Enhancing Security in Public IaaS Cloud Systems through VM Monitoring

A Consumer's Perspective

Taimur Sultan Said Al Said

School of Computer Science and Informatics

Cardiff University

A thesis submitted in partial fulfilment of the requirement for the degree of

Doctor of Philosophy

December 2016

Declaration

This work has not previously been accepted in substance for any degree and is not concurrently submitted in candidature for any degree.

Signed ..... (candidate)

Date .....

Statement 1

This thesis is being submitted in partial fulfilment of the requirements for the degree of PhD.

Signed ..... (candidate)

Date .....

Statement 2

This thesis is the result of my own independent work/investigation, except where otherwise stated. Other sources are acknowledged by explicit references.

Signed ..... (candidate)

Date .....

Statement 3

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed ..... (candidate)

Date .....

Taimur Sultan Said Al Said
December 2016

To my mother ...
To my father ...
To Oman ...

Abstract

Cloud computing is attractive for both consumers and providers to benefit from potential economies of scale in reducing cost of use (for consumers) and operation of infrastructure (for providers). In the IaaS service deployment model of the cloud, consumers can launch their own virtual machines (VMs) on an infrastructure made available by a cloud provider, enabling a number of different applications to be hosted within the VM. The cloud provider generally has full control and access to the VM, providing the potential for a provider to access both VM configuration parameters and the hosted data. Trust between the consumer and the provider is key in this context, and generally assumed to exist. However, relying on this assumption alone can be limiting. We argue that the VM owner must have greater access to operations that are being carried out on their VM by the provider and greater visibility on how this VM and its data are stored and processed in the cloud. In the case where VMs are migrated by the provider to another region, without notifying the owner, this can raise some privacy concerns. Therefore, mechanisms must be in place to ensure that violation of the confidentiality, integrity and SLA does not happen. In this thesis, we present a number of contributions in the field of cloud security which aim at supporting trustworthy cloud computing. We propose monitoring of security-related VM events as a solution to some of the cloud security challenges. Therefore, we present a system design and architecture to monitor security-related VM events in public IaaS cloud systems. To enable the system to achieve focused monitoring, we propose a taxonomy of security-related VM events. The architecture was supported by a prototype implementation of the monitoring tool called: *VMInformant*, which keeps the user informed and alerted about various events that have taken place on their VM. The tool was evaluated to learn about the performance and storage overheads associated with monitoring such events using CPU and I/O intensive benchmarks. Since events in multiple VMs, belonging to the same owner, may be related, we suggested an architecture of a system, called: *Inspector Station*, to aggregate and analyse events from multiple VMs. This system enables the consumer: (1) to learn about the overall security status of multiple VMs; (2) to find patterns in the events; and (3) to make informed decisions related to security.

To ensure that VMs are not migrated to another region without notifying the owner, we proposed a hybrid approach, which combines multiple metrics to estimate the likelihood of a migration event. The technical aspects in this thesis are backed up by practical experiments to evaluate the approaches in real public IaaS cloud systems, e.g. Amazon AWS and Google Cloud Platform. We argue that having this level of transparency is essential to improve the trust between a cloud consumer and provider, especially in the context of a public cloud system.

Acknowledgements

Doing PhD has been a remarkable and life-changing experience for me, one that has taught me a lot, and which would not have been possible without the tremendous support of many people.

I would like, first, to thank my supervisor, Professor Omer Farouq Rana, for the support, encouragement and kindness, without which, this PhD journey would not have been achievable. Thank you for your mentoring, patience, trust, and for the precious advice given to me during the stages of the PhD. I have learned a lot from him, especially how to become an independent researcher who is able to justify ideas and explore techniques without fear or hesitation. Thank you for keeping me on track, when my imagination takes me towards non-realistic plans. I feel that I am blessed for being under his supervision.

Many thanks are due to Dr. Peter Burnap, my second supervisor, for motivating me, and enlightening me with ideas. I am also grateful to the School of Computer Science and Informatics for giving me the chance to participate in teaching, giving seminars and guest lectures; this has really shaped my practical and presentation skills. Special thanks are also due to Dr. Rob Davies, Robert Evans, Laurence Semmens and Mrs. Helen Williams for technical and administrative support; they were always there when I needed them.

My deepest gratitude to His Majesty, Sultan Qaboos bin Said, the Sultan of Oman, for granting us this scholarship. You have always been a great leader, supporter and motivator for your nation. We have learned the hard work from you, and only through hard work; we will continue to excel towards the best for Oman.

