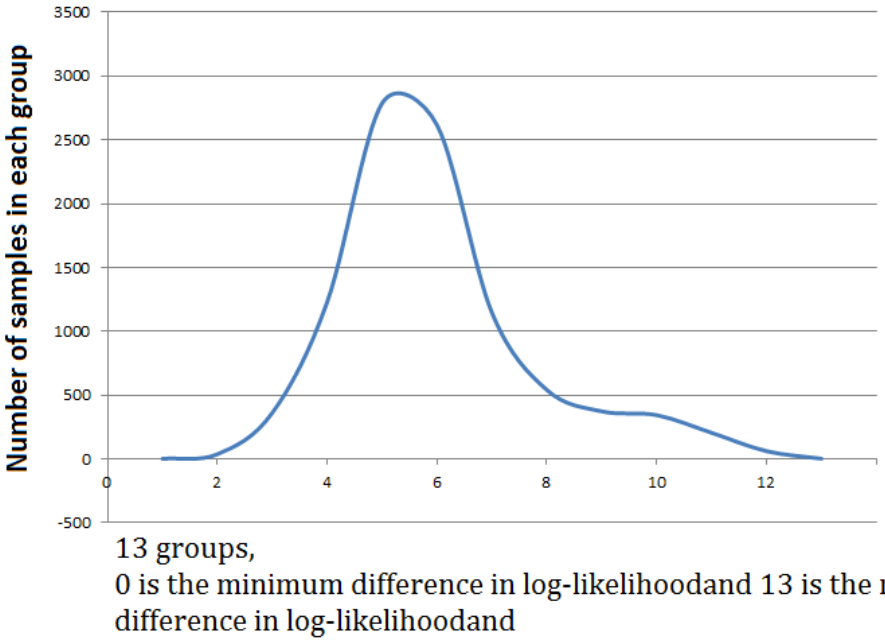

Figure 4.12: Difference in log-likelihood between normal and abnormal samples follows normal distribution.

Table 4.9: Statistics collected from samples.

Normal	Malicious
$N_1 = 8$	$N_2 = 8$
$\mu_1 = 0.625$	$\mu_2 = 9.625$

$\mu = \mu_2 - \mu_1 = 9$ the new centre

$ME = Z_\alpha * SD = Z_{0.001} * SD = -0.99014$

Range: 8.06 to 9.99

$Z = \frac{\mu - \Delta}{SD} = 12.48303$

9. Draw a conclusion: the test statistics Z = 12.48303 is not in the RR so we retain the null hypothesis.

The accuracy, detection, and false positive rates criteria are generally used to measure the efficiency of a detection system. The results of testing 1500 samples with a chunk size of 10 and sequence length of 6, where each sample contained 1000 sequence and 8 iterations, are shown in table 4.10.

Table 4.10: Results of testing 1500 samples with chunk size of 10 and sequence length of 6, where each sample contained 1000 sequences and 8 iterations.

Accuracy	Detection Rate	False Positive Rate
97%	100%	5.66%

Complexity analysis:

Two main factors that affect the complexity of algorithms are time and memory. For the Baum-Welch algorithm, the time complexity scales linearly per iteration with the sequence length and quadratically with the number of HMM states (number of distinct system calls in the list) [97]. The memory complexity also scales linearly with the sequence size and number of states [97].

The time complexity is $O(N^2k)$, and the memory complexity is $O(Nk)$ [96]. Therefore, to reduce the required computational complexity, we reduced the value of N to an acceptable level. Although Linux has about 326 different system calls, we only considered 25 of them, which reduced the time complexity considerably.

Results:

We found that method one provided the best results, with a high accuracy and detection rate and an acceptable false positive rate in comparison with the other two methods. Method two, when only $IOCTL$ system calls were monitored, provided a detection rate of $\approx$ 83%. For method three, when only the main threads were monitored, 3 main threads were found; the first one provided a very low anomaly signal, while the other two failed to distinguish between normal and malicious samples.

Furthermore, we compared the detection rate of the system when using different sizes of training samples (195,000 system calls, 390,000 system calls, and 780,000 system calls); we applied this only to method one because methods two and three had failed. The result of the comparison is shown in table 4.11.

Table 4.11: Different sizes of training samples for HMM classifier.

# of System Calls	195000	390000	780000
Detection Rate	$\approx$ 93%	$\approx$ 96%	$\approx$ 97%

After obtaining these primitive results, we decided to use the 780,000 system calls for training and afterward applied the chunk approach to generate the results shown earlier in the section.

4.3.2.5 Discussion

Method one detected all of the malicious samples but also labelled some normal chunks as malicious with a false positive (FP) rate of 5.66%. The FP rate could be lowered by reducing the threshold T_1. For this, the average difference could be divided by 4 instead of 2 (as used here). This will decrease the FP rate but will increase the false negative one. Therefore, in addition to other factors, the choice of this threshold may depend on how critical this VM is. Providers should design this threshold based on the nature of their clients, their industry, and the geographical area.

We argue that method two failed because $IOCTL$ system calls were not the main system calls to distinguish the malicious attack we used (stress test attack). Other attacks might be detected by monitoring mainly $IOCTL$ system calls; however, deciding which system call to monitor is a complicated task and going deeper in this track might convert the anomaly detection system into a misuse detection system (signature-based).

The reason for the failure of method three might be that we monitored the wrong threads. Threads in general are not stable because they are created and terminated continuously; therefore, the approach of monitoring specific threads under the main process of the VM should be flexible and may change with time. More research can be done in this area to find a better definition of 'main threads' rather than 'the thread that initiates $IOCTL$ system calls', which is the definition used in this research. We argue that by monitoring the right threads, the training and detection time will be less, and the accuracy will improve.

This proposed system satisfies the special requirements for a cloud host-based IDS, in addition to providing the ability to detect anomalies in ephemeral VMs. The system processes the collected data (system calls) in a way that reduces the complexity of HMM while being able to detect anomalies reliably in ephemeral VMs.

4.4 Similar Systems

In the literature, there are number of systems close to what we proposed here. The closest and strongest candidate was designed by Azmandian *et al*. in [14]. We discussed their IDS, which is called VMM-IDS, briefly in the state of the art chapter, section 3.3. In the VMM-IDS, the hypervisor was modified in order to be able to extract and collect low-level architectural events from VMs while running. The monitored events include the execution of privileged instructions, memory access, and I/O use. The hypervisor logs *all* events and the related information in order to reconstruct the actions of VMs (bridge the semantic gap). The logged events and information include for example Read/Write events with the control register number, its value, and if it was read or write, which helped to understand the VM operating system state. Another example involved page faults with the register number, its value, and whether it was read or write, which helped to understand how VMs' applications use memory.

Azmandian *et al*. divided the extracted features from collected events into two groups: (1) features that are generated from the sum of events; and (2) features that are generated from the relationship between events. Their system was tested in a virtualisation-based environment (but not a cloud computing one). As explained in section 3.1.3, high volume of data is one of the main characteristic of the cloud. Registering all events and then analysing them using machine learning algorithms might prevent the IDS from working on the fly and make it more of a log analyser rather than a detection system. We believe that the detection system of Azmandian *et al*. needs to be tested in the cloud context.

Zou *et al*. published two interesting papers closely related to our topic. In their first paper, [174], they proposed a VM monitoring system for virtualised platforms called VMDriver. It has two components. The first is in the hypervisor, which is responsible for intercepting and collecting events (system calls, registers, and memory pages) and then sending them to the second component. The second component exists in a separate VM, called the management VM (MVM), which is responsible for analysing the collected low-level binary data and reconstructing the semantic information based on prior knowledge of the monitored VM's operating system. This method of bridging the semantic gap is called "OS-dependent se-

mantic reconstruction" [174]. Each monitored VM has a corresponding monitoring driver in the MVM. What is interesting is that they can encapsulate the needed information about the monitored VM's OS with its monitoring driver, and the driver can be loaded dynamically when the VM arrives. We think that this idea suits cloud environments, because the monitoring drive can be migrated with the VM. In their second paper, [191], in 2013, they added more features to their old system to be able to work in the cloud. The problems the researchers tried to solve were security monitoring and trust within the cloud. They proposed trusted monitoring for cloud-based platforms. They used trusted computing to maintain the integrity of the monitoring systems. They also suggested placing the MVM under the management of a trusted third party to solve the problem of trust in cloud computing. We believe that the systems proposed by Zou *et al*. have a great potential to work efficiently in the cloud and help solve the problem of security monitoring in it. However, they require prior knowledge. Our idea of monitoring VMs through system calls using only low-overhead and simple methods can be integrated with the system of Zou *et al*. systems. This might improve the trust and security monitoring dilemmas in the cloud.

4.5 Attacks Against System Call IDS

Sekar *et al*., in [147], stated that anomaly detection systems that are based on system call monitoring (without considering arguments) will struggle to detect the following attacks:

- Attacks that rely on system call arguments, i.e. "attacks involving les accessed via symbolic links" [147]. To cover this type of attack, many research papers have suggested considering system call arguments in the used detection method [157] and [128].

- Attacks that do not cause any change in the behaviour of the targeted VM. Usually these attacks are related to the network, and the proposed systems are HIDSs, which are not a substitute for the NIDS.

- Attacks that are designed specifically to work in a host monitored by a system call-based anomaly HIDS. An example of this type of attack is *the mimicry attack* [169]. In a mimicry attack, the attacker designs its attack to achieve malicious purposes using sequences of system calls that look normal (mimicking the sequence of system calls of normal behaviour) [128], [108], and [169]. Therefore, in our context, the attackers have to analyse the normal sequences of system calls of the VM and design their attack accordingly.

  Almost all anomaly detection systems (without arguments) that use sequence of system calls are threatened by this type of attack, including the BoSC-based detection systems that we used in this research. A BoSC-based detection system is even easier to mimic because it ignores the sequence of system calls and only relies on frequencies. However, our systems are less threatened by a mimicry attack because we monitor the whole VM with all of its applications and services, rather than a single application or process. In contrast, most other system call-based HIDSs monitor only one application or process

within the host, giving the attacker the chance to analyse the behaviour of the monitored application and also giving him/her the chance to inject a code to mimic the normal sequences of system calls. In our context, the attacker might be detected while injecting the exploit code or while analysing the normal sequences of system calls. Our systems are still threatened by a mimicry attack, but the exploit code is harder to design, and the process of injecting system calls might cause the invocation of new system calls, which might be detected by the detection system.

4.6 Conclusion and Future Work

In this chapter, we proposed two anomaly detection systems that work by monitoring all of the system calls invoked by the VMs hosted in a public IaaS cloud. The monitoring systems work efficiently and successfully detect changes in the behaviours of VMs without the need to access VMs or collect data from inside them; they rely exclusively on the system calls invoked by VMs to detect any behaviour change. System calls were collected and used to create a normal behaviour profile representing the VM, and any change in behaviour was detected as an anomaly. The main assumption for our detection systems was that VMs have relatively consistent behaviours because they are limited in their used services and applications. This assumption was relative to regular servers, which may aggregate multiple services on a single instance. Regular best practices for IaaS cloud services would argue against deploying such mixed workloads in a single VM for security and performance reasons. The new systems satisfy the IaaS public cloud special requirements for detection systems; they can work efficiently without accessing VMs, and they require no prior knowledge about VMs. They cannot be disabled by VM users because they are installed in the cloud host rather in the VMs themselves. They can work almost instantly (fast training, monitoring, and detection). They generate small normal behaviour profiles, which can be migrated with the correspondent VMs in the case of migration. They need no human intervention to work, but are fully automated, with no need to update rules or insert any detection signatures.

The first detection system was built based on the BoSC representation method, and for a sequence size of 10, it achieved 100% accuracy and 0 false positives; it generated very strong anomaly signals.

The other detection system was designed to monitor ephemeral VMs, which are VMs that live for short periods of time to perform specific tasks and then terminate. The model of ephemeral VMs is very popular in the public IaaS cloud. The detection system was built using the hidden Markov model representation method. The system worked efficiently, providing an accuracy of 97% and a false positive rate 5.66%. We tested three methods: monitoring all the system calls, monitoring only system calls with a high threat level, and monitoring only threads (processes) that initiated IOCTL system calls. The first method one provided good results, whereas the second and third methods failed to distinguish normal and abnormal behaviours and provided a weak anomaly signal or no signal at all, respectively.

We processed data to reduce the time and space needed for monitoring and detection. Only the used system calls were considered (not all the Linux system calls); by considering only the used system calls and creating the 'other' item to

cover those rarely used, we reduced the amount of processed data to the minimum. This step also significantly reduced the HMM states and the database needed by the BoSC.

Further research is needed to cover the following topics:

- Testing improved versions of HMM to detect anomalies in ephemeral VMs.
- Considering system call arguments in the classifier.
- Investigating the problem of classifier drafting, especially for ephemeral VM anomaly detection systems.
- We assumed that the VMs are not malicious from the beginning. However, if they are, the anomaly detection system will fail. This area needs more research; the problem is related to authentication mechanisms and the process of initiating new VMs.
- Investigating the threat of a mimicry attack, while monitoring the whole VM by a system call HIDS, rather than just one application or process.

Chapter 5

Attacks in IaaS Cloud

Sharing resources, co-residency, pay-for-use, over-commitment, and migration are essential features for a public IaaS model to operate efficiently and effectively. In contrast, they introduce new security concerns and needs to the cloud. Recently, many studies have been conducted to discover the vulnerabilities and investigate possible techniques to misuse these features and attack the cloud or its consumers, such as in [139], [183], and [164]. Public IaaS cloud computing is a *service*, and disturbing this service is the first main threat facing the cloud.

In this part of the research, we introduce a cloud-specific denial of service family of attacks that is based on misusing the 'migration' and 'over-commitment' features of the cloud. These attacks also employ co-residency, co-residency checks, and covert channels to disturb the service. We call this new class of attacks *CIDoS*, which stands for *Cloud-Internal Denial of Service*.

In the CIDoS class of attacks, a group of malicious VMs coordinate their resource consumption to stress some of the cloud resources, aiming to trigger a migration order for fake reasons. Attackers increase their consumption of resources by building on the already existing consumption (riding the workload wave) to reach thresholds (time and strength) that trigger a migration order. The scale of this attack has the potential of causing a complete paralysis of the cloud. If attackers successfully force the cloud to enter a continuous state of migration, the whole service will have difficulty functioning.

VMs migration

By studying the IaaS cloud, we found that VM migration is a main feature in the cloud that allows load balancing, rapid elasticity [125], and cost reductions by maximising the utilisation of hosts or evacuating rarely utilised hosts to save power. VM migration is essential; however, at the same time, it has a high cost. If it is an online migration, it requires many synchronisation steps to transfer VMs without disturbing their work, which consumes the network and host resources. If it is an offline migration, it requires turning off VMs before migration, which disturbs the service and might affect the users. Furthermore, migration should be done without the consumers' awareness, and without requiring that they adapt to or track the change in location [15].

The process of moving a VM from host to host is very expensive because it consumes bandwidth, computational power, and affects security by increasing the attack surface. In migration, the whole VM is transmitted through the network, which gives the attacker the chance to attack the VM passively by eavesdropping for sensitive data or information. If the attacker succeeds in taking a snapshot of the VM in the transmission channel, this will give him or her the chance to attack the VM offline without raising suspicion or leaving any trace. All parts of the VM have

to be migrated, including the running memory, which increases the danger of data loss and requires tools to maintain the integrity and confidentiality of the transmitted data. Furthermore, authentication and authorisation mechanisms should be in place, and the security of the source and destination servers should be considered. Finally, there should also be monitoring and alerting mechanisms for VMs during migration to detect any stealing, tampering, or dropping of the data. Migration decision mechanisms might also be a target for attackers, initiating migration orders for false reasons or cancelling a valid order, as we did in the CIDoS attack.

Cloud host over-commitment

Cloud hosts tend to oversubscribe or over-commit based on the assumption that not all of the co-resident VMs will use their resources to the maximum at the same time [182]. By over-committing cloud servers host more than they can afford, which reduces the cost without affecting the service. The aim is to provide adequate services with the minimum resources (i.e. turn *on* the minimum number of hosts to save power, which is the most money consuming factor in the cloud, about a third of the costs of the datacentre [11]).

Over-commitment could also be a source of threats to the cloud, as stated in [20], "the host is oversubscribed; that is, if all the VMs request their maximum allowed CPU performance, the total CPU demand will exceed the capacity of the CPU." This is what we call misusing over-commitment, and it is most harmful when there is a rapid coordination between groups of malicious VMs.

Migration and over-commitment are essential to efficiently manage a cloud's resources. Because migration is known to be expensive, if we could trigger migration orders for false reasons, we might affect the availability of the service, especially if a large number of migration orders are triggered.

Cloud attacks

In the literature, we found that most of the published cloud attacks follow the same generic theme: they require co-residency, co-residency checks, and the use of covert channels for communication or stealing data or resources. We used the same theme to develop our new cloud DoS attack.

A CIDoS attack requires the following steps. First, it requires that numerous malicious VMs (over a specific threshold) coexist in the same physical host. Second, the attack class relies on the use of covert channels for communicating and coordinating the attack among attack members. Covert channels are communication channels that are not designed to carry information, and are typically beyond the authorisation of access control media. In covert channels, flaws in the isolation between resources are exploited to enable communication [173]. The severity of the covert channel is measured by the channel capacity. Ensuring the elimination of all covert channels is extremely difficult if there are shared resources.

Third, the attack class operates a novel protocol through the covert channel to coordinate and trigger the attack. When attackers are ready to attack, by the end of a successful protocol operation, each of the participating VMs increases its utilisation following the expected workload pattern of the host and building over it in an attempt to break the host's ability to cope with the increased stress (a technique similar to jamming signals). The attack is hard to detect because attackers maintain steady behaviours close to their usual workloads. The attack deceives the cloud by stressing some of the host resources, showing that the host is very busy for a pe-

riod of time that is long enough to trigger migration and short enough to not raise suspicion.

In this chapter, we introduce the idea of the CIDoS attack, describe the novel protocol used to coordinate the attack among malicious VMs, and investigate different mitigation strategies that involve prevention, detection, and response. The description in the chapter is for the theoretical version of the attack only.

The designed protocol is based on a group key agreement protocol (GKA) to establish a session key, packet delay variation (PDV) to synchronise the attack time, pulse-code modulation (PCM) to digitise the workload patterns, and leader election protocol to choose a leader for the attack.

For mitigation strategies, we base our detection mechanisms on correlation measurements and distance calculations among the attackers' workload patterns. We use the simple one dimensional discrete cosine transform (DCT) and Euclidean distance to measure the correlation.

After that, the attack is improved by designing a stealthy randomised probing strategy to learn the parameters and thresholds used in the process of making migration decisions in the cloud (reverse engineering the migration algorithms). In a CIDoS attack (without the improvement), it is assumed that the workload thresholds that need to be broken to trigger migration are known to the attackers, whereas, in reality, they are hidden and discovering them requires the design of a new attack, which is the CIDoS attack improvement. We discuss the ideas, concepts, and design of the improvement theoretically in the last part of this chapter, which will be verified in future work.

We perform sequential runs of the attack to extract and reveal the parameters and thresholds of migration algorithms. The attack is based on CIDoS, together with a statistical analysis. We also design a formal model for the migration decision process, and then create a dynamic algorithm to extract the required hidden parameters. The mechanisms to extract these thresholds are adapted to the dynamic changes in cloud management algorithms. Revealing the parameters is a security breach in itself. Furthermore, they can be used in a CIDoS attack to allow the malicious VMs to accurately generate the needed workload to cause the required effect (trigger migration). It is vital that the generated workload is not more than required to reduce the possibility of detection and to make the attack last longer (attackers will try to keep the host on the edge). They can also use their understanding of the applied migration policies and algorithms to avoid being migrated.

The rest of the chapter is organised as follows. The literature dealing with the CIDoS class of attacks is discussed in section 5.1. Section 5.2 shows the threat model, while the details of the attack and the proposed protocol are presented in section 5.3. The mitigation of the attack is discussed in section 5.4. Section 5.5 discusses the improvement of the attack, and section 5.6 presents the conclusion and future work.

5.1 CIDoS Attack Literature

The first step of the attack is attaining co-residency. Many techniques have been proposed in the literature to achieve co-residency and/or verify co-residency in

the cloud. Table 5.1 lists the results of a survey of techniques found in literature used for co-residency and co-residency checks.

Table 5.1: Survey of different techniques for co-residency and co-residency checks.

Technique	Co-residency	Co-residency check
Using cloud Tomography [139]	✓	
Brute force [139] and [164]	✓	
Using covert channel as a ground truth [139]		✓
Abusing new instance placement locality [139]	✓	
Through cross-VM attacks [184]	✓	✓
Using network probes [139]	✓	
Active traffic analysis [19]		✓
TTL scan [75]		✓
Interrupt-overloading side-channel [75]	✓	✓

5.1.1 Co-residency and Co-residency Checks

In a cloud network, there are *regions* and *availability zones*;[1] each region resides in a geographical area, and it has multiple isolated locations (availability zones). Availability zones are offered for consumers to plan their VMs locations in a way that increases availability (by distributing them in different availability zones to escape full failure if one of the locations has fallen). The first step to increase the likelihood of co-residency is for an attacker (who is a cloud consumer) to initiate all of their new VMs in one region and one availability zone [139].

Ristenpart *et al.*, in [139], aimed to map the cloud internal infrastructure (cloud cartography) to find the location of a targeted VM (a victim) and then initiate co-resident VMs to attack the victim using a cross-VM attack. The research covered two parts. First came the *advantageous placement*, which involves initiating VMs in a way that maximises the likelihood of co-residency. The second is a cross-VM attack to steal cryptographic keys using side channels, which we have already discussed in chapter 3. Researchers tested their techniques on Amazon EC2, and they were proven to be successful.

To achieve co-residency, Ristenpart *et al.* proposed and tested two methods. The first one involves performing a network probe using nmap, hping, and wget. To retrieve the external name of a VM and translate public IP addresses into internal addresses, they used DNS resolution queries. Then, a list of internal IP addresses was collected. They hypothesised that each availability zone has dedicated internal IP address ranges. More than 400 VMs were initiated. Observations generated using network probing were collected and analysed. Amazon EC2 interactions were studied to map the cloud network and understand the used placement algorithm. For instance, by studying the observations, researchers found that Amazon

[1] These are the names used by Amazon EC2, which is the leader of the field.

EC2 distributes VMs belonging to the same client to different physical hosts. They also found that the maximum number of co-resident VMs of size small in one host is eight. Furthermore, they found that Amazon EC2 has *'strong placement locality'* when a VM has been terminated just before the second has been launched, and the second will often be placed in the first VM's ex-host. Amazon EC2 also has *'parallel placement locality'*; when two VMs belonging to different clients are launched at the same time, they will often reside in the same host. Information like this is valuable to attackers, and makes it easier to achieve co-residency.

A brute force technique for co-residency was suggested in [139]. Brute force works by launching a large number of VMs and terminating the ones that are not co-resident. Researchers proved that this simple strategy registers reasonable success rates [139].

After achieving co-residency, Ristenpart *et al*. performed a co-residency check using different methods. The first method is matching VMs' Dom0 IP addresses by checking the first hop out *from* the VM or the last hop *to* the VM using a TCP SYN traceroute. The second method is using round-trip times to calculate the time it takes for a signal to arrive and its acknowledgment to return. After checking the co-residency, they verify it using a covert channel based on hard disk usage. If two VMs successfully communicate a message with acceptable false positive rates through the covert channel, they are co-resident. Thus, the covert channel is used to prove co-residency *as a ground truth*.

When a VM in the Ristenpart *et al*.'s model, [139], wants to send a '1' through the covert channel, it writes in a fixed location in the hard disk, and the receiver VM calculates the time needed to read from the same location (calculate the delay). If the reading time is higher than a threshold, the other VM is trying to send a '1'; a delay occurs because the location is busy. To send a '0', the sender does nothing. Consequently, the receiver will read from the fixed location faster, below the threshold (no delay), and will decode it as '0'.

The problem when using network probes to attain co-residency is that popular providers are aware of the threat they might pose. Thus, they are blocked, and most of them are not applicable nowadays [19] and [75]. Other more advance network-based co-residency check techniques were proposed in [19] and [75].

Researchers in [19] employed a network covert timing channel to perform a co-residency check. The attacker has to have a management machine outside the cloud premises. It initiates a large number of VMs hosted in the cloud and check for co-residency using a network covert channel and statistical analysis. The attacker's target is creating a malicious VM that shares the same host with the victim VM (which is known to the attacker beforehand). The victim VM has to have an Internet-facing service, and the attacker will initiate a web session with it.

The attacker creates a large number of VMs to increase the possibility of co-residency. Each of these, in turn, will perform some network activities (inject onto the network interface a stream of UDP packets) to congest the hardware responsible for the outbound interface and multiplexed with the victim traffic, causing a delay in its flow. The attacker will try to generate a recognisable pattern of *delay* and *no delay* network flow, which represents the watermark signature of co-residency. When the management machine detects a watermark signature, co-residency is achieved by the VM in turn. The management machine uses statistical measure-

ment to decode signals. They use a Poisson distribution and Kolmogorov-Smirnov distribution test [19]. Researchers conducted an experiment to test the scheme and successfully proved co-residency in less than 10 s.

In [75], Herzberg *et al*. designed a cloud attack consisting of three steps. The first retrieved the internal IP address of the victim VM using its external publicly known one. They used an interrupt-overloading side-channel to discover the internal IP address. This is a technique based on the fact that operating systems give higher priority to inbound packets than outbound ones. In the interrupt-overloading side-channel, the attacker overloads the host hardware with I/O device interrupts; these interrupts occur due to a burst of packets (UDP datagrams). The attacker uses two machines: the first one is called the *client* and is used to establish a connection with the victim via its public IP. Then, the second machine, which is a VM hosted in the same cloud and is called the *prober*, sends bursts to different addresses in the clock range. The client inspects its connection with the victim looking for timing patterns that might be generated if the prober burst the right internal IP address, causing delays in the sent packets. If a pattern is found, this is the correspondent internal IP address. There should be a synchronisation between the *client* and *prober* to agree on the timing, burst size, and frequency. This technique has been tested on Amazon EC2 using a burst of 1000 packets and three probers. The success rate for discovering the internal IP address was above 60%. Discovering the internal IP address is just the first step of this attack; the main goal is checking for co-residency with the victim VM, and then attacking it using side channel attacks or inter-VM attacks.

Discovering the internal IP address is irrelevant to our attack because we need to prove co-residency among our malicious VMs, not between a victim and the attacker. The second step of Herzberg *et al*.'s attack is counting the number of hops between the victim VM and the malicious VM. A *time to live* (*TTL*) scan is used to count the hops between the two VMs. The attacker knows the victim VM's internal IP address, from the previous step. They then create a series of TCP SYN packets and send them to the victim. The first TCP SYN packet in the series has TTL = 1, and the field increases by one with each sent packet. The TTL value decreases by one with each hop, and once it reaches 0 on the way to the destination, the packet is discarded and an ICMP error is returned to the sender.[2] When the packet arrives at the destination, a SYN/ACK will be sent to the attacker, and the TTL field value in the successfully arrived packet is the number of hops in between the two VMs. If the value of the successful TTL is 0, this might indicate co-residency; however, as stated in [75] "this depends on whether the hypervisor reports itself as a hop on the path from the instances on a physical host." The proposed hop counting technique has been tested on different providers and proved successful in Amazon and Rackspace Cloud [75].

To prove co-residency, Herzberg *et al*. suggested using an interrupt-overloading side-channel after having TTL = 0. In this model, the attacker has three machines: the *client*, *prober*, and *probed* (a possible co-resident VM that requires the co-residency

[2] In Amazon EC2, ICMP messages are blocked to prevent the network from being scanned. However, locking the ICMP messages has only a slight effect on the attack, because upon failure (not reaching the destination) nothing will return to the sender (no failure acknowledgment), which means the packet has been discarded along the way [75].

check). First, the client will establish a connection with the victim VM. Then, the prober will use the interrupt-overloading side-channel (described above) to send a burst to the probed VM, which will overload the host NIC and introduce delay in the victim machine's outbound traffic if they are co-resident. The client will inspect its communication with the victim looking for timing patterns, and if they exist, that is a co-residency signal in addition to TTL = 0.

5.1.2 Covert Channels in Cloud

In covert channels, shared resources are used *indirectly* to leak information to unauthorised VMs or to communicate data among VMs in an unauthorised manner, which violates the security policy. The covert channel's capacity is used to measure its severity, the number of bits they can carry per second, in addition to the error rate.

There are two types of covert channels, storage based and timing based. In storage covert channels, the sender writes in a shared storage location, and the receiver reads from it, either directly or indirectly. In timing covert channels, the sender uses the resources in a way that creates a timing pattern that can be observed and decoded by the receiver, such as inter-packet delays or disk access delay [33].

In [139], Ristenpart *et al*. designed and tested three different covert channels in the Amazon EC2 cloud. The first one is a memory bus covert channel, and by experiment it was able to transmit 0.006 bps.[3] There are no design and implementation details about the memory bus covert channel in [139]; however, Wu *et al*. in [173] studied and tested this type of covert channel in the cloud, and we show their findings later in the section.

The second tested covert channel in [139] is a hard disk based covert channel, and by experiment it was able to transmit 0.0005 bps in the Amazon EC2 cloud environment. The idea behind this covert channel is simple; to transmit '1' read from the shared hard disk to keep it busy, and to send '0' do nothing. Meanwhile, the receiver will try to read from the same shared hard disk, and if there is latency, the disk is busy, and the sender is trying to send '1' otherwise it is '0'.

The third covert channel in [139] is an L2 cache-based covert channel, and by experiment it was able to transmit 0.2 bps in the Amazon EC2 environment. The L2 cache line is divided into two sets: *even* and *odd*. The sender evicts the *even* set to send '1' and the *odd* set to send '0'. The receiver measures the access time of each set; the delay in reading the *even* set decodes as '1', and the delay in reading the *odd* set decodes as '0'.

5.1.2.1 L2 Cache Covert Channels

In this type of covert channel, the two co-resident VMs communicate by constructing an agreed pattern of contention on the shared L2 cache (if it is shared), and the sent message is encoded by the receiver VM using the contention patterns. The main problem of using this type of covert channel in the cloud is that most cloud hosts have multiple CPUs and core migration (changing the CPU assigned to a VM)

[3] 0.006 bps is less than the minimum acceptable government rate for covert channels, which is 0.1 bps [135].

happens regularly [176]. Another problem is the interfering from other VMs sharing the same resource (L2 cache) as the sender and the receiver. In [176], researchers found by experiment that in Amazon EC2, about 50% of the time, a VM remains in the same core for no more than 10 ms, and 25% of the time for no more than 100 ms, which caused a large number of invalid reads.

To overcome the previous problems, Ristenpart *et al*., in [139], suggested that the receiver take the average of multiple samples to decide if it is a '1' or '0'. Furthermore, when the core has changed (core migration), the receiver will revert to the prime stage. Another solution was suggested by Xu *et al*. in [176] to deal with the core migration problem, which is repeated sampling (resend the bit multiple times). However, the receiver might read the same bit a number of times and consider each bit to be new data. There is *'no guarantee'* [176], especially because there is no form of acknowledgment. This covert channel provided a transmission rate of 3.2 bps with an error rate of 9.28% when it was tested in Amazon EC2 [176].

The same problems facing L2 cache covert channels in the cloud were recognised in [173]. Furthermore, they found that some processor *cores* do not share any caches, even if VMs are running in the same processor [173].[4] Moreover, in virtualised environments at context switches (changing the user of the resources), the L2 cache is completely flushed [173]. As a solution to all of these problems facing cache-based covert channels in the cloud, and as an alternative, Wu *et al*. in [173] suggested memory bus covert channels.

5.1.2.2 Memory Bus Covert Channels

In [173], researchers designed and implemented a protocol for cross-VM covert channels based on the memory bus, and it was dedicated to cloud environments. The covert channels they created were very wide; they successfully communicated at 700 bps in the lab and 100 bps in the Amazon EC2 cloud with error rates of 0.09% and 0.75%, respectively. Researchers designed a protocol to establish the covert channels and communicate through them. Their protocol also adopts a new scheduling technique (instead of strict round robin) that works instantaneously with VMs in parallel, which accelerates the communication.

The newly designed covert channels rely on the memory bus to communicate instead of a shared cache, which has many problems functioning in the cloud, as explained in section 3.4.

The main disadvantage of Wu *et al*.'s covert channel is that it has a very high noise rate. Therefore, error correction techniques (i.e. Reed Solomon) are required. In addition, to increase the reliability of the channel, the confirmation of the reception techniques can also be added. The sender should continuously observe the memory access latencies while the receiver is in execution (by performing exotic atomic memory operations). If there are no latency, the sender will assume that the data has not arrived and a resending is required.

5.1.3 Workload Prediction in Virtualised Environments

In a CIDos, the attack leader needs to predict the workload of the host to plan the attack pattern and distribute it among attackers. To monitor the workload of the

[4] Such as an Intel Core2 Quad processor [173].

host in virtualised environments, many measurements can be combined to estimate the current workload and then predict the future workload using algorithms from the machine learning field. Some of these techniques are based on covert channels such as in [139], and others are based on a network such as in [19]. These are only two examples to show the realism of assuming the ability to monitor and predict the host workload.

Bates *et al.*, in [19], successfully monitored the workload of a VM and extracted accurate traffic information from it using a malicious co-resident VM by injecting a watermark signature into the network traffic of the victim. The details of this attack were previously shown in section 3.4.

The researchers in [139] suggested using the host CPU caches to measure the workload by taking load samples to measure the number of CPU cycles and correlating them with the use of the cache (latency in reading from the cache). If there is latency in reading from the cache, that indicates activities in co-resident VMs. They proved that measuring the workload through shared caches is effective by testing it on an Amazon EC2 machine.

5.2 Threat Model

For service providers, disturbing the service is a main concern. Affecting the availability might cause a reputation loss and/or SLA breach, which in turn might affect the revenue of the provider.

A public IaaS cloud is threatened by traditional DoS attacks. However, these kinds of attacks are known threats that have already been studied and understood. We instead focus on designing a new class of denial of service attacks aiming specifically to disturb the service of a large-scale public IaaS cloud.

Cloud DoS attacks can target either the *implementation* of the cloud or its *architecture*. When targeting the implementation, the provider will patch or fix the bug that causes the vulnerability, and the threat is gone. On the other hand, architecture-based threats are harder to eliminate without changing or improving the base of the cloud. Therefore, we focus on cloud architecture-based attacks that target the availability of a large-scale public IaaS cloud.

In the threat model, we consider three parties: the first party are the attackers (we play this role). The second party are composed of defenders, which are security systems and cloud administrators. The third party are other attackers who might interfere with our attack for their benefit. The victims of this attack are benign cloud consumers and/or the cloud service provider.

The attackers in our model could include the following:

- other cloud providers (dishonest competitors) who want to affect the revenue and reputation of their competitors,
- a competitor to one of the cloud consumers,
- a disgruntled ex-employee, or
- hackers who want to prove that they can intrude in the cloud.

The defenders are the security systems in the cloud, cloud administrators, clients who will try to protect their virtual servers, and possible third parties.

In the attack we have the following:

- an attack leader that is selected using a leader election protocol,
- m number of malicious VMs,
- co-resident benign VMs,
- a cloud host where malicious VMs live together with other innocent VMs,
- hosts management machines where host management algorithms are run, and
- a protocol to coordinate the attack.

We make the following assumptions:

- We assume that the migration feature is enabled in the cloud service (some cloud providers disable migration support for security and performance reasons; however, migration is a main feature of the cloud that is essential for load balancing and to allow rapid elasticity [125]).
- Virtual machines (virtual servers) that have a relatively steady workload pattern. Mixing services in one VM is against best practice, especially since the cost model in the cloud is pay-for-use.
- We can predict the host workload pattern based on the workload history (at least for the next short interval of time), as we discussed in section 5.1.3.
- Noise in the communication channel for attackers is dealt with, i.e. by using an encoding scheme to correct errors over the covert channel.
- The leader knows the capacity of the communication channel. This can be done by testing the covert channel, although this aspect is outside the scope of this research.
- The attack is targeting generic resources in the host; however, it can work by targeting any type and number of resources in the host.

The attacker needs to do the following:

1. Increase the probability of co-residency.
2. Determine if its VMs are co-resident on the same 'physical' machine or not.
3. Establish covert channels to coordinate and communicate the attack.
4. Trigger the attack.

Many methods can be used to spread the CIDoS attack,[5] such as the following:

- uploading an infected operating system image in a public repository,
- using the same techniques as worms and viruses to spread the attack,
- spreading the attack through a cross-VM attack,
- exploiting weak authentication methods, and/or
- targeting vulnerable services in VMs.

[5]Spreading the attack is outside the scope of this paper, and these are only examples.

5.3 Robust Coordination of Cloud-Internal Denial of Service Attacks

5.3.1 Attack Coordination Protocol

A protocol was designed to coordinate the CIDoS class of attacks. The requirements, design, analysis, and evaluation of the protocol are discussed in this section.

5.3.1.1 Protocol Requirements

The main goal of the protocol is coordinating the attack from the arrival of the first malicious VM to the moment of attacking. To perform this coordination, a secure communication channel should be established among participants using a shared session key agreed upon by all participants. The protocol should allow only *legitimate* members of the attack to participate (preventing interference from adversaries).

The time should be synchronised just before triggering the attack (the final step of the protocol) to be able to attack almost simultaneously in order to make an effect. The attackers should not raise suspicion to avoid detection; therefore, the amount of the covert channel used should be reduced to the minimum effective rate. The freshness of the messages should be guaranteed to defend against a replay attack.[6] After the last check of *still alive* participants before attacking, the attack should be started as quickly as possible to avoid losing attack members over time (by migration) and to avoid changing the expected workload pattern.

The attack should be triggered only if there are sufficient resources to cause harm (to avoid faulty tries which might result in the detection and blocking of the attackers). The protocol should be robust to suit the dynamic nature of VMs. So it has to tolerate failures. The complexity of the protocol should be the minimum to suit the covert channel's limitations.

The identities of the participants should be hidden to avoid blocking them even after completing the attack. To hide the participants' identities, the messages exchanged among them should not contain any form of ID, including values that might help reveal the true identity of an attacker such as IP addresses or any values derived from it. Furthermore, the participating VMs should maintain a semi-random behaviour (workload pattern) to avoid raise suspicion.

The protocol should also resist substitution and de-synchronisation attacks.[7] It has to be portable and independent of the environment's technical details. Moreover, it should provide a form of authentication to prevent faulty VMs or adversaries from forging or modifying the messages.

These are the general properties and requirements. Additional properties and requirements for the *group key agreement protocol*, which is part of our protocol, are discussed in the following section.

[6] A replay attack is an attack in which an adversary inserts modified or old massages into a communication channel [82].

[7] A substitution attack is when an attacker, in the course of transmission, masquerades as the sender or the receiver of a message by reusing legitimate messages [38]. While a de-synchronisation attack is when an attacker tampers with the authentication capability of a valid node while communicating with another valid node by forging messages to destroy the synchronization [107].

5.3.1.2 TJKT Attack Coordination Protocol

Our protocol requires agreement on both the set of participants and the attack initiation itself. This can be accomplished efficiently by agreeing on a shared secret in a group whose membership is not pre-determined and which may change at any time. This can occur for various reasons, including faults in participants such as potential attackers being migrated from the target host to another. A number of such protocols exist. Because membership in the group is dynamic, and no trusted party can be assumed, no simple and efficient *group key distribution* (GKD) protocol can be used. Instead, a *group key agreement* (GKA) protocol provides the necessary properties. A large number of protocols have been proposed, with some notable contributions by Burmester and Desmedt [30], the GDH and TGDH protocols by Steiner *et al*. and Kim *et al*. [154] and [98] for the case of passive adversaries and variants able to tolerate active adversaries [94].

As noted, of particular interest for attack coordination is the need to tolerate faults; this has been the subject of extensive investigation. While the protocol by Kim *et al*. [98] exhibits some fault tolerance, more recent work has sought to improve the communication and computational complexity [86], [186], and [72] under different assumptions, with Cachin and Strobl describing a constant-round robust GKA [34] with communication complexity $O(n^2)$. Our proposed protocol is based on the protocol by Jarecki *et al*. [86], which provides constant-round complexity and simultaneously achieves the communication and computational complexity $O(n+T)$ for T errors to be tolerated. Although a large number of protocols have been proposed that offer logarithmic communication complexity such as the seminal work by Burmester and Desmedt [31] and subsequent enhancements to robustness [186] and [72], most such protocols rely on point-to-point and hierarchical communication topologies to achieve the $O(\log\ n)$ communication complexity (see Hatano *et al*. [72] for a detailed analysis of the communication complexity of several important protocols).

Unlike in the case of standard GKA protocols, however, we do not require a fixed set of participants to participate in the protocol, but only a group of a certain magnitude. Provided that the elements of this *attack set* can remain synchronised, we also permit dynamic changes in the group membership during the attack.

We assume that all the participants share the same g value; in addition to the assumption of Jarecki *et al*.: "Let G be a cyclic group of prime order q, and let g be its generator. We assume the DDH and Square-DDH problems are hard in G" [86]. For example, G could be a subgroup of order q in the group of modular residues $Z_p{}^*$ s.t. $p-1$ divides q, $\mid p \mid = 1024$ and $\mid q \mid = 160$, or it can be a group of points on an elliptic curve with order q for $\mid q \mid = 160$.

The algorithm 5.1 on the following page is the protocol used to coordinate the attack among participants. It achieves group key agreement, time synchronisation, and leader election.

Each VM randomly picks an r value $\in Z_q$ and then calculates and broadcasts $z = g^r$. The z values will play the identifier role for each participant.

After that, a list called $ActiveList_1$ is created containing all of the broadcasted z values. The number of items in this list is n, which represents the *alive* participants at this point in time. If there are enough participants to attack with ($n >= T_1$), we continue with the protocol, otherwise we terminate. Then, each VM calculates its

Algorithm 5.1: CIDoS attack coordination protocol.

1: Each VM$_i, i = 1, ..., n$, calculates T_1.
2: Each VM$_i, i = 1, ..., n$, selects $r_i \in Z_q$, then computes and broadcasts $z_i = g^{r_i}$.
3: Each VM$_i, i = 1, ..., n$, collects z_i values and creates $ActiveList_1 = z_1, ..., z_n$.
4: **if** $n >= T_1$ **then**
5: Each VM$_i, i = 1, ..., n$, broadcasts $X_{[k,i,i']} = (z_i/z_k)^{r_i}$ for T nearest neighbours to the right and T nearest neighbours to the left. $k \in ActiveList_1$.
6: Each VM$_i, i = 1, ..., n$, collects X values and creates $ActiveList_2 = 1, ..., m$ and sorts them ascending. The leader L is z_1 (the smallest value in the list).
7: **if** $m >= T_1$ **then**
8: Each VM$_i, i = 1, ..., m$, computes sk $sk = (z_{i-1})^{m.r_i}.{X_i}^{m-1}.{X_{i+1}}^{m-2}$ ... X_{i-2} where $X_i = X_{[i-1,i,i']}$ and $X_{[i,i',k]} = {X_{[k,i,i']}}^{-1}$. The key will be $sk = g^{r_1r_2+r_2r_3+r_3r_4+...+r_mr_1}$
9: **else**
10: Terminate.
11: **end if**
12: **else**
13: Terminate.
14: **end if**

X value, and the X values of the T neighbours to the left and T neighbours to the right. The T value here is the number of failure VMs the protocol can tolerate, and it depends on how much risk of failing the attacker is willing to accept.

Next, another active list called $ActiveList_2$ is created with m ordered items (i.e. order z ascending values). The m value represents the number of still alive participants, and if ($m >= T_1$), continue with the protocol, otherwise terminate. All the participants have the same $ActiveList_2$ with the same order; therefore, the leader can be agreed upon as the VM with the smallest value in the list. If, for any reason, the leader stops responding within a specific time frame, the second smallest value in the list will take over the leadership, and then the third smallest depending on the number of lost participants the attack can tolerate.

The final stage of the key agreement protocol is calculating the key with the formula shown in the algorithm, and as Jarecki *et al*. stated "the cycle through the alive nodes can be constructed either from a true Hamiltonian cycle or from a Hamiltonian path taken twice" [86].

The leader has the following responsibilities:

- The leader has to predict the workload pattern of the victim host, where this prediction is for a short period of time, i.e. a few milliseconds, and then creates the value ${W_T}^P$, which contains the digital representation of the target workload pattern. To convert the analogue workload to digital, pulse-code modulation (PCM) (or any other compressed encoding scheme) can be used.

  ${W_T}^P = \{attack\_d, bp\_sample, interval\_s, freq\}$.

  Where $attack\_d$ = attack duration, bp_sample = bits per sample rate, $interval\_s$ = sampling interval, $freq$ = frequency.