Thanks are also due to the Omani Cultural attaché in the UK for their continuous support. Special thanks to Mrs. Yusra Al-Ismali for her support and kindness. Many thanks to the Ministry of Higher Education in Oman, for the administrative support

to manage this scholarship. Especially, I would like to acknowledge Nawal Al-Dehani, for her support and prompt responses.

The journey of the PhD would have been more difficult without the presence of my lovely colleagues, who were there when I needed them. When it got difficult, sitting with them in the common room was a blessing, especially with all the jokes and laughs, accepting my "mental models" and "tricks", such precious moments. I would like to thank my lovely friends and colleagues: Nasser, Dr. Hanaa, Fatma, Liqaa, Dr. Shadha, Dr. Christos. P, Dr. Yulia, Matthew. N, Nyala, Beryl, Akis, Amir.J, Saad, Irene, Khalid, Soha, Ali. Dr. Mona, Dr. Haya, Pete, Ashwan, Shelan, Narmine, Aseela, Shahd, Ibrahim, Liam, Louise, Lowri, Jon Slade. Thanks to my colleagues from other universities: Shadha Al Amri, Haifa Al-Nasseri, Nasser Al Hosni and Abdullah Al-Balushi. Special thanks to Dr. Suaad Al Arifi, Dr. Ryan Chard and Dr. Django Armstrong, whom I had the chance to chat with, regarding my work.

I would like also to thank the anonymous reviewers, whose comments helped us to improve our research work and enhance it before publication.

Thanks to my wife for her support and patience, for my lovely daughters: Mayyah, Muna and Amjaad for being strong and patient when I was away from them. Thanks to my mother and father, brothers and sisters, and all my family members and friends, who coped with my absence during the PhD journey. Hopefully, they will all be proud of me.

Thanks also to those who discouraged me from doing the PhD; you only made me more determined to go for it!
Most of all, thanks to Allah Almighty for guiding me though this journey and blessing me with all the good things in this life.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Overview

Information technology has seen a tremendous dependency on the Internet especially with the introduction of social networking services and the demand for media sharing or storage. This has led to the availability of 'Cloud Computing'. While the concept of cloud computing is not exactly new, it has been given much attention in recent years due to its promised benefits and the various deployment/delivery mechanisms it supports. Cloud computing is a concept which promotes the economic use of resources among clients by employing the *pay-per-use* model; such that clients will pay only for the resources they use. There are many companies nowadays which offer cloud services of different flavours. Amazon Web Services (AWS) is an example of one of the cloud providers which offers considerably cheap compute and storage services through its AWS suite[1]. The public cloud services market was forecast to grow 17.2 percent in 2016 to a total of 208.6 billion dollars worldwide [53]. According to a survey by Rightscale, which covered a large group of organisations, about 89% of the respondents were using public clouds [119].

From a computational resource provider perspective, most of the benefits achieved in cloud computing have been due to increasing economies of scale. These benefits relate to improvements in offered performance, reduction of potential cost for offering resources (according to a variety of different economic models) and reduced energy costs. These economies of scale have been realized through resource provisioning in data centres, which provide a variety of resource scheduling and management strategies

[1] Amazon Web Services http://aws.amazon.com

to improve resource utilization and reduce potential downtime. On the other hand, consumers have realized that the pay-per-use business model of cloud computing enables them to grow/shrink their computational resources dynamically, often based on fluctuations in demand by customers. Although, in practice, such variable resource demand provisioning and outsourcing may not be cost effective (in some instances, running resources in-house may be less costly for a consumer), the ability to not manage and support resources is often attractive for many small companies. Consequently, many organisations have continued to adopt the public cloud business model accepting the fact that they do not have control over their data any more. For example, many organisations have shifted their email services to public cloud providers to cut the cost of the physical servers, outsource the storage administration and reduce maintenance overheads.

The cost effectiveness of cloud computing (for both providers and consumers) is often due to the multi-tenancy nature of such systems, based on sharing of resources among different consumers. In practice, this implies that a consumer's data could reside on the same hosting infrastructure as his competitor. Most cloud providers use a resource management software to isolate one customer from another and ensure that one user is unable to see data from another. The hypervisor is the fundamental component within such a management system which is used to manage virtual machines and achieve the required isolation between them [19]. The hypervisor is therefore a component trusted by both the provider and the customer to: (i) efficiently manage the underlying VMs, ensuring for instance, that there is no VM starvation and all VMs get access to the physical infrastructure; (ii) ensure VM data and processes are isolated from each other, so that a fault in one VM does not negatively impact the operation of another, and data in one VM cannot be accessed through another. In general, performance and security remain two fundamental trusted properties of a hypervisor, without which many users would be unwilling to outsource their services to public clouds.