  We decided to ignore weak amplitude signals, which require more work to break the threshold and only concentrate on medium to strong signals. For example, if VMs (in a particular moment) consume only 15% of the CPU utilisation and the required threshold is on 80%, the attackers have to produce

65% of workload to break the threshold, which might be expensive, especially if they have limited resources to attack with. Therefore, we ignored those weak amplitude signals. A threshold can be used to specify if an amplitude is strong enough to be considered or not. This is also important to give the participants enough scheduling credits to contribute to the attack at the right time (not making too much work and so not consuming all of their scheduling credits).[8]

- The leader has to calculate the following thresholds:
  - T_s is the severity-threshold, which is a safe threshold representing the value just over the maximum the host can tolerate.
  - T_1 is a safe threshold representing the number of co-resident VMs needed to start the attack. A higher factor of overload is better for the attacker in case some unexpected failure arises such as a failure to communicate part of the protocol or a force for migration for some of the attack members just before starting the attack. The value of T_1 can be calculated by performing tests to measure the capabilities of the host and then adding a constant number derived from the test results to make the threshold safe.
  - T_2 is another threshold, similar to T_1 but with a larger value, which helps to decide if the number of co-resident VMs is much higher than needed or not. If $m >= T_2$, this means we have a relatively large number of participants, so go to *scenario 1* of the CIDoS attack. If $m < T_2$, it means that we have just enough participants and we have to design the attack neatly, which is *scenario 2*. The scenarios are shown in section 5.3.1.3.

5.3.1.3 Attack Synchronisation

After agreeing on a session key, electing a leader, determining the number of alive participants (m value), and a list of all z values $Activelist_2 = \{1, ..., m\}$, the next steps are synchronising the time, preparing, and distributing the attack pattern among participants.

Time synchronisation is important to ensure that all participants react at the right time; especially because timekeeping is still an issue in virtual environments. To synchronise the time, the one-way delay metric packet delay variation (PDV) is used [9][48]. Before broadcasting the attack pattern message, the leader should divide it into packets, add an error correction mechanism (i.e. Reed Solomon), and

[8] *Scheduling credits* is a term used in virtualisation environments to describe the amount of CPU time given by the hypervisor to each VM before switching the context.

[9] The sender broadcasts a packet, i.e. every 20 ms (or attaches the sending time to each packet). Then, when the receiver reads the packet, he or she will calculate the delay time (or early arrival) of each packet to estimate the overall delay. Thus, the attack time can be synchronised depending on the calculated value. We chose PDV to synchronise the time for two reasons. First, we avoided any synchronisation method that relies on communication through the network to avoid detection by network-based security systems. Second, in PDV, only a small amount of data (if any) has to be attached to each packet, which reduces the amount of broadcasted data. The sender needs to attach either the sending time or nothing if there is a previous agreement between participants about the time period[48].

attach a heartbeat to it. Thus, the number of heartbeats will be equal to the number of packets used to distribute the attack pattern. If the number of packets is very small, not enough to synchronise the time, the leader can send extra heartbeats. The receivers read these heartbeats and use statistics to compute the variation of the absolute (PDV) values to evaluate the delay in the covert channel. The time to attack is calculated by each participant from the last heartbeat, taking into consideration the packet delay variation.

To prepare and distribute the attack pattern among participants, two scenarios are suggested.

Scenario 1 - attack pattern random sampling

In this scenario, the number of participating malicious VMs is relatively high ($m >= T_2$). For that reason, there is no scarcity of co-resident malicious VMs; therefore, there is no need for a neat distribution of the workload among attackers. Attackers will benefit from an abundance to make it a more robust attack. Each participant decides locally where he/she wants to create a peak resource consumption. This decision can be made knowingly based on the workload history of each participant. Each participant should contribute to a pattern that is close to its previous workload to not raise suspicion; however, the peaks (time and frequency) each participant makes should be around the centre of each interval to build over the peak of the existing workload, plus the workloads of the other malicious participants.

The leader in this scenario has to broadcast the following message:

$\{z_l, {W_T}^P, T_s, attacking\_time\}_{sk}$,

z_l is the z value of the leader, ${W_T}^P$ is the predicted workload pattern of the victim, and $attacking\_time$ is the time units the receiver should wait before start attacking.

Scenario 2 - spread-spectrum attack distribution

In this scenario, the leader distributes the attack pattern among participants using spread spectrum techniques[10]. A neat distribution is required because just enough VMs exist ($T_1 =< m < T_2$). All the participants should be involved in building over the base predicted signal with overlapping and by following different patterns to avoid detection. The leader will start by predicting the workload pattern of the victim host (short period prediction), followed by informing each participant of *how much* work he/she should do and *when*. The leader can use a direct sequence spread spectrum modulation technique [153], dividing up the attack signals into pseudo-random sequences that are distributed among participants. The leader will also distribute the required amplitude from each participant, in addition to the sampling interval.

The highlighted area in figure 5.1 represents the difference between the predicted workload and the required attack pattern (to break the severity-threshold

[10]Spread spectrum distribution techniques are usually used in radio systems to help protect signals from jamming, interference, and interception; these also help reduce spectral congestion by improving the efficiency of the spectrum utilisation. When using spread spectrum techniques, a signal is spread over a wider bandwidth [2]. The signal is diffused over a wider bandwidth by injecting the corresponding spread-spectrum code or modulation code (a higher frequency signal in terms of the radio frequency field) [2]. We chose the spread spectrum technique because it provides some form of privacy and security; without knowing the corresponding spread-spectrum code (modulation code), the receiver (or the eavesdropper) cannot read the signals, which will look like random noise in the communication channel. In addition, any inference will be rejected by the demodulation mechanism.

T_s). The attacker should use the expected workload and build over it. The highlighted area should be distributed among participants using the spread spectrum. It will be divided into units and distributed *randomly* between them. The random distribution is essential to avoid having correlation between the malicious VMs workload patterns and thus avoid detection.

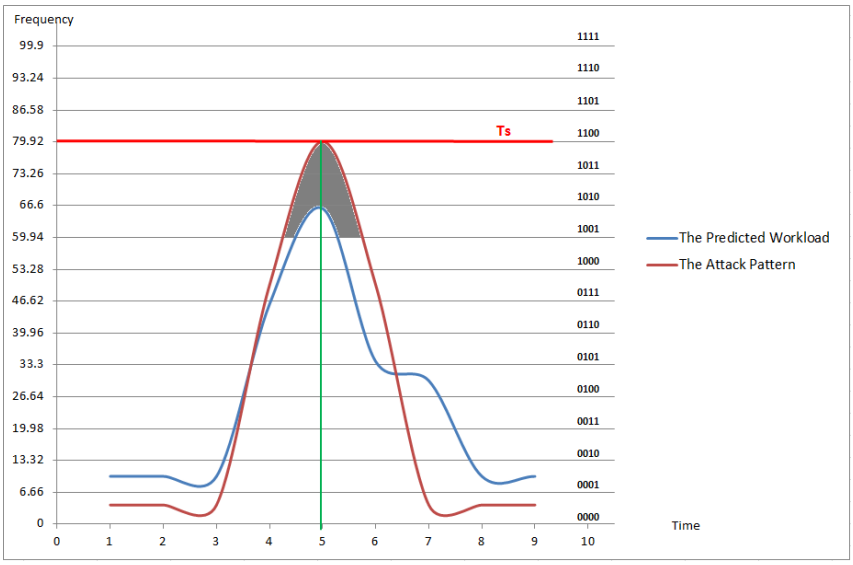

Figure 5.1: Building the attack on only one interval as an example - this figure is *illustrative*.

The leader will broadcast all of the units except the ones addressed to itself. The broadcast message has the following format:

$\{z_l, attack\_sig, ampl\_sig, interval\_s, attacking\_time\}_{sk}$,

z_l is the leader's z value, $attack\_sig$ is the attack signals into pseudo-random sequences for each participant, $ampl\_sig$ is the desired amplitude for each participant, $interval\_s$ is the sampling interval, and $attacking\_time$ is the attacking time from the last heartbeat.

Figure 5.2 illustrates the spread spectrum generation process.

The degree of details and size of intervals are calculated depending on the covert channel capacity. If the channel is wide, the leader can send very detailed attack patterns. While if the channel is very tight, the leader can choose to convert the attack to a *brute force* one by sending only an *attack-now* signal without any details. Furthermore, to design a more accurate threshold, we use environment parameters revealed by reverse engineer cloud migration algorithms, which are discussed in detail in section 5.5. The VM life expectancy can also be calculated and used to design accurate thresholds. If VMs are known to have long life expectancy (without migration) we can choose less safe thresholds.

After terminating the protocol because of an inadequate number of participants, each participant can save a buffer (or maintain a rough counter) of the number of co-resident VMs; if the counter is too low (too much lower than T_1), newcomer VMs should not be allowed to run the protocol. This is important to avoid restarting the protocol with each new arrival, even if the attack formation is just at its beginning

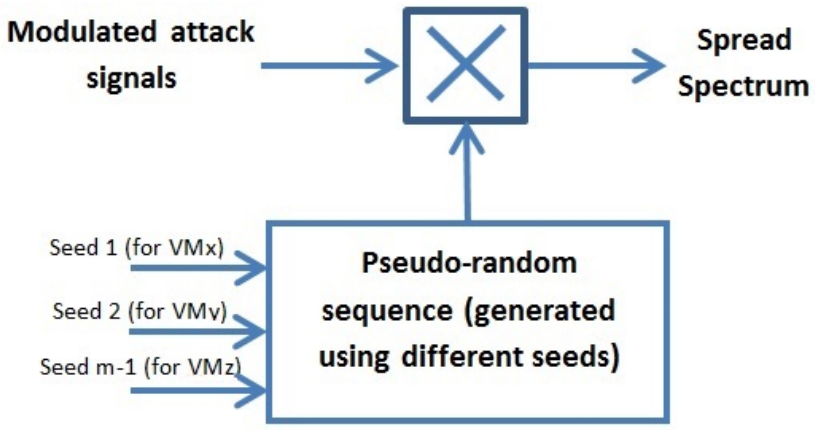

Figure 5.2: Spread spectrum generation process for scenario 2 - this figure is *illustrative*.

(only one or two VMs co-exist).

Before broadcasting, messages should be compressed for two reasons: first to minimise the use of the covert channel, which is one of the requirements, and second, to make the stream of data look random by adding an extra encoding layer. Thus, in total, there will be three layers of encoding: spread spectrum decomposition, a compressed encoding scheme (differential PCM), and encryption.

5.3.1.4 Evaluation

The protocol can tolerate up to T failures. The value of T can be calculated depending on the average life expectancy of the VMs. The computational complexity for the group key agreement protocol is $O(n + T)$.

In the protocol, three messages need to be broadcast: the z values (message 1), X values (message 2), and heartbeats + attack patterns (message 3). Messages one and two are part of the group key agreement protocol, while message three needs more examination to find the overall time resolution needed for the attack. The size of message three relies heavily on the used covert channel, because the leader decides the amount of detail depending on the channel capacity. However, there is a need to approximate this value, and we will do this in an example (the example is for scenario 2 because it is the most demanding one).

For instance, using the following values: $m = 8$ (number of participants), $bp\_sample = 4$ (number of bits needed to represent each sample), $num\_levels = 2^4 = 16$ (number of levels for the amplitude), $interval\_s, = 5$ (sampling interval) represented using 4 bits, $attack\_d = 20$ (the attack duration) represented using 4 bits, the channel capacity is 100 bps, and the packet size is 32 bits (16 used for data and the rest for error correction and preamble).

The size of message three is approximately 56 bits, which can be broadcast in less than 2 s (after encryption, compression, and adding the error correction technique), and the complexity here is linear.

5.3.2 Discussion

The complexity of the protocol can be reduced more by first studying the workload nature of the host to find the most effective prediction algorithm to compress the workload patterns. Then, reducing the T value if it is known to the attacker that the VMs in this host have a relatively high life expectancy, i.e. they can choose less safe thresholds.

The thresholds in the attack should be designed carefully to avoid starting an unformed attack, which increases the possibility of detection and might lead to investigation, blocking the attackers' VMs, and/or adding the names of the correspondent clients to a black list. Furthermore, the random distribution of the attack signals reduces the correlation between participants' workload patterns for the attack, which is important to fool intrusion detection systems.

Figure 5.3 shows the workloads of the host and attackers during the attack.

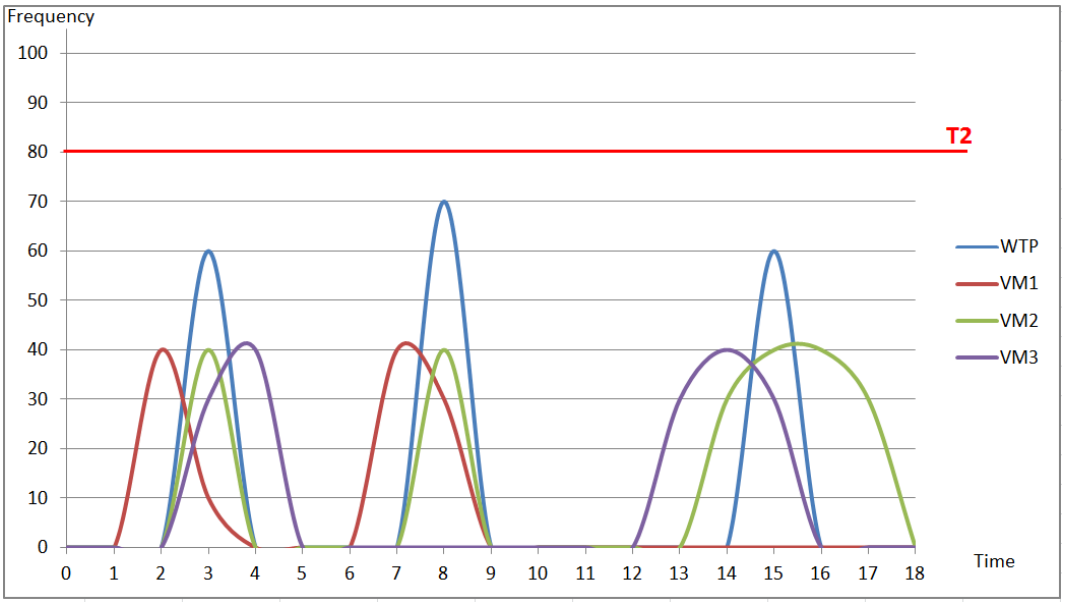

Figure 5.3: Scenario 1, attack pattern random sampling (for only 3 virtual machines) - this figure is *illustrative*.

5.4 Mitigation of CIDoS Attacks

This section discusses the detection mechanisms for different CIDoS attack scenarios. The suggested mechanisms are based on correlation measurements and the calculated distances between the attackers' workload patterns. The aim is to detect the CIDoS attack as early as possible, before the attack has been completely formed.

The detection process is based on monitoring the workload pattern of each VM, dealing with the workloads patterns as signals. We monitor a group of signals and calculate the correlation between them. For simplicity, only the CPU utilisation is monitored; however, the same idea can be generalised, and other resources such as memory utilisation, network bandwidth, and I/O can be considered. To calcu-

late the correlation, we use the simple one dimensional discrete cosine transform (DCT) and Euclidean distance. We also discuss variant prevention and response techniques.

5.4.1 Mitigation Literature

The first obvious solution to prevent the CIDoS class of attacks and other cloud attacks is blocking all the covert channels. In general, it is essential to block covert channels when discovered. However, there will exist under the ground fact of sharing resources and the insufficiency of isolation in the available hypervisors, as stated by Bates *et al*. in [19]: "we conclude that closing the employed covert channel is difficult without costly dedicated hardware or reduced network performance." It is hard to guarantee that all of the covert channels are blocked.

In [112], researchers developed a solution for the threat of covert channels in cloud computing. A new secure VM was created to mimic the vulnerable VM. The secure VM receives a copy of all of the traffic destined to the vulnerable VM and measures the difference between the outbound traffic of the two VMs. However, we believe that this solution is hard to deploy in a large-scale public cloud service, where there is a large number of portable VMs. In an experiment, Anyi *et al*. system, in [112], increased the inter-packet delay by approximately 0.05 ms. They also increased the CPU usage by about 35%, which is unacceptable in tense environments such as the public cloud.

Enhancing the isolation between co-resident VMs will definitely improve the security of the cloud and make attackers' tasks harder, and there has been some research in this area such as in [136]. However, there is still insufficient isolation [19].

The second possible solution is preventing VMs belonging to the same client (which is a possible attacker) from being co-resident. The vast majority of available cloud attacks are based on achieving co-residency between attackers or between the attacker and the victim. As mentioned earlier, in section 3.1.10, some cloud providers offer the choice of co-residency to their consumers to reduce the communication time between their VMs. This is especially essential in time critical businesses, i.e. financial business. Furthermore, co-residency could reduce the network congestion caused by VM communications. However, the provider will be aware of consumers with the co-residency choice *on*; they pay extra money for the co-residency service. Therefore, a provider can allow only them to be co-resident while distributing other VMs among hosts. This solution is possible if the VMs are initiated by the same account; however, if attackers use more than one account, finding the true identity of the client is a complicated task.

Furthermore, if the attack is spread using inter-VM attack, distributing the VMs belonging to the same account to different hosts will spread the attack more rather than quarantine it.

Zhang *et al*., in [185], suggested preventing cloud attacks by avoiding co-residency with attackers using a *'moving target defence'*. They also used Shamir's secret sharing approach to protect secrets, i.e. encryption keys. In Shamir's secret, the key is split into k pieces and spread among k VMs. The attacker cannot retrieve the key without having all of the k pieces. Researchers also encourage migrating VMs with sensitive data *regularly* to prevent co-residency. One of the main

contributions of Zhang *et al*. was the development of migration strategy based on methods from game theories; the Vickrey-Clarke-Groves mechanism was used. For simplicity, they assumed that the migration process is secure and has a fixed constant cost.

Large-scale public cloud computing is a very intense environment that hosts a large number of VMs simultaneously. If the VMs migrate regularly and last only a short period of time in each host, the cloud resources will be saturated, i.e. the bandwidth, because migration has a high cost. On the other hand, if these VMs last for a long period of time in one host, there might be enough time for the co-residency to be accomplished and the attack to be formed. To improve security, Zhang *et al*. suggested that VMs have to last the *right* period of time in each host and this *right* period of time varies depending on the attack. Deciding the migration time based on available attacks mechanisms requires prior knowledge of these mechanisms (a technique similar to the one used in signature-based HIDS). Another point is that the assumption of secure migration is unrealistic. Migration increases the cost, attack surface, and the potential threat thus should be used with caution.

Researchers in [75] recommended numerous defence mechanisms against a variety of cloud attacks such as blocking internal communication between hosts and using a separate network adapter for each VM. These solutions might be effective in other cloud attacks but not for the CIDoS class of attacks because the communication among VMs is through an unauthorised medium (a covert channel), which is hard to block. Using separate network adapters will prevent the attack if the used covert channel is based on network timing, but if it is a memory bus covert channel (which suits the cloud most), having separate network adapters will not make any difference.

In [70] numerous incident handling practices for cloud computing attacks were discussed, i.e. cut the network connection to the compromise VM, use honeywall to observe suspicious traffic, and add more resources to mitigate DoS attacks. Furthermore, preventing cloud cartography and preventing attackers from mapping the cloud can also stop a number of cloud attacks, as suggested in [139]. However, little research has been conducted in this area, and we believe that it needs more investigation by cloud security researchers.

In this research, we developed detection methods designed specifically for the CIDoS class of attacks. To detect the attack, we monitored the workload patterns of the attackers. We dealt with VM workload patterns as signals and processed them as signals. Many methods have been used to compare signals. For instance, in [127], different methods to calculate the distance between signals were tested. Any of them can perform the task, and we chose the *Euclidean distance with thresholds* method because of its simplicity and efficiency. Thresholds are used to reduce the amount of processed data and therefore the cost. We also adopt other techniques from the signal processing field to detect the most advanced scenario for the attack (scenario 1). The chosen technique should be able to transfer the time domain to the frequency to calculate the correlation between signals. Many methods can be used to accomplish this task, such as the discrete cosine transform (DCT) and wavelet transform. We used the DCT, and only a one dimensional DCT function was needed [103].

5.4.2 Mitigation Strategies

5.4.2.1 Prevention

To prevent a CIDoS attack, cloud providers can disable both the over-commitment and migration features, and allocate requested resources to each VM from the beginning, even if they are not being used. This solution is valid in an uncrowded cloud. However, the main idea of cloud computing is sharing to reduce cost by increasing the utilisation of servers to the maximum. Thus, over-commitment and migration are essential features to accomplish these goals. They are also correlated to each other; you cannot have over-commitment without migration because if one of the tenants in an over-committed host has requested more resources (the elasticity feature), the provider might need to migrate this tenant to another less pressured physical host.

The second solution is preventing co-residency using a placement algorithm that distributes the VMs belonging to the same client to different hosts.[11] This option is hard to maintain over time because VMs consistently move from host to host for different reasons, such as to save energy, reduce the stress of an over-utilised host, conduct maintenance, or after the VM has been restarted.

Some providers place all the VMs that belong to a client in the same host to reduce the cost of communication between them. Moreover, some consumers require co-residency, especially in businesses where millisecond delays in communication make a difference. Offering a co-residency option to consumers increases the possibility of a CIDoS attack because it performs the co-residency and co-residency check steps on behalf of the attackers.

The last suggested prevention solution is creating a black list to block suspicious accounts and use more reliable ways to identify the true identities of these accounts using techniques similar to the ones used in online payment security.

5.4.2.2 Detection

The applied detection methods have the following goals:

- detect the attack as early as possible before it is completely formed,
- distinguish between legitimate and malicious changes in workload,
- generate a sufficiently strong anomaly signal, and
- maintain a low detection cost in terms of time and complexity to be able to work on the fly.

The main novel mechanism developed in this research, to detect the attack as early as possible, is based on workload signals correlation. Different parameters can be used to measure the workload. The parameter we choose here is CPU utilisation (for simplicity). However, combining other parameters might improve the detection. In reality, the choice of parameters depends heavily on the attack mechanism. If the attacker stresses the CPU, then obviously monitoring the CPU utilisation will detect the attack, but with different circumstances, other parameters should also be considered.

[11] This strategy is used by Amazon EC2.

Although there is a leader in this attack, finding the leader is not essential. Any participating VM can be considered as the leader, because the correlation will be among all of the participants.

Data collection:

We collect the workload signals of co-resident VMs; these VMs might be a collection of normal and malicious VMs. In a large-scale public cloud, it is highly probable that, first, each of these VMs belongs to a different client and second, there is no communication between them internally. For these reasons, each VM is expected to generate a unique workload pattern that is uncorrelated to the workloads of other VMs. However, in environments where co-residency is a choice of the client, a correlation between the signals of related VMs might exist. This correlation depends on the type of VMs and the services they provide. However, we assume here that the cloud provider will be aware of this correlation (i.e. from the contract) and the nature of these VMs is also known to the provider.

After collecting workload signals, the next step is processing them to reduce noise and increase accuracy. We only considered data that might indicate the attack to improve the performance and reduce the cost. We only considered strong workload signals (high amplitude). Signals below a specific threshold were ignored and replaced by zeroes. This threshold is a fixed arbitrary threshold that can be chosen by experiment.

To verify the validity of the suggested detection techniques, we created synthetic workload signals based on our understanding of the attack and the following assumptions:

- the workload pattern of each VM differs randomly from other VMs unless abnormal, and
- each VM has a stable workload pattern that is not random, because in the real-world, servers usually have stable workload patterns (they perform the same series of tasks repeatedly) and this is the main idea behind regular anomaly-based HIDS.

We used a random factor to build each VM workload pattern based on the data for scenarios 1 and 2. We also used another two random factors to choose where to make the peaks (around the centre) and the strength (amplitude) of these peaks. These random factors were only applied to the scenario 1 data.

Only short samples were created because the attack is designed to be fast and target only the next short interval of time to avoid VMs migration and a host workload pattern change.

We have two noise sources to deal with; the first is faulty malicious VMs (outside the leader controlled suspicious VMs), which might cause a false negative alarm. Faulty malicious VMs are not a problem because if the number of participating VMs is high (scenario 1), the attack can still be formed and detected without the faulty VMs. If the number of faulty VMs is high too, in this case the attack will fail (as accurate timing is essential for this attack to succeed); therefore, this also not a problem.

The second noise source is innocent VMs that have weird behaviours. In reality, these VMs should be detected and diagnosed using a regular host-based IDS; furthermore, they are not harming our results because it is highly unlikely that

their random behaviours will match the leader's plan and generate a false positive alarm.

We used a sliding window to accelerate the detection and accuracy, as in [151].

The required number of VMs to build the attack is m. In the real-world, the m value is based on the physical host server specifications. The stronger the server, the more VMs are required to perform the attack. For our experiment, we chose $m = 6$.

Detecting scenario 1 - pattern random sampling:

This scenario is hard to detect because a large number of VMs are participating in the attack (far more than required). Therefore, they can avoid changing the behaviour suddenly, which might be detected by regular behaviour monitoring tools, i.e. HIDS. Each participating VM can change its workload gradually to reach the targeted workload pattern, while maintaining a harmony with its old workload pattern. In this scenario, VMs also can afford a degree of randomness by deciding *where* and *how much* their peaks are, but they should be around the centre of the peaks in the distributed plan. Since all the VMs are following the same plan, they change their workload patterns to be close to each other. Therefore, we assume that there will be a correlation between their workload patterns, and that this correlation will increase over time until the attack has been completely formed, and the sum of the workloads break the severity-threshold T_s. The detection mechanism is based on this correlation.

For the early detection of the attack, a signal processing technique is used to map the time domain to the frequency domain and then calculate the correlations (data transformation for statistical analysis). Any suitable Fourier transform technique can be used; we use the one-dimensional discrete cosine transform (DCT) because it is simple to apply and has low complexity (N^2) multiplications [23]. DCT calculates the sum of the cosine function for sequential data points with different frequencies; in comparison to the sine function, the use of the cosine function is more efficient for performing the required tasks because fewer data points are needed to represent the signal [114]. We applied the DCT, and then used hypothesis testing based on a threshold to decide whether or not VMs were acting suspiciously. High correlation coefficients between VM workloads that increase over time are a strong sign of a CIDoS attack.

In our experiment, we compared the correlation coefficients between the signals of six malicious VMs with two innocent normally behaved ones. Thus, we had eight samples in total, S = 8, S_1, S_2, S_3, S_4, S_5, and S_6 (the malicious signals) and S_7 and S_8 (the innocent signals).

The number of points extracted from each signal was n. In our experiment, we chose $n = 300$ (the n value is chosen based on the length of the signal and the acceleration of the frequencies in a way that allows us to cache the characteristics of this signal).

We ignored all the relatively weak amplitudes (under a specific threshold) because they have no meaning in our attack. We only considered the strong amplitudes and it was not important exactly how strong they were because the strength of each VM was decided locally and was random. Therefore, we represented the amplitudes over a specific threshold as *1s* and below the threshold as *0s*. The choice of the threshold was made based on the required balance between false positive and

false negative errors. We called this threshold ts_1.

We created a relatively short signal for each VM following the previous assumptions and then represented it using n points. The threshold ts_1 was used to decide which amplitude should we considered, and it was equal to 50% of the maximum available strength in our experiment. Then, we applied the DCT to each sample to linearly transform the signals into the frequency domain. The DCT expressed each signal in terms of a sum of cosine functions with different frequencies. The function of the one-dimensional DCT coefficients is:

$$w_k = \sum_{t=0}^{n-1} s_t cos\left[\frac{\pi}{n}(t+\frac{1}{2})\right], k = 0, ..., n-1 \text{ [103]}$$

$$c_k = \begin{cases} \frac{1}{\sqrt{2}} & k = 0 \\ 1 & k > 0 \end{cases}, k = 0, ..., n-1 \text{ [103]}$$

$$\textit{DCT coefficients} = \sqrt{\frac{2}{n}} * c_k * w_k \text{ [103]}$$

A set of coefficients were calculated from each sample, and then the sum of these coefficients was calculated and used to represent its sample. It was also used to measure the correlation between VM workloads; high correlation indicated the suspicion of a CIDoS attack. The generated abnormality signal had to be strong enough, and we used hypothesis testing to determine whether or not the signal was strong enough. If the difference between the sum of coefficients of a sample and the sum of coefficients of another sample was less than a specific threshold, and it decreased over time, an alarm was raised for a CIDoS attack to detect the attack as early as possible.

In our experiment, the final sums of the coefficients for S were as follows: S_1 = 8.04, S_2 = 7.74, S_3 = 8.07, S_4 = 8.42, S_5 = 7.05, S_6 = 8.62, S_7 = 12.08, and S_8 = 10.94.

Hypothesis testing:

The problem of deciding whether the generated anomaly signal was strong enough or not can be formulated in terms of hypothesis testing as follows:

1. The null hypothesis: $H_0 : Q_1 - Q_2 > 1$

   $H_1 : Q_1 - Q_2 <= 1$ (one sided hypothesis)

   Q_1 is difference in coefficients for normal samples and Q_2 is difference in coefficients for malicious samples.

2. Assume H_0 is true

3. Difference in coefficients for VMs workloads follows approximately normal distribution.

4. Level of significance $\alpha = 0.001$ $(99.9\%$ confidence level)

5. Find Z scores: $Z_\alpha = Z_{0.001} = -3.09$

6. Find the region of rejection RR which is a set of values less than or equal to $\alpha : (RR <= -3.09)$

7. Collect samples

Table 5.2: Statistics calculated from the samples.

Normal	Malicious
N_1 = 2	N_2 = 6
μ_1 = 3.477384	μ_2 = 0.455994

8. Extract difference in coefficients for signals and calculate some statistics shown in table 5.2.

   $SD = \sqrt{\frac{\sigma_1^2}{N_1} + \frac{\sigma_2^2}{N_2}} = 0.590617176$

   $\Delta = 1$ (the value in the null hypothesis)

   $\mu = \mu_2 - \mu_1 = -3.02139$ the new centre

   $ME = Z_\alpha * SD = Z_{0.001} * SD = -1.825007075$

   Range: -4.846397075 to -1.196382925

   $Z = \frac{\mu - \Delta}{SD} = -6.808792837$

9. Draw a conclusion: the test statistics Z = -6.808792837 is not in the RR so we retain the null hypothesis.

In the previous testing, we only covered measuring the coefficients at a particular point of time. However, we can use a technique similar to a sliding window and continuously or periodically measure the coefficients. Then, if the differences decrease over time, this is another sign of abnormality and a strong indicator that these VMs are following a plan and not behaving innocently. We cover the idea of detecting a correlation that increases over time in the detection method used for scenario 2.

Detecting scenario 2 - spread spectrum attack distribution:

In this scenario, malicious VMs adjust their workload to reach the required amount of workload at the same moment to stress the resource and trigger migration. To avoid detection, VMs try to approach each other gradually by persistently adjusting their workload signals to match the detailed plan that is designed and distributed by the leader. To detect the attack, we calculated the Euclidean distance between the normalised signals of all the VMs and the leader [127]. We found that the calculated distances varied enough to distinguish between normal and malicious VMs. The distances between malicious signals were low (in comparison with normal signals), and they decreased over time. Other techniques can be used to measure signal correlation, but the chosen technique should have low complexity and be able to perform on the fly.

If the distance rate between the leader's workload signal and those of the other VMs decreases over time for a number of VMs simultaneously, an alert should be raised. Two thresholds are needed, TD_2 and TD_3. TD_2 is the threshold to decide if the distance between the leader and other VMs is worrying or not; if the distance is below TD_2, a suspicious case should be registered and monitoring should continue. TD_3 is the threshold to decide the maximum acceptable number of suspicious cases. If the number of cases is over TD_3, this means there is a relatively large number of VMs that are trying to adjust their behaviour to match each other.

Therefore, there is a suspicion of an attack, and an alert should be raised. We wrote an algorithm to apply the previous steps and detect the attack, see algorithm 5.2.

ts_1 is the threshold used to decide which amplitude we are considering (we ignored weak amplitudes).

Algorithm 5.2: Early Detection Algorithm for CIDoS Attack, Scenario 2.

Require: A set S of m signals and each signal contain n items, S = $\{\{x_{00}, x_{01}, , x_{0n}\}, \{x_{10}, x_{11}, , x_{1n}\}, , \{x_{n0}, x_{m1}, , x_{mn}\}\}$ and thresholds ts_1, TD_2, and TD_3

for each x_{ij}, 0 ≤ i ≤ m **do**
  for each 0 ≤ j ≤ n **do**
    if $x_{ij} < ts_1$ **then**
      x_{ij} = 0
    end if
  end for
  Calculate μ and σ of x_i
  Normalise x_i to $\widehat{x_i}$
end for
Chose one of the VMs to be considered as the leader $\widehat{x_k}$
for each $\widehat{x_{ij}}$ where 0 ≤ i ≤ m **do**
  for each $\widehat{x_{ij}}$ where 0 ≤ j ≤ n **do**
    $\widehat{x_{ij}}$ = $\widehat{x_{ij}}$ - $\widehat{x_{kj}}$
    Distance D = D + $\|\widehat{x_{ij}}\|$
  end for
  if D < TD_2 **then**
    Count C = C + 1
  end if
  D = 0
end for
if C > TD_3 **then**
  Alert
end if

The values of TD_2 and TD_3 can be specified depending on the environment parameters such as the capacity of the host and VM size.

Brute Force scenario:

A sudden simultaneous increase in the consumption of VMs could indicate an attack, but it may also be a simple normal peak in activities. It is therefore of great importance to know the cause of the increase, which could be obtained from different sources such as the VM owners (the clients) and/or information obtained from the VMs' old behaviour (history). Monitoring the VMs' behaviour before and after the increase helps to distinguish between normal and malicious sudden increase in consumption. Typically in the CIDoS class of attacks, malicious VMs maintain a very low consumption rate to provide space for newcomers and thus achieve co-residency. Their consumption does not increase until there is a sufficient number of malicious VMs to trigger the attack. The sudden increase in consumption can be detected using a simple statistical analysis.

Results and discussion for detection strategies:

In both scenarios, the detection is based on the same idea, which is measuring the correlation between signals. If the correlation is higher than normal and it keeps increasing over time, this is a strong indication of a CIDoS attack. In our experiment, we found that the distances between malicious signals are low in comparison with the distance between normal ones. Some normal VMs might coincidentally have high correlation with malicious VMs in some points. However, the correlation between malicious VMs has to *decrease* over time because they are coor-

dinating to match the distributed pattern, and normal VMs will not follow the same pattern. Therefore, it is highly unlikely that the coincident correlation between normal and malicious VMs will last for the whole test and cause a false alarm.

Ignoring weak amplitudes by representing them using zeroes and strong amplitudes using ones makes it possible to reduce the detection time and complexity of operations. In addition, the Euclidean distance and one-dimensional DCT are very simple and straightforward methods. Applying simple methods is essential to run the detection on the fly and detect the attack in time before it is completely formed.

5.4.2.3 Response

In the case of a widely spread CIDoS attack, if the provider network is seriously infected, one of the suggested responses is temporarily disabling migration. Disabling migration stops the consumption of resources and gives cloud administrators the chance to deal with the attack without a serious service disruption. It is also important to alert other hosts and/or the central distributed intrusion detection system (DDIS) to be aware of the attack, and monitor and block (if needed) suspected VMs.

We can also respond to the attack by targeting the timing. Some of the suspicions VMs can be suspended for milliseconds, which is enough to disturb the timing of the attack. However, providers have to make sure that this action does not violate the service level agreement, SLA, with the clients or affect the integrity of their data.

Another possible response action is distributing suspected VMs to different hosts, which forces a restart of the attack. The problem with this solution is that if the attack is spread through inter-VM infection, it will be distributed rather than quarantined, and this will increase the damage. Another useful technique is isolating one of the suspected VMs and monitoring its behaviour. If the behaviour has changed after isolation, or the VM has been terminated, that is another suspicious sign and the defender in this case might block all the VMs with correlated workload patterns. It is also important to block any discovered communication channels used by the attackers, i.e. covert channels.

5.5 Dynamic Parameter Reconnaissance for CIDoS Class of Attacks

In CIDoS, m co-resident VMs increase their workloads to reach thresholds (time and strength) and trigger migration. These thresholds were assumed to be known to the attackers, while in reality they are hidden and discovering them requires designing a new attack, which is described in this section.

We designed a series of attacks to reveal some of the cloud parameters that are used by migration algorithms. We also designed a formal model for the migration decision process and then created a dynamic algorithm to extract the required parameters. The mechanisms to extract these thresholds were adapted to dynamic changes in cloud algorithms. Revealing parameters is hence a security threat in itself, and they can be used by malicious VMs to accurately generate the needed

workload to continuously trigger migration, resulting in thrashing. It is vital that the generated workload is no more than required to avoid detection and make the attack live longer. Furthermore, attackers may also use the revealed parameters to avoid being migrated.

The attack proposed in this section is a stealthy randomising testing strategy to learn the thresholds that are used in the process of making migration decisions (reverse engineer migration algorithms). We performed a series of attacks (called *tests*) to reveal these thresholds; these tests were based on CIDoS, together with a statistical analysis. The attack proposed in this section is still theoretical, and we will verify it in future work.

For simplicity, we assume that cloud hosts are homogeneous (having the same specifications). However, if this is not the case, the attack still works but instead of discovering the host specifications only once for the whole cloud, attackers should perform this task once per host.

5.5.1 Migration Algorithms Literature

The success of the CIDoS class of attacks is based on the understanding of the algorithms used to manage the migration process in the cloud. One of the main challenges we faced was the fact that the migration policies and algorithms used by today's cloud providers are not publicly revealed. However, many research studies discuss these policies and algorithms.

VM migration is the process of transferring a whole VM from one host to another [16] for various reasons, which are as follows:

- to save energy and thus reduce cost by evacuating low utilised hosts and then turning them *off* or putting them in *sleep mode* [32],
- for fault tolerance when dealing with faulty or malicious VMs or for maintenance reasons, and
- to reduce the load in an over-utilised host to avoid SLA violations [32].

To allow migration, cloud providers have to deal with many challenges in security and performance. These challenges include the following:

- minimising the migration time to avoid consuming the network, which might cause a lack of response and thus an SLA violation, as discussed in [20]. Furthermore, in [168], researchers found that because of live migration (which is considered to be the default feature in today's virtualisation community, as stated in [16]), application performance was degraded by 10% (the percent varies depending on the application nature) [168], and
- protecting the VM being migrated from data loss or security breaches.

Thus, VMs migration has a high cost and should be used with caution.

Migration mechanisms:

Node management machines are responsible for managing migration, including the issuing of migration orders based on a number of parameters and policies such as utilisation and/or power-consumption policies [20], [32], and [144]. Status reports have to be collected from each host periodically to show its general status and the status of its tenants [20]. The data from these reports are the inputs for

a collection of migration algorithms. There are many different policies and algorithms used in the cloud; we will discuss some of the most popular ones. However, this is not an inclusive list.

These algorithms include the following:

- migration decision algorithms:
  - overload detection algorithm [16], [32], and [20].
  - power saving algorithm [144].
- VM selection algorithm [16] and [175].
- VM placement algorithm [20] and [54].

In [32], researchers focused on reducing the cost and carbon footprint by reducing the energy consumption. They designed an architectural framework and principals to accomplish this task. Furthermore, policies and algorithms were proposed for resources allocation that aimed at reducing the energy consumption while considering the SLA. For VM selection, the following policies were suggested: 1- migrate the minimum number of VMs, 2- migrate the least used VMs and 3- choose the VM to be migrated randomly (using a uniformly distributed random variable). A best fit decreasing algorithm with modification was used to allocate VMs with upper and lower utilisation thresholds.

In [144], Sammy *et al*. concentrated on reducing the power consumption by declaring very low-utilised servers to be *retirement servers*, which means they should be turned *off* soon to save power. Therefore, *retirement servers* should not accept any new tenants but should wait for the current tenants to shut-down or be migrated if they exceed a threshold called the *retirement threshold*. The algorithm used to accomplish this task was dynamic round-robin.

Beloglazov *et al*., in [20], proposed algorithms based on dynamic measurements generated by statistically analysing historical data. Four methods were proposed to detect overloaded hosts: the median absolute deviation (which specifies the value of upper utilisation based on the CPU utilisation deviation strength), interquartile range, local regression, and robust local regression. For VM selection, three polices were suggested: migrating VMs with the minimum migration time calculated based on the memory usage and NT bandwidth, random selection based on a uniformly distributed discrete random variable, and migrating VMs with the highest CPU utilisation correlation with other VMs calculated using multiple correlation coefficients.

For the detection of under-loaded hosts (power saving algorithms), Beloglazov *et al*. suggested finding the host with the minimum utilisation and evacuating it without over-utilising any other host in the cloud.

For the VM placement problem, researchers suggested sorting VMs based on their CPU utilisations and then allocating them to hosts that provide the minimum cost in terms of power consumption. They used the best fit decreasing algorithm with a modification.

Another study, [16], also used dynamic utilisation thresholds to detect overloaded hosts. The dynamic thresholds were calculated based on the workload history (statistical analysis). Bala *et al*. measured the statistical dispersion using

the median absolute deviation. For VM selection, the *multipath correlation coefficient* was used to describe the relationship between measurements. These measurements were grouped in different levels, where each level affected the subsequent ones. The machine with the minimum expected workload and that had the least influence on others was migrated (VMs with zero inter-correlation factors could be migrated). This policy reduced the migration time and the number of migrations needed.

Researchers in [46] proposed prediction algorithms for short term prediction to detect an overload before it occurs to avoid an SLA violation. They also proposed long term prediction to be used in managing resources in the cloud. For short term prediction *multivariate linear regression* based on the CPU utilisation was used.

Overload detection algorithm:

The upper utilisation threshold can be set to decide whether or not the host is overloaded. However, as stated in [20], "fixed utilization thresholds are not efficient for IaaS environments with mixed workloads that exhibit nonstationary resource usage patterns;" they suggested dynamic thresholds. Prediction algorithms are also needed to create these dynamic thresholds and predict an overload before it occurs [16]. Different techniques are used for prediction, such as statistical analysis or machines learning algorithms [20].

Power saving algorithms:

Power saving algorithms are responsible for detecting hosts with low utilisation. Fixed and dynamic thresholds can also be used here, as in [144] and [16].

VM selection algorithm:

Many policies are used for VM selection algorithms, such as migrating the minimum number of VMs [32], migrating the least active VMs [32], migrating VMs randomly [20] and [32], or migrating the VM with the highest correlation (maximum correlation policy) [20] and [149].[12] Discovering the policy used can help reveal the required parameters and improve the CIDoS attack, as shown in section 5.5.

Placement algorithm:

The new location can be chosen based on different factors; the most popular ones are the power consumption factor (to reduce it) [32] and utilisation factor (i.e. the minimum utilised host). Many algorithms were suggested in [20], [32], [175], and [54].

5.5.2 Estimating Cloud Migration Parameters

Attackers aim to extract some of the main parameters used by migration algorithms and use them to build more efficient, harder to detect, and more robust DoS attacks. Because of the power saving policy, the number of running hosts is dynamic and, as a consequence, the used thresholds should be dynamic, and the process of extracting parameters has to be dynamic too. Furthermore, we need to discover these parameters as fast as possible because of the dynamicity of the environment. We need to do the following:

[12]The Maximum Correlation policy based on the assumption that the higher the correlation between the resource usage by applications running on an over-utilised host related to higher probability of the host overloading [149].

1. extract the required parameters,
2. measure the reliability of the extracted parameters and reduce the probability of accidental errors,
3. consider the noise,
4. consider not changing the host behaviour heavily to avoid affecting the prediction algorithms, and
5. measure the success of the attack.

We dealt with this problem as a regular statistical experiment.

Extracting the required parameters:

This task is accomplished by reverse engineer the algorithms that are responsible for making the migration decision. We start by designing a formal model of the migration decision process.

Migration decision process:

We consider two migration policies, over-loaded host detection to prevent SLA violations and under-loaded host detection to save energy.

The host management node collects status reports periodically from all the hosts under its control. Data from these status reports and other data from the environment are the inputs of the algorithms. Then, an algorithm runs to decide whether the host under examination is currently over-loaded or predicted to be over-loaded. The output of this algorithm is *zero* (if the host is not over-loaded) or *one* (if the host is over-loaded and migration is needed).[13] If the output is *one*, the management node will run the VM selection algorithm to decide the best candidate VMs for migration, and then VM placement algorithm to choose the best candidate destination hosts for the VMs under migration. Lastly, VMs will be migrated either online or offline.

The inputs of the over-loaded host detection algorithm are as follows:

- the general overall status of all the hosts in the same availability zone as the host under examination,
- the specifications of the host under examination,
- the history of, current, and predicted CPU utilisations,
- the history of, current, and predicted memory utilisations,
- the history of, current, and predicted network traffic rates to and from the host,
- possible errors,
- time, and
- hidden unknown variables.

We target large-scale public cloud providers. Therefore, the effects of a change in the general overall status of hosts are not usually dramatic. Thus, it can be considered to be constant and represented by a constant value in the migration algorithms.[14] We assumed that the hosts are homogeneous (same specifications). Therefore, the effect of the host specifications can also be considered to be a constant

[13] The final migration decision has to be binary. However, the process for reaching this decision is based on dynamic parameters and thresholds.