The above mentioned cloud business model and the current trends in public cloud deployment strategies present a number of dimensions which motivate our research focus:

- ***Cloud provider's full control:*** the cloud provider generally has full access to a consumer's data. Even though trust is always assumed and a Service Layer Agreement (SLA) is agreed between the consumer and provider – it is possible for

either faults to arise or for a provider to proactively view consumer data content – leading to a violation of terms in the SLA. For example, employees at a cloud provider could steal confidential data belonging to a consumer or manipulate configuration parameters of the consumer VM [109]. Stealing or tampering with the data of consumers could be driven by different motives, especially if there is a conflict of interest between various cloud consumers or between the provider and the consumer. The full control imposed by the cloud provider could also be utilised by governments where power can be exerted to force the provider to disclose confidential information about the consumer [7, 152]. This may lead to a shut down of virtual servers, copying VMs dynamic memory, tampering with data, etc. For example, in 2010, Amazon was forced to terminate all operations of the WikiLeaks website which was hosted in AWS data centers [152]. In some countries, this is backed up by laws. For example, in the US, the Patriot Act gives the U.S. government the right to access hosted data [20].

- ***Lack of transparency:*** services provisioned by some public cloud providers are at the expense of transparency. Consumers are provided with limited details about who has accessed their data, and how and where their data is located within the cloud. In a survey carried out by Fujitsu Research Institute, it was found that 88% of cloud consumers were worried about who could have access to their data [72], even though the provider explicitly offered privacy protection to consumers. Also, due to the virtual nature of servers in the cloud, the VM could get migrated to another (jurisdictional) region/ availability zone (in real time) other than the region where the VM was initially deployed – primarily as means to allow the cloud provider to load balance requests across multiple physical data centres or to perform host maintenance, e.g. as in the case of Google [58]. This has often been identified as being of benefit for both the consumer and the cloud provider, in that the consumer does not see any interruption in service when the migration takes place, and for the cloud provider to benefit from reduced operational costs. However, such automated migration could also pose security concerns, especially when the consumers are not informed about the migration event. Hiding the migration event from consumers could introduce many risks and issues that include (but are not limited to): (1) VM theft, where the VM could have been migrated to a non-disclosed location as a consequence of attacking the hypervisor of the cloud provider; (2) Violation of SLA, where the consumer identifies a particular region to host his data.

- ***Increased use of public cloud services to host mission-critical applications:*** Deploying services to cloud systems has been appealing to many organisations. According to Gartner[2], there is an apparent trend of moving mission-critical applications to the cloud [52]. In 2012, it was predicted by Gartner that 50 percent of companies will have mission-critical data in the cloud by 2016. This trend of moving mission-critical applications to the cloud, combined with the lack of transparency and provider's control, has brought to light many potential privacy and security implications.

- ***Increased number of attack vectors affecting cloud services:*** In a report by Alert Logic[3], research shows that there is an increase in *attack frequency* on organisations that store their infrastructure in the cloud [89] as a result of the increased trend of migrating data and applications to the cloud. It was demonstrated by researchers that private keys of a VM could be reconstructed from another VM running on the same physical host [165]. Although they demonstrated the attack in a laboratory setting, the same threat could be faced in public IaaS cloud system, which means that another tenant in the cloud could cause harm to the consumers data. Enabling a consumer to encrypt their data provides one possible solution to this. However, for data to be processed at the public cloud, it is necessary to decrypt it within the VM. Techniques for performing operations directly on the encrypted data, making use of Homomorphic Encryption [55] (for instance), are still not mature enough to handle general-purpose operations or have suitable performance to be used in practice. Some attacks such as: *bluepill* [126] showed that it may be possible for hypervisors to be under the full control of malicious rootkits– which means that all VMs and data can be compromised.

- ***Cost of monitoring cloud services:*** the cost of monitoring VM instances hosted in public IaaS cloud systems using SaaS can be high. Most cloud providers offer monitoring of resource usage as a service, e.g. Cloudwatch[4] by AWS. These monitoring solutions focus on resource and performance monitoring. No particular focus on security is provided. Monitoring cost is calculated by the number of virtual servers, which means that over time the cost can be considerably high.

[2]http://www.gartner.com/
[3]Alert Logic https://www.alertlogic.com/
[4]https://aws.amazon.com/cloudwatch/

Since monitoring requires analysing logs in the VMs, this can consume CPU resources. Achieving focused monitoring may be the solution to this.

Throughout the thesis, we refer to these virtual servers that hold importance to the customer as: *Critical VMs*; any VM that is very sensitive to the customer due to the tasks it performs or the data it processes or generates. To overcome the concerns highlighted above, it is necessary to keep the owner of a critical VM informed about what has happened to their VM once it has been deployed within a public cloud system. It is therefore not enough for a provider to offer guarantees about data and VM privacy or pre-negotiate an SLA with a consumer. A more proactive process is needed to ensure that the consumer is kept aware of operations that are carried out on their deployed VM. There are a lack of tools which focus on monitoring security-related events within VMs, including when a migration of a VM occurs by the cloud provider.

In this thesis, we utilize *monitoring* of security-related VM events as a means to enhance the trust between the cloud consumer and the provider. We present the architecture and implementation of an extended VM that enables recording of events which occur within a VM once it has been deployed. The consumer/ VM owner is therefore able to get better visibility on how their data is being accessed or processed, thereby giving them greater potential trust that their VM or data has not been compromised. The system stores the events in an encrypted area so that only the owner can access the data. It allows the consumer to choose the level of detail at which recording of events is needed in order to minimise the overheads on performance and storage due to the limited resources of the VM. We propose a taxonomy of security-related VM events used to enable focused monitoring to ensure that monitoring does not affect the performance of the running applications in the VM. We present a detailed evaluation of the system using some software benchmarks, including *Povray* and *bzip2*. The chosen benchmarks are I/O and CPU intensive; to allow measuring the performance and storage overheads caused by monitoring. This is highlighted in Chapter 4.

To enable the cloud consumer to monitor, analyse and correlate the captured events from *multiple* monitored VMs, we propose an architecture of a central server which is called: "Inspector Station". The architecture supports receiving events, categorizing and analysing them for the purpose of generating reports which can help the consumer make informed decisions and observe the health (state) of their VM. It applies an algorithm to find patterns in collected events across multiple VMs, and to detect the

migration of monitored VM instances. This is highlighted in Chapter 6.

To detect the migration of virtual machines in public IaaS cloud, we propose a VM migration detection approach which can give an estimate of a potential migration. Instead of relying only on one metric, e.g. measuring ICMP Ping latencies, which is commonly used to check proximity to servers, we use several metrics combined together. We call these metrics: *Migration metrics*. Once these metrics are collected for each monitored VM, a profile which determines the "normal" state of the VM will be formed (we call it: *normal profile*). The normal profile contains information about the current state of the VM based on the measured migration metrics. This *normal profile* can then be compared with other profiles of the same VM which are generated *periodically* called: *periodic profiles*. To assist in the detection of VM migration by observing changes in the metrics, we employ a decision function which takes into account the importance of the metrics from the perspective of the consumer. Consumers can use the value returned by the decision function to assist in making the decision as to whether migration has occurred or not. The proposed approaches for detecting migration are highlighted in Chapter 5.