[14] The effect on the migration decision when the number, i.e. 10.000 hosts *on* or 10.003 hosts *on*, is too low.

and added to the constant value. The error is reduced by replicating the test many times and then using hypothesis testing to decide whether to accept the revealed parameters or not. For simplicity, we did not consider the memory utilisation and NT traffic rate; we only considered the CPU utilisation.

There are many methods for CPU utilisation prediction, most of which are based on the history of the host. A multivariate linear regression model (MLR) can be used to perform the prediction, as in [46]. The CPU utilisation history is partitioned into intervals and analysed to measure "how closely prediction matches observed utilisation across the utilisation spectrum" [46].

The algorithm 5.3 is used for over-loaded host detection. The algorithm notations are as follows:

- $x_{history}$, $x_{current}$, and $x_{predicted}$ are the history, current, and prediction of CPU utilisation consequently,
- x_{time} is the time,
- α is the constant value,
- upperU is the dynamic upper utilisation threshold: f (α+$\beta_1 x_{current}$+$\beta_2 x_{history}$+$\beta_3 x_{time}$),
- mig is a Boolean variable which is set to 1 if a migration required, and
- if the current CPU utilisation or the predicted CPU utilisation is over the threshold upperU, migration is required. If $x_{current}$ or $x_{predicted} \geq$ upperU then mig =1.

Algorithm 5.3: Over-loaded Host Detection.

```
1: Input: α, x_time, x_history, x_current, x_predicted Output: mig
2: mig = 0
3: upperU = UpperThreshold(α, x_time, x_history, x_current)
4: if x_current or x_predicted ≥ upperU then
5:     mig = 1
6: end if
7: Return mig
```

Algorithm 5.4 on the next page is for under-loaded host detection. The algorithm notations are:

- $x_{history}$ and $x_{current}$ are the history and current of CPU utilisation consequently,
- x_{time} is the time,
- α is the constant value,
- VMsNumbers is the number of tenants in the host under examination,
- lowerU is the dynamic lower utilisation threshold: f (α+$\beta_1 x_{current}$+$\beta_2 x_{history}$+$\beta_3 x_{time}$),
- mig is a Boolean variable which is set to 1 if the host is under-loaded and all VMs in that host can be migrated without causing over-utilisation to any other host, and
- if the current CPU utilisation is lower than the threshold lowerU, the host is under-utilised and need to be evacuated (if possible) then turn it *off* to save energy.

Cloud migration parameter estimation:

Algorithm 5.4: Under-loaded Host Detection.

```
1: Input: α, x_time, x_history, x_current, VMsNumbers Output: mig
2: mig = 0
3: lowerU = lowerThreshold(α, x_time, x_history, x_current)
4: if x_current < lowerU then
5:     if hostsAvailability(VMsNumbers) then
6:         mig = 1
7:     end if
8: end if
9: Return mig
```

We developed an algorithm to extract the parameters from the migration model, see algorithm 5.5.

Algorithm 5.5: Dynamic attack permitting cloud migration hidden parameter estimation.

```
1: Input: chunksList, Bprofile_normal Output: x_current, x_time
2: for chunk in chunksList do
3:     wait()
4:     CIDoS.run(chunk.x_current, chunk.x_time)
5:     Bprofile_current = updateProfile ()
6:     SuspicionValue = compare (Bprofile_current, Bprofile_normal)
7:     if SuspicionValue > simThreshold then
8:         successCounter = 0
9:         for i = 0 → replicationNum do
10:            x_current1 = increment x_current
11:            wait()
12:            Bprofile_normal = updateProfile()
13:            CIDoS.run(chunk.x_current1, chunk.x_time)
14:            Bprofile_current = updateProfile()
15:            SuspicionValue = compare (Bprofile_current, Bprofile_normal)
16:            if SuspicionValue < simThreshold then
17:                successCounter = successCounter + 1
18:            end if
19:        end for
20:        if successCounter ≥ successThreshold then
21:            Return x_current, x_time
22:        end if
23:    end if
24: end for
```

First, the attack leader specifies the range of the test (the minimum and maximum CPU utilisation and time) that might cause migration (this is the *test range*). Then, the leader designs a series of all possible test phases, portions them into chunks, and gathers them into a list called *chunksList*. Each item in the list is called *chunk*, and it has two variables, $x_{current}$ and x_{time}. The inputs of the algorithm are *chunksList* and $Bprofile_{normal}$, which is the profile of the host's normal behaviour before the attack. The algorithm then tests the chunks in the list one by one until it finds the parameters that cause migration. *CIDoS.run* is a function with two arguments (CPU utilisation and time) to run a phase of the CIDoS attack (the whole attack that has been described in section 5.3). This function is responsible for coordinating the resource consumption of malicious VMs to stress the host and cause migration. It attacks using the time and strength passed to it in the variables $chunk.x_{current}$ and $chunk.x_{time}$.

To be able to measure the success of the attack (do the tested parameters cause migration or not?), we create normal behaviour profiles for the host workload pattern before and after the test. To create a new behaviour profile (for after the attack), we use the function *updateProfile* and calculate the distance between the old normal behaviour profile and the new one using the function *compare(profile1, profile2)*. The function *updateProfile* is a regular anomaly-based IDS used to detect anomalies in the workload pattern of the host, i.e. HMM-based IDS (more details are shown later in the section). The behaviour profile is updated using fresh data (newly collected data from the current workload). The result of the comparison between profiles is contained in the variable *SuspicionValue*. If *SuspicionValue* is greater than the threshold *simThreshold*, the host behaviour has changed (probably because of migration), so initially accept the tested parameters and replicate the test *replicationNum* number of times to increase the reliability and reduce the effect of accidental errors. *successCounter* is the number of successful replications. If it is greater than or equal to the threshold *successThreshold*, then accept the tested parameters as reliable and exit the algorithm. The function *wait()* is used to create a gap in time between test phases to avoid affecting the prediction algorithms.

If a VM has been migrated in the middle of one of the tests for another non-malicious reason, this will not affect the reliability of the test, it will be discovered in the replications. As stated, replications are used to increase the reliability and decrease the effect of accidental errors. The attacker needs m malicious VMs to attack with, where the value of m can be calculated based on the host specifications. With a stronger host (high specifications), more malicious VMs are needed to attack.

The time complexity of the algorithm depends on how many tests are needed, the size of the gap between tests, and how many replicates should we make to have acceptable reliability. A smaller *test range* means fewer tests are needed. Furthermore, if we distribute the tests between different hosts, we can perform them in parallel, and the gap in time will be less, or there will be no need for a gap. However, the communication between attackers through the network increases the possibility of being caught by a NIDS.

How to check migration:

The leader checks for migration (measures the success of the attack) by building a normal behaviour profile for the host workload and re-examines the workload after each phase of the test. If there is a deviation in the workload (anomaly detected in terms of the HIDS), it is highly probable that a migration has occurred. If there is no deviation (no anomaly detected), the current test phase has failed, and another phase should be performed using different $x_{current}$ or x_{time} values.

How to build normal profiles:

We assumed in the threat model that the attack leader can monitor the host workload [19] and [139], and can create a normal behaviour profile for the host (host-based anomaly detection system). The host-based anomaly detection system detects any change in the host's behaviour.[15] We first have to build a detection system; different algorithms can be used to build the system, some of which are based on machine learning techniques such as the hidden Markov model and others are based on statistical learning techniques such as regression [41]. These algorithms

[15]The degree of sensitivity of the anomaly detection system (detection rate) depends on the threshold values (how much change in behaviour is accepted).

are used to model the workload of the host (create the normal behaviour profile of the host workload). To obtain the required data for building the model, the attack leader can gather observations from the host, for example by calculating the host's response time or using a side channel to collect data. Then, the host's normal behaviour profile can be used to detect any deviations from the normal workload pattern.

The attack leader calculates the *SuspicionValue* by comparing the registered normal profile to the current profile. To compare the two profiles, we calculate the distance between them using techniques such as the Kullback Leibler distance metric. If that value of *SuspicionValue* is high, an alert for an anomaly should be raised (possible migration). To decide whether or not *SuspicionValue* is high, the leader has to specify another threshold, *simThreshold*. This threshold is calculated based on the available resources for the attack and the degree of assurance needed by the attackers.

If there is an anomaly (the host workload pattern has changed significantly), which might be an indication of a migration; and this is how we check for migration. We also have to consider the noise; a migration might occur for another normal reason (not because of an attack). Furthermore, the workload might change for another reason, rather than migration, which might affect the reliability of the test and the revealed parameters. To increase the reliability we use *Experiment Replication* (in statistical terms), which reduces the effect of the noises generated by errors.

Interactive hypothesis testing by the attacker:

We replicate the experiment numerous times to obtain statistically significant results. The attack leader has to do the following:

- specify the number of replications needed, *replicationNum*,
- specify the acceptable level of reliability, *successThreshold*,
- count the number of successful replications, *successCounter*, and then
- run an interactive statistical hypothesis testing algorithm to decide whether or not to accept and distribute the tested values of $x_{current}$ and x_{time} as reliable.

The attack leader can form the hypothesis testing in many different ways, for example:

1. The null hypothesis H_0: SuspicionValue $= 0$

   H_1: SuspicionValue $\neq 0$ (one sided hypothesis)

   The SuspicionValue variable is equal to *zero* if there is no change in the host behaviour (no migration) and is equal to *one* if there is a change in the host behaviour (possible migration).

2. Assume H_0 is true.

3. The null hypothesis distribution is computed by the number of permutations, which is equal to *replicationNum* (it should be *replicationNum*+1, but because successCounter ≥ 1, there will be at least one successful experiment).

4. Specify the significant level α.

   The values of the signification level is calculated based on the costs of committing a type I error (accepting and distributing inaccurate $x_{current}$ and x_{time}) and a type II error (rejecting an accurate $x_{current}$ and x_{time}). The attack leader can calculate the cost depending on different factors such as the available resources.

5. The leader calculates the *successThreshold* based on α.

6. Compute the t-test statistic.

7. Then a decision rule is formed based on the threshold to decide whether to reject H_0 (accept the tested values as reliable thus distribute them and use them for attacking) or to not reject H_0 (not accept the tested values and go to the next test phase).

8. Collect samples by running the experiment *replicationNum* a number of times (experiment replications) and count the number of successes and number of failures.

9. Draw a conclusion whether to reject or not reject the null hypothesis, H_0.

Although the test is replicated numerous times to obtain statistically significant results, there is still a possibility for type I and II errors. A type I error is rejecting a true null hypothesis. However, after the rejection, the leader will try higher $x_{current}$ and x_{time} values, which will trigger the attack using slightly higher than required values. In a type II error, which is accepting a false null hypothesis, the attacker might attack using insufficient time and strength (CPU utilisation). This might cause the attack to fail and cloud increase the possibility of being caught by security defences in the cloud. Therefore, when selecting the *successThreshold* value, the leader should consider the available security defences and balance the two errors.

Attacking the cloud using the revealed parameters:

After accepting the values of $x_{current}$ and x_{time}, the attack leader in the host can form a CIDoS attack based on these values. The value of T_s (the severity-threshold to be broken by the attackers) is $x_{current}$, and the duration of the attack is x_{time}. As described in section 5.3, in *scenario 1* of the CIDoS attack, the leader broadcasts the value of T_s, whereas in *scenario 2*, the leader distributes the units that each malicious VM has to cover, and these units are calculated by the leader depending on the value of T_s.

Without accurately knowing the migration parameters, attackers have to increase the workload to put the host in an over-utilised state. By using relatively accurate parameters, attackers can trigger migration without over-utilising the host by making the cloud management algorithms predict that the host will be in an over-utilised state, which causes it to migrate some of the VMs to avoid future SLA violations. Thus, the current workload will not break the thresholds, but the predicted workload will. This will make the attack harder to detect, and attackers will need less resources for the attack. Moreover, the parameters can be broadcast to all the CIDoS VMs, even in other hosts. This will significantly increase the damage,

especially because migration policies are usually the same in all the availability zones, and we assumed that all the hosts were homogeneous.

In addition, by having a predicted workload that is not over the threshold but is very close to it (because the used parameters are relatively accurate), the cloud might migrate *only one* VM from the host (rather than a large number of VMs, which would be the case if the workload was far over the threshold). Migrating one VM increases the lifetime of the attack because the participating VMs can increase their workloads to cover the loss of one VM and continue with the attack. This is especially valid in *scenario 1*, where large number of malicious VMs are available. The attackers can keep covering the loss gradually until there are not enough malicious VMs to attack with or there are no other non-malicious VMs in the host. The leader can determine that there are no other non-malicious VMs in the host if, for a series of migrations, only malicious VMs are being migrated. The leader can also determine this if the only existing host workload is the collection of the workloads of the malicious VMs. If the leader finds that this host is only occupied by malicious VMs, he/she can either reduce the workload to the minimum to allow new arrivals or terminate most of the malicious VMs to activate the policy of save energy by migrating all the VMs in that host and then turning it *off*, which causes greater damage than regular migration. This will consume the cloud resources and cause the cloud management machine to make decisions based on false reasons. Moreover, causing a migration of only one VM will also make the attack harder to detect because migrating a large number of VMs at the same time might raise suspicion and lead to further investigations.

5.5.3 Analysis and Discussion

Finding the values of $x_{current}$ and x_{time}, or any other hidden parameters, that are used by cloud management algorithms is a security threat by itself, especially considering that it is highly probable that these parameters are used everywhere in the cloud and are part of its underlying structure, which is relatively consistent. By using accurate parameters, the attack will live longer and become harder to detect. For simplicity, we only considered CPU utilisation. However, by adding memory utilisation and network traffic, the attack will be stronger and even harder to detect.

A change in the VM workload during the attack might be detected by a sensitive anomaly-based HIDS; however, a sensitive HIDS generates a large number of false negative errors, which increases the noise and decreases the value of alerts. The attack can be detected reliably by being aware of its technique and calculating the correlation between the workload patterns of VMs, which requires complex calculations and communication, and thus has a high cost.

If there is coordination among hosts (rather than a single host) to form the CIDoS attacks, cloud management machines will make a series of false resource consuming decisions, which might saturate the network. The cloud management machine might also start turning *on* more hosts to cope with the fake increase in demand or turning *off* hosts.

Moreover, if the attacker discovers the VM selection algorithm, he/she can avoid being migrated by, for instance, intensely using the memory, which will increase the cost of migrating the attacker VM and thus prevent being selected by the VM selection algorithm for migration. This is just an example; how to escape

migration depends on the selection algorithm used. However, the number of used algorithms is relatively small, which eases the attacker's task of discovering them. By avoiding migration, the attack will live longer because the group of malicious VMs that form the attack will stay together for a longer period of time and constantly attack the same host.

5.6 Conclusion and Future Work

In this chapter, we proposed a class of denial of service attacks, called Cloud-Internal DoS attacks, which are architecture-based. This class of attacks is designed to work specifically in large-scale public IaaS cloud environments. It misuses the 'migration' and 'over-commitment' features of the cloud. The attack is actualised by a novel protocol based on group key agreement (JKT protocol [86]) to establish a session key, PDV (to synchronise time), PCM (to digitised workload patterns), and leader election protocol (to choose the leader of the attack). The attack pattern is calculated and distributed using spread spectrum techniques. This attack, as with many cloud attacks, require co-residency and covert channels for communication. The protocol has a very low complexity $O(n + T)$ for the key agreement protocol and needs to send a very small stream of data (less than 100 bits in our example).

In the design of the protocol, we considered defending against different forms of man-in-the-middle attacks, i.e. replay attacks. Therefore, the protocol provides authentication and allows only members of the attack to participate in an active manner. The protocol is also portable and independent of the technical details of the environment.

Then, we suggested and tested methods for detecting different scenarios for the CIDoS class of attacks. We used the one-dimensional DCT and Euclidean distance to measure the correlation between the workload patterns of participating malicious VMs. The suggested methods successfully detected the attack and generated strong anomaly signals. We developed an algorithm to measure the correlation between workload patterns and check whether the correlation is increasing over time, which is a strong sign of the attack.

After that, we extended the CIDoS attack by performing a series of attacks to discover the migration thresholds and parameters. We proposed a technique to reverse engineer the cloud migration algorithms, overload the detection algorithm and save energy algorithm, and reveal hidden parameters and thresholds. Then, these parameters were used to improve the CIDoS attack, making it more harmful and harder to detect. We also designed a formal model for the migration decision process. Then, an algorithm was developed to extract the parameters from the model. We used anomaly-based HIDS to measure the success of the attack. The reliability of the extracted values was calculated using interactive statistical hypothesis testing. These values can be used to attack the host and can also be distributed to other malicious VMs in different hosts.

In future work, the attack can be scaled by coordination between hosts in the cloud, not just within one host. There is also a need to investigate the mitigation of the improved version of the attack. Furthermore, the next natural step for the improved version of the attack is moving it from the theoretical to the practical and

testing it in the lab. The area of avoiding migration, i.e. by intensifying the use of memory, is also interesting and needs further investigation.

Chapter 6

Summary, Conclusion, and Future Work

Nicholas Carr, in his popular book "The Big Switch", [37], made a very interesting comparison between computing power and electrical power in the early 20th century. Electricity started with in-house electrical generators, in companies and factories, and then moved to mass production for a power grid, which anyone can tap into. Electricity moved from being a product (generators, etc.) to a service. Organisations moved from being buyers to consumers, where they pay for what they use. Sharing has reduced cost. In addition, there is no longer a need for a team of electrical engineers to maintain the service and run the generators. The same thing is happening with computing power today; it is moving toward being a service, as a pay-for-use model, with smaller and less expert IT teams and a lower cost due to sharing, the selection of low-cost locations, and decreases in the cost of electricity, network bandwidth, operations, and hardware. There will be less money wasted and carbon emissions by increasing the utilisation of physical servers and other computing power to the maximum 24 hours a day and 7 days a week. This is even better than a fully equipped office rented by three organisations, where the day is distributed among them with eight hours each, without affecting each other's data or possessions. They share the service and the cost. The main concern in this model is security. What if the other organisation steals my data, or destroy them, even accidentally? It is exactly the same with cloud computing.

A large-scale public IaaS cloud is the model most suitable to play this role, and security is the main concern. To help cloud services reach their destinations, we have to improve all aspects of security and provide security assurance to consumers.

6.1 The Problem

One of the main problems facing the large-scale public IaaS cloud is monitoring the areas where they have no control. This is the area inside consumers' VMs. Providers have the responsibility to secure these VMs, the cloud infrastructure, and prevent any attack originating from their network, because it may impose liability on them. For providers to fulfil their security responsibilities, they have to have some form of control on these VMs activities. The isolation between VMs is maintained by the hypervisor, but there is insufficient isolation, and many attacks successfully break the isolation (i.e. VM escape, covert channels, and inter-VM attacks). Much research has been conducted to improve the isolation; however, the ground truth of sharing resources makes 100% isolation almost impossible.

Providers can install monitoring tools inside the hosted VMs, which will give them a clear vision and evaluation of the VMs' security status; this is not recommended for the following reasons:

1. It might increase the providers' liability in relation to a VM's illegal activities or content (i.e. if there is defamation or infringement of copyright contents in the consumers' VMs and any other illegal activities).
2. It will affect the privacy of consumers.
3. It will slow the monitoring process because in a large-scale cloud there will be hundreds of thousands of VMs (this is expected to be millions in the future), and collecting detailed data from inside the VMs will generate an extremely large amount of data to analyse and react to in real-time.
4. Cloud consumers are themselves organisations who are applying security standards and regulations and sharing their data with users, who might be restricted by regulations.
5. The cost model in the cloud is pay-for-use and generating extra activities inside VMs produced by the monitoring tools will increase the cost and cause confusion in cost measurement because consumers will not accurately pay only for what they used.

For these reasons, having no instrumentation within VMs, in addition to limiting the level of intrusiveness introduced by providers, is highly recommended. These are two main requirements for cloud security monitoring tools. In addition, other requirements include automatically collecting and analysing data, and then extracting information and reporting (when needed) without human intervention, because manual interaction with data in such tense mixed environments as the large-scale public cloud is not feasible. Second, the monitoring tool has to work on the fly, instantly monitoring, detecting, and reporting. Third, the used monitoring tools have to cope with high volumes of data without falling over. A high volume of data is a main characteristic of a large-scale public cloud, [148], [166] and [71], due to the large number of VMs, which target to increase the utilisation to the maximum. In addition, the VM's lifecycle such as cloning, migration, and snapshots for backups generates a large amount of data. Moreover, all the data and commands to and from VMs travel long distances through the Internet; all graphical outputs, mouse clicks, and even keyboard keystrokes come from long distances, which generate a greater data flow. A large amount of data requires tools that are simple to avoid posing a great overhead. They must be distributed (to avoid central monitoring), autonomous to be able to migrate with the corresponding VM, and automated.

When designing a monitoring tool to work in the large-scale public cloud, we have to assume that this tool will work in a hostile environment [138]. Almost any organisation or individual, and possibly an attacker, who has a payment card (i.e. credit card) can be a consumer. Furthermore, the process of renting the service is automatic, without any human intervention from the provider side because *on-demand self-service* is one of the essential characteristics of cloud computing, according to NIST [122].

The monitoring tools also have to be tamper resistant so a malicious consumer or an attacker cannot disable the monitoring tool or neutralise it. Furthermore, the

monitoring tools must have enough generality to monitor a wide variety of VMs [191]. Lastly, the tools have to be designed in a way that allows sharing security information with consumers or third parties when needed [70].

In the cloud architecture and supporting technologies, new threats and vulnerabilities have been created. Multi-tenancy is a great source of threat, especially in a public cloud where the service is shared by different organisations and individuals. In addition, isolation insufficiency emphasizes the threat imposed by sharing resources.

Another threat is that all administrative and critical commands and data in VMs are coming from abroad, travelling long distances through the Internet. Therefore, strong confidentiality mechanisms are required. Encryption can maintain the confidentiality of data during transformation. However, there is no guarantee that data are not being misused by cloud administrators, by, for example, revealing some sensitive information using searching and indexing on encrypted data.

A cloud also suffers from other security problems such as VMs patch management, the problem of infected images, the problem of trust, and the problem of lack of interoperability and vendor lock-in, which were discussed briefly in the thesis.

There are more security challenges in the cloud introduced by some of the usage paradigms such as ephemeral VMs. The use of ephemeral VMs are very popular in the cloud; these are VMs that live for very short periods of time (i.e. an hour or two) to perform a specific task and then terminate. This type of VM poses special security requirements for monitoring tools. For example, the setup of the monitoring tool has to be very quick and use only a small amount of data.

Three essential features of the large-scale public IaaS cloud make it more vulnerable to a denial of service attack (DoS), which are its openness, public multi-tenancy, and on-demand elasticity. The large-scale public cloud service is offered to the public through the Internet; therefore, malicious consumers can easily gain access to the cloud to perform DoS attacks. Cloud consumes (that are a service providers themselves in some models) are coming through long distances. Thus, any block of the service to them or their consumers on the way to the cloud is considered a DoS and might affect the reliability of the service.

Public multi-tenancy allows different clients to share the same physical host, and in some cases, as proved in the literature, malicious clients penetrate and steal data or resources from their innocent neighbours such as in a resource freeing attack (RFA) [164]. On-demand elasticity can be misused to perform economic-loss attacks, where innocent consumers are forced to pay more for fake reasons, i.e. the economic denial of sustainability attack (EDoS), which is discussed in [77], [152], and [167]. The goal of these attacks is financial gain, not disturbing the service. The victim of this attack will either pay extra money for the service he/she has not used or, if the victim has inactivated the *automatic* elasticity (to fix the cost), the attack will turn into a regular DoS.

Previous threats and challenges emphasised the importance of monitoring VMs to detect any malicious activities that threaten the infrastructure, other VMs, or the outside world. Monitoring is also important to detect DoS and evaluate the cloud's overall security status. It is also essential to discover and improve weak security controls and acknowledge malicious or security ignorant VMs in order to notify them, isolate them, and/or terminate their contracts if there is enough legal support

for termination.

Security monitoring is usually performed using intrusion detection systems (IDS), network-based IDS (NIDS) and host-based IDS (HIDS). A NIDS and HIDS are complimentary systems; a NIDS monitors the traffic, while an HIDS monitors the activities inside hosts. There are many problems facing a NIDS in the cloud, including the problem of monitoring and analysing encrypted traffic; and almost all data in the cloud network are encrypted. There is also a common NIDS problem when implemented in switched environments [102]. Switched environments will not allow NIDS sensors to inspect all the traffic because switches work based on connections, and the sensor will only receive packets addressed to it (unlike a hub that echoes every packet). There is a problem with coping with a large amount of data without being saturated, dropping packets, or failover [102]. Intrusions inside hosts that are not related to the network are outside the NIDS view. Therefore, a VM can be hijacked and start attacking the host without passing the NIDS' sensors. The communication between co-resident VMs is also outside of the NIDS view; therefore, inter-VM attacks and VM-escape attacks will not be discovered by a NIDS. Monitoring hosts and VMs are tasks of an HIDS, which monitors all of the communication between VMs and the activities inside VMs. Cloud NIDS restrictions and challenges emphasise the importance of using an HIDS in the cloud.

Because of the need for automation and minimum human intervention principles in the cloud, a signature-based HIDS is less suitable for monitoring hosts than an anomaly-based HIDS. In addition, an anomaly-based HIDS is more capable of detecting unknown attacks than a signature-based one. There are different types of HIDS that work in the cloud:

1. one that monitors the host,
2. one that is installed inside a VM (by the client), and used to monitor the VM from inside, and
3. one that is installed one the host (by the provider), and used to monitor VMs from outside.

The first is a regular HIDS to monitor the host activities; however, it is usually designed not to monitor a host with dynamic VMs (VMs that are continuously created, suspended, and migrated). The second HIDS is also a regular HIDS under the control of the client; therefore, the provider cannot use it as a source of data and monitoring. If the provider is allowed to access it, that will be against the requirements of monitoring tools in the large-scale public cloud, as stated before. The third HIDS is also called a hypervisor-based HIDS and is a good candidate to perform the required security monitoring because it is able to monitor hosted VMs without accessing them. Furthermore, it is tamper resistant because of being outside the monitored VMs.The main problem facing a hypervisor-based HIDS is bridging *the semantic gap*, extracting useful security information from the low-level data that is collected by monitoring VMs from outside only. The raw data must be reconstructed to understand what is really happening inside VMs.

All of these security requirements, challenges, and threats were discussed in detail throughout the thesis.

After showing the security problem and need in a large-scale public IaaS cloud, we proposed our solutions.

6.2 Proposed Solutions

For a hypervisor-based HIDS to work efficiently in the cloud, it has to conform to the following requirements. First, it has to cope with a large volume of data. Second, it has to work on the fly. It also has to avoid a single point of failure [95]. The monitoring must not disturb the cloud service nor impose a large overhead. The monitoring system has to be tamper resistant, so clients and attackers cannot disable it. It has to be able to monitor all the VMs of different types and specifications. The level of intrusiveness has to be the minimum without requiring any instrumentation within VMs. The dynamicity of VMs must be considered i.e. it has to be able to migrate with the corresponding VM. It should not require any prior knowledge about VMs. It has to be flexible and not centralised. Furthermore, the detection system should not require any human intervention to work effectively. The semantic gap has to be considered. All these requirements are in addition to the regular HIDS requirements, which are an acceptable overhead, a low false positive rate and an acceptable detection rate.

By following these requirements, and supported with our understating of the nature of a large-scale public IaaS cloud, we developed two hypervisor-based detection systems in chapter 4.

The main assumption behind the proposed detection systems is that VMs are limited in the used services and applications. This assumption is relative to regular servers that may aggregate multiple services on a single instance; regular best practices for IaaS cloud services would argue against such mixed workloads being deployed in a single VM for security and performance reasons. This assumption leads us to assume that VMs have relatively consistent behaviour; and consistency is essential to be able to build a normal behaviour profile that represents the monitored VM accurately. Moreover, in general, servers usually tend to repeatedly perform the same series of tasks, which provides a sound data set for training. This assumption drives us to a further assumption, in the IaaS cloud, any change in VM behaviour should be treated as a sign of abnormality.

By monitoring the whole VM as a single process, we were able to extract useful security information and detect anomalies. If the behaviour of the VM is not consistent or there are a mixture of applications and services on it, it will be hard to extract any useful security information, which will affect the efficiency of the detection system.

We proposed the idea of monitoring the whole VM as single process in an IaaS cloud environment. We successfully set a test environment, collected data, and built a normal behaviour profile for the VM using two representation methods to build two detection systems. One to monitor regular cloud VMs; while the other is to monitor ephemeral VMs.

The representation method used in the first detection system was a bag of system calls, which is very basic and less demanding representation method. The detection system we designed successfully classified normal and abnormal sequences of system calls and generated a strong anomaly signal. We found that a sequence length of 10 provided the best detection efficiency and a reasonable cost, while 6 was the best if the detection time was critical.

6.2.1 BoSC Hypervisor-based HIDS

The first proposed solution was a hypervisor-based HIDS built using bag of system calls (BoSC) method from [93]. We choose BoSC because it has been proven in the literature to provide a high accuracy and detection rate with low false positives and low computational demand. In BoSC, the order of the system calls is ignored, and only the frequencies are considered. By ignoring the order of the system calls, the classifier is built rapidly and can react on the fly. Furthermore, the generated database (normal behaviour profile) has a very small size, which allows it to migrate without imposing a high overhead.

We tested the new detection system in an experiment. We built an IaaS cloud network in the lab with an acceptable setup for production environments. The setup was built using Eucalyptus as the cloud management software, KVM as the hypervisor, Ubuntu as the operating system, and the Tryton application as the service provided by VMs. The Dogtail tool and Python were used to write the code for automating the use of Tryton with a randomness factor, and the Strace tool was used to collect system calls generated by the Tryton client VM and Tryton server VM.

We collected about 60 h of normal data; however, we used the minimum possible to train the classifier. Then, for testing purposes, we ran malicious activities inside the monitored VMs (activities to change the monitored VMs' normal behaviour). We ran a stress test code and collected system calls to use them for testing the classifier. We trained the classifier using normal data and tested it using normal samples not involved in the training, along with malicious samples. To reduce the overhead, increase the speed of training and detection, and reduce the size of the generated database (the normal behaviour profile), we considered the used system calls only, rather than all of the Linux system calls. We also covered the rarely used system calls with a new item added to the list of used system calls called 'other'. This trick significantly reduced the size of the databases without affecting the detection accuracy.

To register system calls, we used the sliding window from [63]. We tested sizes of 6, 10, and 20, and then compared their results to determine the effect of changing the length. The classifier with a sequence length of 10 provided the best results, with 100% accuracy, no false positives, and with an acceptable overhead (see table 6.1 for a comparison between different lengths results). A size of 6 was the best if we needed a very fast detection system; while a size of 10 provided the best accuracy and false positive rate, but with a slightly higher consumption of time than the size of 6. A size of 20 was not efficient because its accuracy and false positive rate were equal to those for a size of 10, but with a much higher computational time in comparison with sizes of 6 and 10.

We tested whether the generated anomaly signal was strong enough using hypothesis testing. The results proved that it was possible to build a normal behaviour profile for a whole VM via system call monitoring using the bag of system calls representation method. We successfully created an identity, and a sense of self, for the targeted VM. The used mechanism also successfully detected anomalies and generated strong enough anomaly signals.

The training algorithm has linear complexity, which makes it fast enough to be applied in real-time environments. The other algorithm is the detection algorithm,

Table 6.1: BoSC-based classifier results for sequence lengths of 6, 10, and 20.

	SIZE 6	SIZE 10	SIZE 20
Accuracy	98.36%	100.0%	100.0%
Detection Rate	100.00%	100.0%	100.0%
False Positive Rate	11.11%	0.00%	0.00%
Consumed Detection Time by Units	32	324	2492

which is very simple and short, but the bottleneck is the database. The algorithm's speed is dependent on the chosen sequence length; with an increase in sequence length, the consumed time increases significantly.

The used representation method is simple. Thus, it can deal with a large amount of data. The system can monitor VMs with almost no intrusiveness and requires no prior knowledge about the VMs. The detection system is separate from the monitored VM. Therefore, it has high attack resistance. It can work on the fly due to fast training, monitoring, and detection. It is also small and autonomous; thus, it can be migrated with the corresponding VM. Lastly, it does not require any human intervention, which is essential to be able to work in a large-scale cloud.

6.2.2 HMM Hypervisor-based HIDS for Ephemeral VMs

To monitor ephemeral VMs, we built another hypervisor-based detection system using the hidden Markov model (HMM) method. We reduced the number of HMM states, which are equal to the number of distinct system calls, to the minimum by only considering the used system calls and used the 'other' trick as in the previous classifier. By reducing the states, the computational demand and overhead were significantly reduced. We also tested this detection system in an experiment using the same setup as the previous classifier. We divided the data into chunks, with each chunk having 10 traces; if a chuck registered 4 mismatches, it was considered malicious. This trick reduced the detection time because in the experiment we found that about 50% of the malicious chunks registered 10 out of 10 mismatches, which meant the chuck was classified as malicious before the end of the analysis (this could be in the middle of the analysis), and when 4 mismatches were detected, the classifier stopped the analysis, raised and alert, and moved to the next chunk. In the experiment, the size of a chunk was equal to 150 byte, and in 50% of the samples, the classifier was able to decide if a chunk was malicious or not by checking only the first 75 byte of the chunk. Dividing traces into chunks helped to make the classifier work on the fly using a small amount of data for detecting anomalies.

In the HMM hypervisor-based HIDS, we tested three methods. First, we monitored the whole VM as a single process; this process had threads or children that were also monitored with the process. In the second method, we monitored only system calls with a high threat (IOCTL system calls and system calls with a high threat level). We imported the list of high threat level system calls from [21] with modifications. In the third method, we monitored the main threads only, which were defined as the threads that invoke IOCTL system calls. We chose the IOCTL system call because in a KVM virtual environment, an IOCTL system call is the

most critical system call for the reason that VMs communicate with resources in the host kernel using IOCTL.

Methods two and three failed to recognise malicious events, while method one provided a 97% accuracy, 100% detection rate, and 5.66% false positive rate (see 6.2). It needed only about 19 MB of data to train the classifier and less than 150 KB of data for detection, which are very small amounts of data. This made building the classifier and detecting anomalies very fast, which suited the nature of ephemeral VMs.

Table 6.2: Results of testing 1500 samples with chunk size of 10 and sequence length of 6, where each sample contained 1000 sequence and 8 iterations.

Accuracy	Detection Rate	False Positive Rate
97%	100%	5.66%

We proved by hypothesis testing that the proposed classifier generated a strong enough anomaly signal.

As we can see, the proposed system satisfies the special requirements for a cloud host-based IDS, in addition to the ability to detect anomalies in ephemeral VMs. The system processes the collected data (system calls) in way that reduces the complexity of the HMM while still being able to reliably detect anomalies in ephemeral VMs. It requires no prior knowledge about the VM from inside, dealing with VMs as black boxes, which is a requirement in a public IaaS cloud, where VMs are under the full control of the clients.

6.3 Cloud-Internal Denial of Service Class of Attacks

In this research, we also developed an architecture-based DoS attack targeting large-scale public IaaS cloud environments, which is shown in chapter 5. This attack takes advantage of the cloud host tendency to commit to more than it can deliver, 'over-commitment'. Over-commitment is essential to increase the utilisation to the maximum possible, because it is assumed that VMs will not use all of the resources dedicated to them to the maximum 24/7. If there is coordination between some of the host tenants (attack members), and they increase their workload following a plan to make the host detect the occurrence of over-utilisation, a migration order will be initiated to reduce the stress in the host. Migration is targeted because it is a very expensive process; it consumes bandwidth, processing time, and memory, causing a series of management algorithms to be run and increasing the attack service and risk of security incidents during migration.

We designed a protocol to coordinate the attack among members. This protocol is based on group key agreement (JKT protocol [86]) to establish a session key, PDV (to synchronise time), PCM (to digitised workload patterns), and a leader election protocol (to choose the leader of the attack).

Attack members in the CIDoS attack increase their workload following the attack pattern distributed by the attack leader. The leader calculates the host-expected workload and estimates the amount of workload needed to break a severity-threshold and trigger migration. The estimated workload is then distributed among attack members. The attack pattern is distributed using spread

spectrum techniques. The attack, as in many cloud attacks, requires co-residency and covert channel for communication. The protocol has a very low complexity $O(n + T)$ for the key agreement protocol and requires the transmission of a very small stream of data (less than 100 bits in our example).

In the design of the protocol, we considered defending against different forms of man-in-the-middle attacks, i.e. replay attacks. Therefore, the protocol provides authentication and allows only members of the attack to participate in an active manner. The protocol is also portable and independent of the environment's technical details

To discover the severity-threshold that has to be broken, we designed another attack (which is still theoretical). This new attack is an improvement on the CIDoS class of attacks. We tried to reveal some of the cloud parameters that are used in migration algorithms so we could use them to calculate the severity-threshold. We wanted the host to predict that it will be over-utilised without actually over-utilising the host. There are prediction algorithms running in the cloud to predict over-utilisation before it occurs to prevent a possible SLA breach. We focused on overload detection algorithm and energy saving algorithm. We also designed a formal model for the migration decision process. Then, an algorithm was developed to extract the parameters from the model. The required parameters are revealed by running numerous episodes of the CIDoS attack using parameters within a range specified by the attack leader. After each run of the protocol, a migration check has to be performed (to measure the success of the attack). We used an anomaly-based HIDS to measure the success of the attack. The reliability of the extracted values was calculated using interactive statistical hypothesis testing. These values can be used to attack the host and can also be distributed to other malicious VMs in different hosts.

By using accurate parameters in a CIDoS attack, the attack will live longer and become harder to detect because the number of VMs migrated in the same time is small. It is sufficient to make the cloud management algorithms predict that the current over-utilisation threshold will be broken without actually breaking it. For simplicity, we only considered the CPU utilisation. However, by adding memory utilisation and network traffic, the attack will be stronger and even harder to detect.

Revealing migration parameters is a security threat in itself, and the revealed parameters can be used in different attacks. The harm is significant because it is highly probable that these values are used everywhere in the cloud and changing them require changing the management algorithms of the entire large-scale public IaaS cloud, which requires a large amount of configuration and testing before they can be applied.

Furthermore, if this attack is coordinated between hosts (rather than being confined to a single host), the cloud management node will make a series of false resource-consuming decisions, which might saturate the network; the cloud management might also start turning *on* more hosts to cope with the fake expansion. Moreover, if the attacker discovers the VM selection algorithm, he/she can avoid being migrated by, for instance by intensely using the memory, which will make the cost of migrating them high; this is just an example, many techniques can be used to escape migration. The method to escape migration depends on the used VM selection algorithm. However, the number of used algorithms is relatively small,

which makes it easier for the attacker to discover them. By avoiding migration, the attack will live longer because the group of malicious VMs that form the attack will stay together for long time and constantly attack the same host.

In chapter 5, we also discussed some mitigation strategies for the CIDoS attack (without the improvement). The detection methods were based on a correlation calculation. We used the one-dimensional DCT and Euclidean distance to measure the correlation between the workload patterns of participating malicious VMs. The suggested methods successfully detected the attack and generated strong anomaly signals. We developed an algorithm to measure the correlation between workload patterns and check whether the correlation was increasing over time, which is a strong sign of an attack.

To verify the validity of the suggested detection techniques, we created synthetic workload signals based on our understanding of the CIDoS attack and the following assumptions: first, the workload pattern of each VM differs randomly from other VMs unless it is abnormal, and each VM has a stable workload pattern which is not random, because in the real-world, servers usually have stable workload patterns; they perform the same series of tasks repeatedly.

In our experiment, we found that the distances between malicious signals were low in comparison with the distances between normal ones. Some normal VMs might coincidentally have high correlation with malicious VMs in some points. However, the correlation between malicious VMs has to decrease over time because they are coordinating to match, and normal VMs will not follow the same pattern. Therefore, having a coincident correlation between normal and malicious VMs will not last for the whole test and will not cause a false alarm.

The suggested detection methods successfully detected the attack and generated strong enough anomaly signals. We proved that the generated anomaly signal was strong enough using hypothesis testing, where we retained the null hypothesis.

When representing the detection system, we ignored weak amplitudes by representing them as zeroes, while considering only strong amplitudes by representing them as ones. This trick reduced the detection time and complexity of the operations. In addition, the Euclidean distance and one-dimensional DCT are very simple and straightforward techniques. Applying simple techniques is essential to run the detection on the fly and to be able to detect an attack in time before it is completely formed. Its contribution is the ability to detect the attack using suitable detection mechanisms. We also discussed some prevention and response techniques for the CIDoS attack.

6.4 Future Work

This research has raised many questions in need of further investigation. First and foremost, the improved version of the CIDoS attack has to be moved from the theoretical to practical stage and tested in the lab, which we have already started to do. We are using CloudSim to test the improved version of the CIDoS class of attacks. CloudSim is a framework for simulating an IaaS cloud environment. It runs the most popular migration algorithms, which we are attempting to reverse engineer to extract the required migration parameters. The work on the attack testing

will involve simulations. This will be my next project in coordination with other researchers as a team at King Abdul Aziz University in Saudi Arabia.

The simulations will check the performance of the proposed algorithm to discover some of the cloud migration hidden parameters. We will investigate the following questions:

- Can we detect which migration algorithms are in use?
- Can we measure the sensitivity of the algorithms to estimate the trigger point?
- Can we say something about the decision characteristics themselves?

Some examples of the resource management algorithms from CloudSim that we will consider are shown in figure 6.1.

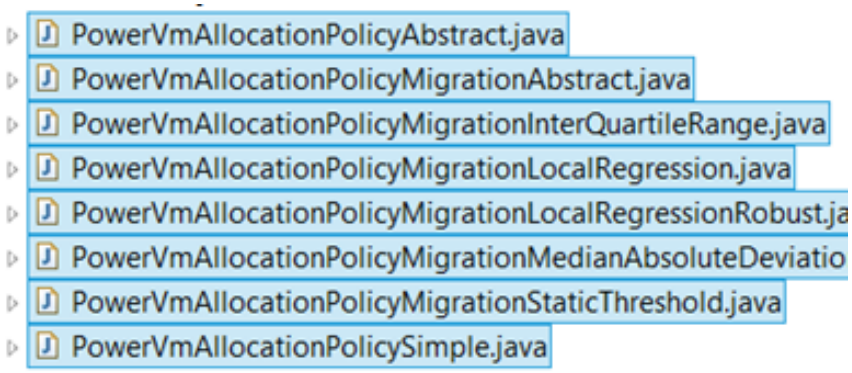

Figure 6.1: A list of cloud resource management algorithms from CloudSim.

Most of these algorithms are based on a study conducted in 2012 by Beloglazov and Buyya at the University of Melbourne, Australia [20].

We need to manually study the used algorithms and find the trigger thresholds. Then, as a proof of principle we will try to detect any of the used algorithms. We can only apply the attack in a simulated environment. We studied the possibility of applying it in Amazon EC2. However, legal restrictions prohibit any security test in the Amazon network.

Another interesting and critical area of research is a method to prevent a VM from being migrated by misusing the VM selection algorithms discussed in section 5.5.1; in this research, we suggested that the VM could avoid migration by increasing the cost of migration through intensifying the use of memory. However, this is just an example. Other ways to avoid migration, in addition to the suggested one, could be studied and tested in the future.

In the area of cloud hypervisor-based HIDS, the following areas require further investigation and testing. First, considering system call arguments in the hypervisor-based detection and measuring the accuracy, overhead, and detection rate. In our research, we ignored arguments to reduce the complexity of training and running the IDS. However, the effect of considering the arguments on the accuracy and the exposed overhead has not been studied, but is a good topic for future work.

The effect of a mimicry attack on a hypervisor-based HIDS in a cloud environment that was introduced in [169] and discussed in section 4.5 requires further

investigation and testing in cloud environments. The problem of classifier drafting, especially for ephemeral VM anomaly detection systems, requires more attention. Another interesting problem is how to determine whether or not a newly initiated VM is a short-lived one. Although this is not a security problem, many security solutions may rely on it.

The area of preventing cloud cartography and preventing attackers from mapping the cloud (section 5.1.1 and [139]) are important and require more research because they would prevent many cloud attacks that rely on mapping the cloud.

We assumed that VMs are not malicious when they first start running. However, if they are, the anomaly detection systems will fail. This area needs more research; this problem is related to authentication mechanisms and the process of initiating new VMs. Finally, we believe that the usage pattern of the cloud should be more regulated, and providers should encourage their consumers to maintain a consistent behaviour, because with anomaly detection systems, a more consistent VM is more secure.

Bibliography

[1] ISECT LTD. ISO/IEC 27018, 2014. 17

[2] ABU-RGHEFF, M. A. 3 - fundamentals of spread-spectrum techniques. In *Introduction to {CDMA} Wireless Communications*, M. A. Abu-Rgheff, Ed. Academic Press, 2007, pp. 153 – 194. doi:http://dx.doi.org/10.1016/B978-075065252-0.50004-1. 109

[3] ALARIFI, S., AND WOLTHUSEN, S. Detecting anomalies in iaas environments through virtual machine host system call analysis. In *Internet Technology And Secured Transactions, 2012 International Conference for* (2012), pp. 211–218. 8

[4] ALARIFI, S., AND WOLTHUSEN, S. Anomaly detection for ephemeral cloud iaas virtual machines. In *Network and System Security*, J. Lopez, X. Huang, and R. Sandhu, Eds., vol. 7873 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2013, pp. 321–335. doi:10.1007/978-3-642-38631-2_24. 9

[5] ALARIFI, S., AND WOLTHUSEN, S. Robust coordination of cloud-internal denial of service attacks. In *Cloud and Green Computing (CGC), 2013 Third International Conference on* (Sept 2013), pp. 135–142. doi:10.1109/CGC.2013.28. 9, 20

[6] ALARIFI, S., AND WOLTHUSEN, S. Mitigation of cloud-internal denial of service attacks. In *Service Oriented System Engineering (SOSE), 2014 IEEE 8th International Symposium on* (April 2014), pp. 478–483. doi:10.1109/SOSE.2014.71. 9

[7] ALARIFI, S., AND WOLTHUSEN, S. D. Dynamic parameter reconnaissance for stealthy dos attack within cloud systems. In *IFIP* (2014). 9

[8] ALHARKAN, T., AND MARTIN, P. Idsaas: Intrusion detection system as a service in public clouds. In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on* (May 2012), pp. 686–687. doi:10.1109/CCGrid.2012.81. 46

[9] ALLIANCE, C. S. The Notorious Nine Cloud Computing Top Threats in 2013. *Cloud Security Alliance* (Feb. 2013). Accessed: 2014-01-06. 16, 29

[10] ANTONOPOULOS, N., AND GILLAM, L. *Cloud Computing: Principles, Systems and Applications*, 1st ed. Springer Publishing Company, Incorporated, 2010. 10

[11] ARMBRUST, M., FOX, A., GRIFFITH, R., JOSEPH, A. D., KATZ, R. H., KONWINSKI, A., LEE, G., PATTERSON, D. A., RABKIN, A., STOICA, I., AND ZAHARIA, M. Above the clouds: A berkeley view of cloud computing. Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009. 4, 10, 11, 15, 27, 56, 96

[12] ASRIGO, K., LITTY, L., AND LIE, D. Using vmm-based sensors to monitor honeypots. In *VEE* (2006), H.-J. Boehm and D. Grove, Eds., ACM, pp. 13–23. 39

[13] AYAD, A., AND DIPPEL, U. Agent-based monitoring of virtual machines. In *Information Technology (ITSim), 2010 International Symposium in* (June 2010), vol. 1, pp. 1–6. doi:10.1109/ITSIM.2010.5561375. 33

[14] AZMANDIAN, F., MOFFIE, M., ALSHAWABKEH, M., DY, J., ASLAM, J., AND KAELI, D. Virtual machine monitor-based lightweight intrusion detection. *SIGOPS Oper. Syst. Rev. 45*, 2 (July 2011), 38–53. doi:10.1145/2007183.2007189. 45, 56, 91

[15] BADGER, M. L., GRANCE, T., PATT-CORNER, R., AND VOAS, J. M. Cloud computing synopsis and recommendations. Tech. rep., NIST, Gaithersburg, MD, United States, 2012. 5, 7, 8, 95

[16] BALA, A., AND CHANA, I. Vm migration approach for autonomic fault tolerance in cloud computing. *Int'l Conf. Grid and Cloud Computing and Applications — GCA'13 —* (2013). 30, 122, 123, 124

[17] BARNETT, R. Waf virtual patching challenge: Securing webgoat with modsecurity. Tech. rep., Breach Security, 01 2009. 32

[18] BASHIR, M. N., KESAN, J. P., HAYES, C. M., AND ZIELINSKI, R. Privacy in the cloud: Going beyond the contractarian paradigm. In *Proceedings of the 2011 Workshop on Governance of Technology, Information, and Policies* (New York, NY, USA, 2011), GTIP '11, ACM, pp. 21–27. doi:10.1145/2076496.2076499. 15

[19] BATES, A., MOOD, B., PLETCHER, J., PRUSE, H., VALAFAR, M., AND BUTLER, K. Detecting co-residency with active traffic analysis techniques. In *Proceedings of the 2012 ACM Workshop on Cloud computing security workshop* (New York, NY, USA, 2012), CCSW '12, ACM, pp. 1–12. doi:10.1145/2381913.2381915. 48, 50, 98, 99, 100, 103, 113, 128

[20] BELOGLAZOV, A., AND BUYYA, R. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurr. Comput. : Pract. Exper. 24*, 13 (Sept. 2012), 1397–1420. doi:10.1002/cpe.1867. 96, 122, 123, 124, 144

[21] BERNASCHI, M., GABRIELLI, E., AND MANCINI, L. V. Operating system enhancements to prevent the misuse of system calls. In *ACM Conference on Computer and Communications Security* (2000), D. Gritzalis, S. Jajodia, and P. Samarati, Eds., ACM, pp. 174–183. viii, 61, 62, 85, 86, 140

[22] BILGE, L., AND DUMITRAS, T. Before we knew it: An empirical study of zero-day attacks in the real world. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (New York, NY, USA, 2012), CCS '12, ACM, pp. 833–844. doi:10.1145/2382196.2382284. 35

[23] BLAHUT, R. *Fast Algorithms for Signal Processing*. Cambridge University Press, 2010. 117

[24] BOHN, R. B., MESSINA, J., LIU, F., TONG, J., AND MAO, J. Nist cloud computing reference architecture. In *Proceedings of the 2011 IEEE World Congress on Services* (Washington, DC, USA, 2011), SERVICES '11, IEEE Computer Society, pp. 594–596. doi:10.1109/SERVICES.2011.105. 21

[25] BRAX, C. *Anomaly Detection in the Surveillance Domain*. Ph.D. thesis, University of SkvdeUniversity of Skvde, School of Humanities and Informatics, The Informatics Research Centre, 2011. 24

[26] BREITGAND, D., KUTIEL, G., AND RAZ, D. Cost-aware live migration of services in the cloud. In *Proceedings of the 3rd Annual Haifa Experimental Systems Conference* (New York, NY, USA, 2010), SYSTOR '10, ACM, pp. 11:1–11:1. doi:10.1145/1815695.1815709. 30

[27] BRESSOUD, T. C., AND SCHNEIDER, F. B. Hypervisor-based fault tolerance. *SIGOPS Oper. Syst. Rev. 29*, 5 (Dec. 1995), 1–11. doi:10.1145/224057.224058. 40

[28] BUGIEL, S., NÜRNBERGER, S., PÖPPELMANN, T., SADEGHI, A.-R., AND SCHNEIDER, T. Amazonia: When elasticity snaps back. In *Proceedings of the 18th ACM Conference on Computer and Communications Security* (New York, NY, USA, 2011), CCS '11, ACM, pp. 389–400. doi:10.1145/2046707.2046753. vii, 20, 31, 33, 34

[29] BUGNION, E., DEVINE, S., AND ROSENBLUM, M. Disco: Running commodity operating systems on scalable multiprocessors. In *ACM Transactions on Computer Systems* (1997), pp. 143–156. 40

[30] BURMESTER, M., AND DESMEDT, Y. A secure and efficient conference key distribution system. In *Advances in Cryptology EUROCRYPT'94*, A. Santis, Ed., vol. 950 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1995, pp. 275–286. doi:10.1007/BFb0053443. 106

[31] BURMESTER, M., AND DESMEDT, Y. Efficient and secure conference-key distribution. In *Security Protocols*, M. Lomas, Ed., vol. 1189 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1997, pp. 119–129. doi:10.1007/3-540-62494-5_12. 106

[32] BUYYA, R., BELOGLAZOV, A., AND ABAWAJY, J. H. Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges. *CoRR abs/1006.0308* (2010). 122, 123, 124

[33] CABUK, S., BRODLEY, C. E., AND SHIELDS, C. Ip covert timing channels: Design and detection. In *Proceedings of the 11th ACM Conference on Computer and Communications Security* (New York, NY, USA, 2004), CCS '04, ACM, pp. 178–187. doi:10.1145/1030083.1030108. 101

[34] CACHIN, C., AND STROBL, R. Asynchronous group key exchange with failures. In *Proceedings of the Twenty-third Annual ACM Symposium on Principles of Distributed Computing* (New York, NY, USA, 2004), PODC '04, ACM, pp. 357–366. doi:10.1145/1011767.1011820. 106

[35] CAMPBELL, R., MONTANARI, M., AND FARIVAR, R. A middleware for assured clouds. *Journal of Internet Services and Applications 3*, 1 (2012), 87–94. doi:10.1007/s13174-011-0044-9. 15, 20, 28

[36] CARBONE, M., CONOVER, M., MONTAGUE, B., AND LEE, W. Secure and robust monitoring of virtual machines through guest-assisted introspection. In *Research in Attacks, Intrusions, and Defenses*, D. Balzarotti, S. Stolfo, and M. Cova, Eds., vol. 7462 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, pp. 22–41. doi:10.1007/978-3-642-33338-5_2. 48

[37] CARR, N. *The Big Switch: Rewiring the World, from Edison to Google*. W.W. Norton & Company, 2009. 134

[38] CHALLAL, Y., BETTAHAR, H., AND BOUABDALLAH, A. A taxonomy of multicast data origin authentication: Issues and solutions. *IEEE Communications Surveys and Tutorials 6*, 1-4 (2004), 34–57. 105

[39] CHEN, P., AND NOBLE, B. When virtual is better than real [operating system relocation to virtual machines]. In *Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on* (2001), pp. 133–138. doi:10.1109/HOTOS.2001.990073. 40

[40] CHEN, Y., PAXSON, V., AND KATZ, R. H. What's new about cloud computing security? Tech. Rep. UCB/EECS-2010-5, EECS Department, University of California, Berkeley, Jan 2010. 26, 27

[41] CHERKASOVA, L., OZONAT, K. M., MI, N., SYMONS, J., AND SMIRNI, E. Anomaly application change or workload change towards automated detection of application performance anomaly and change. In *DSN* (2008), IEEE Computer Society, pp. 452–461. 128

[42] CLOUD SECURITY ALLIANCE. CLOUD SECURITY ALLIANCE RELEASES NEW CLOUD CONTROLS MATRIX V3.0.1 AND CONSENSUS ASSESSMENTS INITIATIVE QUESTIONNAIRE V3.0.1, 2014. 16

[43] COHEN, W. W. Fast effective rule induction. In *In Proceedings of the Twelfth International Conference on Machine Learning* (1995), Morgan Kaufmann, pp. 115–123. 61

[44] CVE. CVE-2011-1751, 2011. 31

[45] DASTJERDI, A., BAKAR, K., AND TABATABAEI, S. Distributed intrusion detection in clouds using mobile agents. In *Advanced Engineering Computing and Applications in Sciences, 2009. ADVCOMP '09. Third International Conference on* (Oct 2009), pp. 175–180. doi:10.1109/ADVCOMP.2009.34. 43

[46] DAVIS, I. J., HEMMATI, H., HOLT, R. C., GODFREY, M. W., NEUSE, D. M., AND MANKOVSKII, S. Regression-based utilization prediction algorithms: An empirical investigation. In *Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research* (Riverton, NJ, USA, 2013), CASCON '13, IBM Corp., pp. 106–120. 124, 126

[47] DEBAR, H., CURRY, D., AND FEINSTEIN, B. The Intrusion Detection Message Exchange Format (IDMEF). RFC 4765, RFC Editor, March 2007. 44

[48] DEMICHELIS, C., AND CHIMENTO, P. IP Packet Delay Variation Metric for IP Performance Metrics (IPPM). RFC 3393 (Proposed Standard), November 2002. 108

[49] DENZ, R., AND TAYLOR, S. A survey on securing the virtual cloud. *Journal of Cloud Computing: Advances, Systems and Applications 2* (Nov. 2013). doi:10.1186/2192-113X-2-17. 37

[50] DOCUMENTATION, E. Euca2ools user guide by eucalyptus community. Accessed: 2014-01-27. 69

[51] DOCUMENTATION, U. Getting started with ubuntu enterprise cloud. 66

[52] DUNLAP, G. W., KING, S. T., CINAR, S., BASRAI, M. A., AND CHEN, P. M. Revirt: Enabling intrusion analysis through virtual-machine logging and replay. In *In Proceedings of the 2002 Symposium on Operating Systems Design and Implementation (OSDI* (2002), pp. 211–224. 40

[53] ELHAGE, N. Virtunoid: A KVM Guest Host privilege, escalation exploit, 2011. 31, 48, 49

[54] ELMROTH, E., TORDSSON, J., HERNÁNDEZ, F., ALI-ELDIN, A., SVÄRD, P., SEDAGHAT, M., AND LI, W. Self-management challenges for multi-cloud architectures. In *Proceedings of the 4th European Conference on Towards a Service-based Internet* (Berlin, Heidelberg, 2011), ServiceWave'11, Springer-Verlag, pp. 38–49. 123, 124

[55] EUCALYPTUS SYSTEMS, I. Eucalyptus. Accessed: 2014-01-27. 65, 73

[56] EUCALYPTUS SYSTEMS, I. Eucalyptus cloud computing platform administrator's guide enterprise edition 2.0. Tech. rep., Eucalyptus Systems, 2011. https://www.eucalyptus.com/docs/eucalyptus/2.0/ag-ee.pdf. vii, 54, 55, 66, 67, 68, 69, 70

[57] EUROPEAN UNION AGENCY FOR NETWORK AND INFORMATION SECURITY (ENISA). Procure Secure: A guide to monitoring of security service levels in cloud contracts, 2014. 17

[58] EVELYN BROWN . Final Version of NIST Cloud Computing Definition Published, 2011. 12

[59] FAN, K., MAO, D., LU, Z., AND WU, J. Ops: Offline patching scheme for the images management in a secure cloud environment. In *Services Computing (SCC), 2013 IEEE International Conference on* (June 2013), pp. 587–594. doi:10.1109/SCC.2013.57. 32

[60] FARID, D. M., RAHMAN, M. Z., AND RAHMAN, C. M. Article: Adaptive intrusion detection based on boosting and naive bayesian classifier. *International Journal of Computer Applications 24*, 3 (June 2011), 12–19. Published by Foundation of Computer Science. 24

[61] FOR STANDARDIZATION, I. O. ISO27001: Information Security Management System (ISMS) standard. *Online: http://www.27000.org/iso-27001.htm* (Oct. 2005). 23

[62] FORREST, S., HOFMEYR, S., AND SOMAYAJI, A. The evolution of system-call monitoring. In *Proceedings of the 2008 Annual Computer Security Applications Conference* (Washington, DC, USA, 2008), ACSAC '08, IEEE Computer Society, pp. 418–430. doi:10.1109/ACSAC.2008.54. 58

[63] FORREST, S., HOFMEYR, S., SOMAYAJI, A., AND LONGSTAFF, T. A sense of self for unix processes. In *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on* (1996), pp. 120–128. doi:10.1109/SECPRI.1996.502675. 3, 42, 56, 58, 59, 60, 61, 62, 78, 139

[64] FOUNDATION, T. A. S. Mapreduce tutorial. 45

[65] FUNFROCKEN, S. How to integrate mobile agents into web servers. In *WET-ICE* (1997), IEEE Computer Society, pp. 94–99. 36

[66] GARFINKEL, T., AND ROSENBLUM, M. A virtual machine introspection based architecture for intrusion detection. In *In Proc. Network and Distributed Systems Security Symposium* (2003), pp. 191–206. 31, 32, 38, 39, 40, 41

[67] GARRISON, C. *Digital Forensics for Network, Internet, and Cloud Computing: A Forensic Evidence Guide for Moving Targets and Data*. Elsevier Science, 2010. 27, 34

[68] GHANAM, Y., FERREIRA, J., AND MAURER, F. Emerging Issues and Challenges in Cloud Computing: A Hybrid Approach. *Journal of Software Engineering and Applications 5*, 11A (Nov. 2012). doi:10.4236/jsea.2012.531107. vii, 27

[69] GOLDBERG, R. P. Survey of virtual machine research. *Computer 7*, 9 (Sept. 1974), 34–45. doi:10.1109/MC.1974.6323581. 39

[70] GROBAUER, B., AND SCHRECK, T. Towards incident handling in the cloud: Challenges and approaches. In *Proceedings of the 2010 ACM Workshop on Cloud Computing Security Workshop* (New York, NY, USA, 2010), CCSW '10, ACM, pp. 77–86. doi:10.1145/1866835.1866850. 21, 22, 23, 28, 38, 114, 136

[71] GUL, I., AND HUSSAIN, M. Distributed cloud intrusion detection model. *International Journal of Advanced Science and Technology 34* (2011). 38, 135

[72] HATANO, T., MIYAJI, A., AND SATO, T. T-robust scalable group key exchange protocol with o(logn) complexity. In *Information Security and Privacy*, U. Parampalli and P. Hawkes, Eds., vol. 6812 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2011, pp. 189–207. doi:10.1007/978-3-642-22497-3_13. 106

[73] HE, J., TANG, C., YANG, Y., QIAO, Y., AND LIU, C. 3d-ids: Iaas user-oriented intrusion detection system. In *Information Science and Engineering (ISISE), 2012 International Symposium on* (Dec 2012), pp. 12–15. doi:10.1109/ISISE.2012.12. 36, 37, 46

[74] HERBST, N. R., KOUNEV, S., AND REUSSNER, R. Elasticity in cloud computing: What it is, and what it is not. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)* (San Jose, CA, 2013), USENIX, pp. 23–27. 13

[75] HERZBERG, A., SHULMAN, H., ULLRICH, J., AND WEIPPL, E. Cloudoscopy: Services discovery and topology mapping. In *Proceedings of the 2013 ACM Workshop on Cloud Computing Security Workshop* (New York, NY, USA, 2013), CCSW '13, ACM, pp. 113–122. doi:10.1145/2517488.2517491. 98, 99, 100, 114

[76] HOANG, X., AND HU, J. An efficient hidden markov model training scheme for anomaly intrusion detection of server applications based on system calls. In *Networks, 2004. (ICON 2004). Proceedings. 12th IEEE International Conference on* (Nov 2004), vol. 2, pp. 470–474 vol.2. doi:10.1109/ICON.2004.1409210. 86

[77] HOFF, C. The economic denial of sustainability concept, 2008. from Rational Security: http://rationalsecurity.typepad.com/blog/edos. 34, 49, 136

[78] HOFMEYR, S. A., FORREST, S., AND SOMAYAJI, A. Intrusion detection using sequences of system calls. *J. Comput. Secur. 6*, 3 (Aug. 1998), 151–180. 60, 61, 80

[79] HOOPES, J. *Virtualization for Security: Including Sandboxing, Disaster Recovery, High Availability, Forensic Analysis, and Honeypotting*. Including Sandboxing, Disaster Recovery, High Availability, Forensic Analysis, and Honeypotting Series. Elsevier Science, 2009. 31, 32

[80] HU, J. Host-based anomaly intrusion detection. In *Handbook of Information and Communication Security*, P. Stavroulakis and M. Stamp, Eds. Springer Berlin Heidelberg, 2010, pp. 235–255. doi:10.1007/978-3-642-04117-4_13. 35, 36, 37, 85

[81] HU, J., YU, X., QIU, D., AND CHEN, H.-H. A simple and efficient hidden markov model scheme for host- based anomaly intrusion detection. *Netwrk. Mag. of Global Internetwkg. 23*, 1 (Jan. 2009), 42–47. doi:10.1109/MNET.2009.4804323. 85

[82] HUANG, C. T., AND GOUDA, M. G. An anti-replay window protocol with controlled shift. In *ICCCN* (2001), J. J. Li, R. P. Luijten, and E. K. Park, Eds., IEEE, pp. 242–247. 105

[83] IEEE. 2301 - Guide for Cloud Portability and Interoperability Profiles (CPIP), 2014. 17

[84] ISECT LTD. ISO/IEC 27017, 2014. 17

[85] JANSEN, W., AND GRANCE, T. Sp 800-144. guidelines on security and privacy in public cloud computing. Tech. rep., National Institute of Standards & Technology, Gaithersburg, MD, United States, 2011. 17

[86] JARECKI, S., KIM, J., AND TSUDIK, G. Robust group key agreement using short broadcasts. In *Proceedings of the 14th ACM Conference on Computer and Communications Security* (New York, NY, USA, 2007), CCS '07, ACM, pp. 411–420. doi:10.1145/1315245.1315296. 6, 106, 107, 132, 141

[87] JIANG, L., TIAN, G., AND ZHU, S. Design and implementation of network forensic system based on intrusion detection analysis. In *Proceedings of the 2012 International Conference on Control Engineering and Communication Technology* (Washington, DC, USA, 2012), ICCECT '12, IEEE Computer Society, pp. 689–692. doi:10.1109/ICCECT.2012.51. 35

[88] JIANG, X., AND WANG, X. "out-of-the-box" monitoring of vm-based high-interaction honeypots. In *Proceedings of the 10th International Conference on Recent Advances in Intrusion Detection* (Berlin, Heidelberg, 2007), RAID'07, Springer-Verlag, pp. 198–218. 41

[89] JIN, H., XIANG, G., ZHAO, F., ZOU, D., LI, M., AND SHI, L. Vmfence: A customized intrusion prevention system in distributed virtual computing environment. In *Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication* (New York, NY, USA, 2009), ICUIMC '09, ACM, pp. 391–399. doi:10.1145/1516241.1516310. 42, 43

[90] JONES, R. A., AND MARTINEZ, J. E. *Federal Cloud Computing: Strategy and Considerations*. Nova Science Publishers, Inc., Commack, NY, USA, 2012. 7

[91] JOSHI, N., AND J, V. Intrusion detection in virtualized environment. *International Journal of Computational Engineering Research 3* (June 2013), 19–25. 19, 36, 38

[92] JYOTHSNA, V., PRASAD, V. V. R., AND PRASAD, K. M. Article: A review of anomaly based intrusion detection systems. *International Journal of Computer Applications 28*, 7 (August 2011), 26–35. Published by Foundation of Computer Science, New York, USA. 23, 24, 58

[93] KANG, D.-K., FULLER, D., AND HONAVAR, V. Learning classifiers for misuse and anomaly detection using a bag of system calls representation. In *Information Assurance Workshop, 2005. IAW '05. Proceedings from the Sixth Annual IEEE SMC* (June 2005), pp. 118–125. doi:10.1109/IAW.2005.1495942. viii, 3, 4, 55, 58, 59, 62, 63, 64, 78, 80, 82, 139

[94] KATZ, J., AND YUNG, M. Scalable protocols for authenticated group key exchange. In *Advances in Cryptology - CRYPTO 2003*, D. Boneh, Ed., vol. 2729 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2003, pp. 110–125. doi:10.1007/978-3-540-45146-4_7. 106

[95] KHOLIDY, H., AND BAIARDI, F. Cids: A framework for intrusion detection in cloud systems. In *Information Technology: New Generations (ITNG), 2012 Ninth International Conference on* (April 2012), pp. 379–385. doi:10.1109/ITNG.2012.94. 36, 38, 46, 47, 138

[96] KHREICH, W. *Towards Adaptive Anomaly Detection Systems using Boolean Combination of Hidden Markov Models*. Ph.D. thesis, Ecole De Technologie Superieure, Universite Du Quebec, Canada, 2011. 90

[97] KHREICH, W., GRANGER, E., SABOURIN, R., AND MIRI, A. Combining hidden markov models for improved anomaly detection. In *Communications, 2009. ICC '09. IEEE International Conference on* (june 2009), pp. 1 –6. doi:10.1109/ICC.2009.5198832. 90

[98] KIM, Y., PERRIG, A., AND TSUDIK, G. Simple and fault-tolerant key agreement for dynamic collaborative groups. In *Proceedings of the 7th ACM Conference on Computer and Communications Security* (New York, NY, USA, 2000), CCS '00, ACM, pp. 235–244. doi:10.1145/352600.352638. 106

[99] KORAMBATH, P., AND NARCIS, M. Investigating private cloud storage deployment using cumulus, walrus, and openstack/swift. Tech. rep., Institute for Digital Research and Education (IDRE), 2012. 66

[100] KOSORESOW, A. P., AND HOFMEYR, S. A. Intrusion detection via system call traces. *IEEE Software 14* (1997), 35–42. 61

[101] KOURAI, K., AND CHIBA, S. Hyperspector: Virtual distributed monitoring environments for secure intrusion detection. In *Proceedings of the 1st ACM/USENIX International Conference on Virtual Execution Environments* (New York, NY, USA, 2005), VEE '05, ACM, pp. 197–207. doi:10.1145/1064979.1065006. 42

[102] KRUTZ, R., AND VINES, R. *Cloud Security: A Comprehensive Guide to Secure Cloud Computing*. Wiley, 2010. 8, 16, 17, 18, 27, 29, 31, 36, 37, 56, 137

[103] KUHR, S. *Implementation of a JPEG Decoder for a 16-bit Microcontroller*. Ph.D. thesis, University of applied sciences, Stuttgart, Germany, 2001. 114, 118

[104] LAUREANO, M., MAZIERO, C., AND JAMHOUR, E. Intrusion detection in virtual machine environments. In *Proceedings of the 30th EUROMICRO Conference* (Washington, DC, USA, 2004), EUROMICRO '04, IEEE Computer Society, pp. 520–525. doi:10.1109/EUROMICRO.2004.48. 38, 42

[105] LEE, W., AND STOLFO, S. J. Data mining approaches for intrusion detection. In *Proceedings of the 7th Conference on USENIX Security Symposium - Volume 7* (Berkeley, CA, USA, 1998), SSYM'98, USENIX Association, pp. 6–6. 58

[106] LEE, W., STOLFO, S. J., AND CHAN, P. K. Learning patterns from unix process execution traces for intrusion detection. In *In AAAI Workshop on AI Approaches to Fraud Detection and Risk Management* (1997), AAAI Press, pp. 50–56. 61

[107] LI, T., WANG, G., AND DENG, R. H. Security analysis on a family of ultra-lightweight rfid authentication protocols. *JSW 3*, 3 (2008), 1–10. 105

[108] LINN, C. M., RAJAGOPALAN, M., BAKER, S., COLLBERG, C., DEBRAY, S. K., AND HARTMAN, J. H. Protecting Against Unexpected System Calls. In *Proceedings of the 14th conference on USENIX Security Symposium* (Berkeley, CA, USA, 2005), p. 16. 92

[109] LIPPMANN, R. P., CUNNINGHAM, R. K., FRIED, D. J., GRAF, I., KENDALL, K. R., WEBSTER, S. E., AND ZISSMA, M. A. Results of the DARPA 1998 Offline Intrusion Detection Evaluation. *MIT Lincoln Laborator* (1998). 63

[110] LITTY, L. *Architectural Introspection and Applications*. University of Toronto, 2010. 39

[111] LITTY, L., LAGAR-CAVILLA, H. A., AND LIE, D. Hypervisor support for identifying covertly executing binaries. In *Proceedings of the 17th Conference on Security Symposium* (Berkeley, CA, USA, 2008), SS'08, USENIX Association, pp. 243–258. 39

[112] LIU, A., CHEN, J., AND YANG, L. Real-time detection of covert channels in highly virtualized environments. In *Critical Infrastructure Protection* (2011), pp. 151–164. 51, 113

[113] LOGICMONITOR, I. Logicmonitor, Feb. 2014. 29

[114] LUO, Q., Ed. *Advances in Wireless Networks and Information Systems*. Lecture Notes in Electrical Engineering. doi:10.1007/978-3-642-14350-2. 117

[115] MAIERO, C., AND MICULAN, M. Unobservable intrusion detection based on call traces in paravirtualized systems. In *Proc. SECRYPT* (2011), J. Lopez and P. Samarati, Eds., SciTePress. 38

[116] MAN PAGE, L. strace(1). Accessed: 2013-09-30. 73, 74

[117] MANAVI, S. Secure Model for Virtualization Layer in Cloud Infrastructure. *International Journal of Cyber-Security and Digital Forensics (IJCSDF) 1*, 1 (2012). 19

[118] MATHER, T., KUMARASWAMY, S., AND LATIF, S. *Cloud Security and Privacy: An Enterprise Perspective on Risks and Compliance*. O'Reilly Media, Inc., 2009. 21

[119] MAZZARIELLO, C., BIFULCO, R., AND CANONICO, R. Integrating a network ids into an open source cloud computing environment. In *Information Assurance and Security (IAS), 2010 Sixth International Conference on* (Aug 2010), pp. 265–270. doi:10.1109/ISIAS.2010.5604069. 44

[120] MCMILLAN, R. Cloud Computing a 'security Nightmare,' Says Cisco CEO. *PCWorld* (Apr. 2009). Accessed: 2014-01-06. 26

[121] MELL, P., AND GRANCE, T. Effectively and Securely Using the Cloud Computing Paradigm, May 2009. 27

[122] MELL, P. M., AND GRANCE, T. Sp 800-145. the nist definition of cloud computing. Tech. rep., National Institute of Standards & Technology, Gaithersburg, MD, United States, 2011. 1, 7, 8, 12, 13, 16, 135

[123] MILENKOSKI, A., IOSUP, A., KOUNEV, S., SACHS, K., RYGIELSKI, P., DING, J., CIRNE, W., AND ROSENBERG, F. Cloud Usage Patterns: A Formalism for Description of Cloud Usage Scenarios. Tech. Rep. SPEC-RG-2013-001 v.1.0.1, SPEC Research Group - Cloud Working Group, Standard Performance Evaluation Corporation (SPEC), April 2013. 13, 14

[124] MIRAJKAR, S., AND BIRADAR, S. Security aspects of multi-cloud computing in modern computing environment. *IOSR Journal of Computer Science 5* (2014), 38–41. 32

[125] MOHAN, A., AND S, S. Survey on live vm migration techniques. *International Journal of Advanced Research in Computer Engineering and Technology 2* (2013). 95, 104

[126] MOLNAR, D., AND SCHECHTER, S. E. Self hosting vs. cloud hosting: Accounting for the security impact of hosting in the cloud. In *WEIS* (2010). 29

[127] MUEEN, A., NATH, S., AND LIU, J. Fast approximate correlation for massive time-series data. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2010), SIGMOD '10, ACM, pp. 171–182. doi:10.1145/1807167.1807188. 114, 119

[128] MUTZ, D., VALEUR, F., VIGNA, G., AND KRUEGEL, C. Anomalous system call detection. *ACM Trans. Inf. Syst. Secur. 9*, 1 (Feb. 2006), 61–93. doi:10.1145/1127345.1127348. viii, 3, 23, 55, 58, 59, 62, 72, 92

[129] O'LEARY, D. E. Knowledge management in enterprise resource planning systems. In *AAAI Technical Report WS-99-10* (1999). 71

[130] ONOUE, K., OYAMA, Y., AND YONEZAWA, A. Control of system calls from outside of virtual machines. In *Proceedings of the 2008 ACM Symposium on Applied Computing* (New York, NY, USA, 2008), SAC '08, ACM, pp. 2116–1221. doi:10.1145/1363686.1364196. 48

[131] PAYNE, B., CARBONE, M., SHARIF, M., AND LEE, W. Lares: An architecture for secure active monitoring using virtualization. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on* (2008), pp. 233–247. doi:10.1109/SP.2008.24. 41

[132] PAYNE, B. D. *Improving Host-Based Computer Security Using Secure Active Monitoring and Memory Analysis*. Ph.D. thesis, Georgia Institute of Technology, 2010. 31, 41

[133] PROJECT, O. Opennebula. 47

[134] QAYYUM, A., ISLAM, M., AND JAMIL, M. Taxonomy of statistical based anomaly detection techniques for intrusion detection. In *Emerging Technologies, 2005. Proceedings of the IEEE Symposium on* (Sept 2005), pp. 270–276. doi:10.1109/ICET.2005.1558893. 24

[135] QIU, L., ZHANG, Y., WANG, F., KYUNG, M., AND MAHAJAN, H. R. Trusted computer system evaluation criteria. In *National Computer Security Center* (1985). 101

[136] RAJ, H., NATHUJI, R., SINGH, A., AND ENGLAND, P. Resource management for isolation enhanced cloud services. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security* (New York, NY, USA, 2009), CCSW '09, ACM, pp. 77–84. doi:10.1145/1655008.1655019. 113

[137] REUBEN, J. S. A survey on virtual machine security. *Helsinki University of Technology* (2007). 31

[138] RHOTON, J. *Cloud Computing Explained: implementation handbook for enterprises*. Recursive Limited, 2009. 8, 19, 21, 30, 135

[139] RISTENPART, T., TROMER, E., SHACHAM, H., AND SAVAGE, S. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security* (New York, NY, USA, 2009), CCS '09, ACM, pp. 199–212. doi:10.1145/1653662.1653687. 48, 49, 50, 95, 98, 99, 101, 102, 103, 114, 128, 145

[140] ROSCHKE, S., CHENG, F., AND MEINEL, C. Intrusion detection in the cloud. In *Dependable, Autonomic and Secure Computing, 2009. DASC '09. Eighth IEEE International Conference on* (Dec 2009), pp. 729–734. doi:10.1109/DASC.2009.94. 43

[141] ROSENBERG, J., SCHULZRINNE, H., CAMARILLO, G., JOHNSTON, A., PETERSON, J., SPARKS, R., HANDLEY, M., AND SCHOOLER, E. SIP: Session Initiation Protocol. RFC 3261, RFC Editor, June 2002. 44

[142] SALESFORCE.COM, INC. Salesforce.com Announces Force.com for Amazon Web Services, Extending the Benefits of the Cloud to Even More Businesses, 2014. 13

[143] SALESFORCE.INC. Salesforce, 2014. 12

[144] SAMMY, K., SHENGBING, R., AND WILSON, C. Energy efficient security preserving vm live migration in data centers for cloud computing. *IJCSI International Journal of Computer Science Issues 9*, 3 (Mar. 2012), 33–39. 122, 123, 124

[145] SAXENA, A., AND A.K.SHARMA. An agent based distributed security system for intrusion detection in computer networks. *International Journal of Computer Applications 12*, 3 (December 2010), 18–27. Published By Foundation of Computer Science. 36

[146] SCARFONE, K. A., GRANCE, T., AND MASONE, K. Sp 800-61 rev. 1. computer security incident handling guide. Tech. rep., National Institute of Standards & Technology, Gaithersburg, MD, United States, 2008. 22, 23

[147] SEKAR, R., BENDRE, M., DHURJATI, D., AND BOLLINENI, P. A fast automaton-based method for detecting anomalous program behaviors. In *Security and Privacy, 2001. S P 2001. Proceedings. 2001 IEEE Symposium on* (2001), pp. 144–155. doi:10.1109/SECPRI.2001.924295. 92

[148] SHELKE, P. K., SONTAKKE, S., AND GAWANDE, A. D. Intrusion detection system for cloud computing. *International Journal of Scientific and Technology Research 1* (2012). 8, 20, 31, 33, 34, 38, 45, 135

[149] SINGH, A., AND KINGER, S. Virtual machine migration policies in clouds. In *International Journal of Science and Research (IJSR)* (2013), vol. 2, International Journal of Science and Research (IJSR), pp. 364–367. 124

[150] SOMAYAJI, A., AND FORREST, S. Automated response using system-call delays. In *Proceedings of the 9th Conference on USENIX Security Symposium - Volume 9* (Berkeley, CA, USA, 2000), SSYM'00, USENIX Association, pp. 14–14. 61, 62

[151] SOO KIM, S. *Real-time analysis of aggregate network traffic for anomaly detection*. Ph.D. thesis, Texas A and M University, 2005. 117

[152] SQALLI, M., AL-HAIDARI, F., AND SALAH, K. Edos-shield - a two-steps mitigation technique against edos attacks in cloud computing. In *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on* (2011), pp. 49–56. doi:10.1109/UCC.2011.17. 34, 49, 136

[153] STALLINGS, W. *Data and Computer Communications*. Alternative eText Formats Series. Pearson/Prentice Hall, 2007. 109

[154] STEINER, M., TSUDIK, G., AND WAIDNER, M. Key Agreement in Dynamic Peer Groups. *IEEE Transactions on Parallel and Distributed Systems 11*, 8 (Aug. 2000), 769–780. doi:10.1109/71.877936. 106

[155] SULTANA, A., HAMOU-LHADJ, A., AND COUTURE, M. An improved hidden markov model for anomaly detection using frequent common patterns. In *Communications (ICC), 2012 IEEE International Conference on* (June 2012), pp. 1113–1117. doi:10.1109/ICC.2012.6364527. 85

[156] TAKABI, H., JOSHI, J., AND AHN, G.-J. Security and privacy challenges in cloud computing environments. *Security Privacy, IEEE 8*, 6 (2010), 24–31. doi:10.1109/MSP.2010.186. 28

[157] TANDON, G., AND CHAN, P. K. On the learning of system call attributes for host based anomaly detection. *International Journal on Artificial Intelligence Tools 15*, 6 (2006), 875–892. 58, 63, 92

[158] TEAM, K. Kernel based virtual machine. 65

[159] TEAM, T. A. Summary of the october 22, 2012 aws service event in the us-east region. Accessed: 2014-01-19. 27

[160] TEAM, T. A. Summary of the amazon ec2 and amazon rds service disruption in the us east region, Apr. 2011. 28

[161] TEAM, T. A. S. Amazon s3 availability event, July 2008. 28

[162] TRYTON. Tryton web site. Accessed: 2014-01-26. 71

[163] VAQUERO, L. M., RODERO-MERINO, L., AND MORáN, D. Locking the sky: A survey on iaas cloud security. *Computing 91*, 1 (Jan. 2011), 93–118. doi:10.1007/s00607-010-0140-x. 27, 28, 29, 31

[164] VARADARAJAN, V., KOOBURAT, T., FARLEY, B., RISTENPART, T., AND SWIFT, M. M. Resource-freeing attacks: improve your cloud performance (at your neighbor's expense). In *Proceedings of the 2012 ACM conference on Computer and communications security* (New York, NY, USA, 2012), CCS '12, ACM, pp. 281–292. doi:10.1145/2382196.2382228. 20, 35, 48, 50, 51, 95, 98, 136

[165] VIEGA, J. Cloud computing and the common man. *Computer 42*, 8 (Aug 2009), 106–108. doi:10.1109/MC.2009.252. 27

[166] VIEIRA, K., SCHULTER, A., WESTPHALL, C., AND WESTPHALL, C. Intrusion detection for grid and cloud computing. *IT Professional 12*, 4 (July 2010), 38–43. doi:10.1109/MITP.2009.89. 38, 135

[167] VIVINSANDAR, S., AND SHENAI, S. Article: Economic denial of sustainability (edos) in cloud services using http and xml based ddos attacks. *International Journal of Computer Applications 41*, 20 (March 2012), 11–16. Published by Foundation of Computer Science, New York, USA. 20, 34, 48, 49, 136

[168] VOORSLUYS, W., BROBERG, J., VENUGOPAL, S., AND BUYYA, R. Cost of virtual machine live migration in clouds: A performance evaluation. In *Proceedings of the 1st International Conference on Cloud Computing* (Berlin, Heidelberg, 2009), CloudCom '09, Springer-Verlag, pp. 254–265. doi:10.1007/978-3-642-10665-1_23. 122

[169] WAGNER, D., AND SOTO, P. Mimicry attacks on host-based intrusion detection systems. In *Proceeding CCS '02 Proceedings of the 9th ACM conference on Computer and communications security* (New York, NY, USA, 2001), pp. 255–264. doi:10.1145/586110.586145. 92, 144

[170] WARRENDER, C., FORREST, S., AND PEARLMUTTER, B. Detecting intrusions using system calls: alternative data models. In *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on* (1999), pp. 133–145. doi:10.1109/SECPRI.1999.766910. 55, 63, 76, 85

[171] WEB SERVICES, A. Amazon web services: Overview of security processes, 2013. 70

[172] WEB SERVICES, A. Amazon cloudwatch, Feb. 2014. 29

[173] WU, Z., XU, Z., AND WANG, H. Whispers in the hyper-space: High-speed covert channel attacks in the cloud. In *Proceedings of the 21st USENIX Conference on Security Symposium* (Berkeley, CA, USA, 2012), Security'12, USENIX Association, pp. 9–9. 51, 96, 101, 102

[174] XIANG, G., JIN, H., ZOU, D., ZHANG, X., WEN, S., AND ZHAO, F. Vmdriver: A driver-based monitoring mechanism for virtualization. In *Reliable Distributed Systems, 2010 29th IEEE Symposium on* (Oct 2010), pp. 72–81. doi:10.1109/SRDS.2010.38. 56, 91, 92

[175] XU, G., DING, Y., XU, X., DONG, Y., ZHAO, J., AND DONG, Y. A heuristic location selection strategy of virtual machine based on the residual load factor. *Journal of Computational Information System 9*, 18 (Sept. 2013), 7389–7396. doi:10.12733/jcisP0166. 123, 124

[176] XU, Y., BAILEY, M., JAHANIAN, F., JOSHI, K., HILTUNEN, M., AND SCHLICHTING, R. An exploration of l2 cache covert channels in virtualized environments. In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop* (New York, NY, USA, 2011), CCSW '11, ACM, pp. 29–40. doi:10.1145/2046660.2046670. 51, 102

[177] YAN, W., AND ANSARI, N. Anti-virus in-the-cloud service: are we ready for the security evolution? *Sec. and Commun. Netw. 5*, 6 (June 2012), 572–582. doi:10.1002/sec.352. 49

[178] YANG, S.-F., CHEN, W.-Y., AND WANG, Y.-T. Icas: An inter-vm ids log cloud analysis system. In *Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on* (Sept 2011), pp. 285–289. doi:10.1109/CCIS.2011.6045076. 28, 44

[179] YLONEN, T., AND LONVICK, C. The Secure Shell (SSH) Transport Layer Protocol. RFC 4253 (Proposed Standard), January 2006. 69

[180] YU, S. Ddos attack and defence in cloud. In *Distributed Denial of Service Attack and Defense*, SpringerBriefs in Computer Science. Springer New York, 2014, pp. 77–93. doi:10.1007/978-1-4614-9491-1_5. 20, 35, 48

[181] ZHANG, S. Deep-diving into an easily-overlooked threat: Inter-vm attacks. Tech. rep., Kansas State University, 2013. 2, 20, 28, 30, 31

[182] ZHANG, X., SHAE, Z.-Y., ZHENG, S., AND JAMJOOM, H. Virtual machine migration in an over-committed cloud. In *Network Operations and Management Symposium (NOMS), 2012 IEEE* (April 2012), pp. 196–203. doi:10.1109/NOMS.2012.6211899. 5, 96

[183] ZHANG, Y., JUELS, A., OPREA, A., AND REITER, M. K. Homealone: Co-residency detection in the cloud via side-channel analysis. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2011), SP '11, IEEE Computer Society, pp. 313–328. doi:10.1109/SP.2011.31. 48, 51, 95

[184] ZHANG, Y., JUELS, A., REITER, M. K., AND RISTENPART, T. Cross-vm side channels and their use to extract private keys. In *Proceedings of the 2012 ACM conference on Computer and communications security* (New York, NY, USA, 2012), CCS '12, ACM, pp. 305–316. doi:10.1145/2382196.2382230. 48, 51, 98

[185] ZHANG, Y., LI, M., BAI, K., YU, M., AND ZANG, W. Incentive compatible moving target defense against vm-colocation attacks in clouds. In *Information Security and Privacy Research*, D. Gritzalis, S. Furnell, and M. Theoharidou, Eds., vol. 376 of *IFIP Advances in Information and Communication Technology*. Springer Berlin Heidelberg, 2012, pp. 388–399. doi:10.1007/978-3-642-30436-1_32. 113

[186] ZHAO, J., GU, D., AND LI, Y. An efficient fault-tolerant group key agreement protocol. *Computer Communications 33*, 7 (2010), 890 – 895. doi:http://dx.doi.org/10.1016/j.comcom.2010.01.001. 106

[187] ZHOU, F., GOEL, M., DESNOYERS, P., AND SUNDARAM, R. Scheduler vulnerabilities and attacks in cloud computing. *CoRR abs/1103.0759* (2011). 48, 49

[188] ZHOU, M. *Network Intrusion Detection: Monitoring, Simulation and Visualization*. Ph.D. thesis, School of Computer Science in the College of Engineering and Computer Science at the University of Central Florida, Orlando, FL, USA, 2005. AAI3188144. vii, 35, 36

[189] ZHOU, Y., AND FENG, D. Side-channel attacks: Ten years after its publication and the impacts on cryptographic module security testing. *IACR Cryptology ePrint Archive 2005* (2005), 388. 49

[190] ZOU, D., ZHANG, W., QIANG, W., XIANG, G., YANG, L. T., JIN, H., AND HU, K. Design and implementation of a trusted monitoring framework for cloud platforms. *Future Gener. Comput. Syst. 29*, 8 (Oct. 2013), 2092–2102. doi:10.1016/j.future.2012.12.020. 20, 29

[191] ZOU, D., ZHANG, W., QIANG, W., XIANG, G., YANG, L. T., JIN, H., AND HU, K. Design and implementation of a trusted monitoring framework for cloud platforms. *Future Gener. Comput. Syst. 29*, 8 (Oct. 2013), 2092–2102. doi:10.1016/j.future.2012.12.020. 21, 29, 30, 33, 47, 56, 92, 136