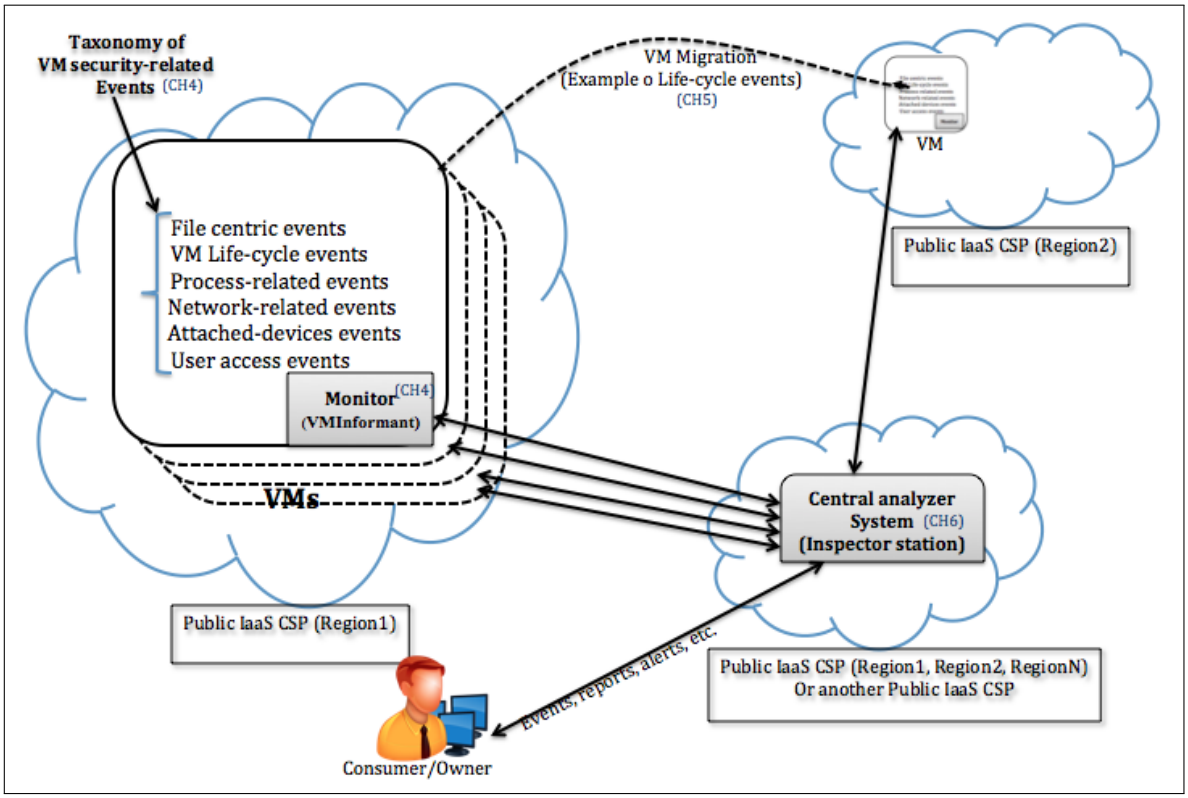

Fig. 1.1 Scope of the problem in the thesis

Figure 1.1 illustrates the scope of the research. It shows that VM security-related events are monitored and detected from inside the VMs hosted in a Public IaaS cloud system. Thus, the approach is from a cloud consumer's perspective. The monitored events are based on a taxonomy that we proposed. It is important to note that the events do not include application-generated events. For example, events generated by an enterprise application such as: SQL server are not among the events to be monitored. Events monitored from multiple VMs are analysed by the consumer in the "Inspector station", which can be located within the regions of the same cloud provider or with another CSP (if this is supported).

1.2 Hypothesis

"If security/privacy events happening inside virtual machines (VMs), hosted in the IaaS cloud infrastructure, are monitored and recorded, a consumer can have greater confidence in what is happening to their VMs. This enhances the level of trust in a cloud provider, and improves the level of security of a consumer's application"

The responsibility of security in the cloud is shared between the cloud provider and the consumer. Putting trust in the provider may be limiting especially if an organisation has critical VMs which run their applications or host their sensitive data. Auditing VM operations by monitoring events and reporting them to the consumer may help support the trust between the cloud provider and the consumer. Section 1.3 highlights the main aim that we establish in order to examine the hypothesis and the objectives which help us to achieve the aim.

1.3 Aim and Objectives

The thesis aims at investigating how security-related VM events can be monitored, recorded and reported to the consumer to support trustworthy cloud computing. Some of the objectives include:

- **Surveying cloud security issues and known attacks:** by understanding issues and concerns underpinning cloud computing in general and IaaS pubic cloud more specifically, we can detect security-specific events and inform the cloud consumer appropriately.
- **Creating a list of the VM-specific events to be monitored:** Because of the state of virtual machines, some of the events to be detected are specific

to them, e.g. when the VM is migrated to another server. Creating a list of VM-specific security-related events can be useful to create dedicated monitoring and protection frameworks.

- **Describing how some of the events can be monitored (technically) from the perspective of the cloud consumer:** Not all events can be detected the same way; especially if we consider control on the IT infrastructure to be with the cloud provider. In addition, detecting and monitoring these events must be from the perspective of a cloud consumer, without relying on input from the cloud provider.

- **Proposing a general modular monitoring system to detect security-related VM events and reporting them to the cloud consumer:** having a modular monitoring architecture will allow adding monitoring services, when needed, to serve the requirements of the consumer.

- **Balancing between monitoring of events and between the performance and storage requirements:** VMs have limited capabilities and their use involves a financial commitment. Consumers use VMs to run their applications. Monitoring events may affect the performance and storage of the VMs but it is important that the effect is reasonable.

1.4 Contributions

This thesis addresses the research hypothesis by providing a set of contributions in the area of cloud security and virtualisation security. Perhaps, devising mechanisms to detect the VM migration event from a consumer's perspective can be considered one of the core contributions presented in this thesis. The reason behind this is that after the automatic migration of running VMs to another host, little is provided to detect or signal the migration event from the consumer's side. The detection is supported by the design of a monitoring tool which run inside VMs and a central analysis unit which manages events generated in multiple VMs. Another core contribution involves the proposal of a taxonomy of VM-specific security-related events. The design of the monitoring tools is based on this taxonomy. The taxonomy is also the basis for performing a focused monitoring in order achieve security and reasonable performance overhead. The rest of this section highlights the contributions in detail as follows:

1. **Detecting security-related VM events:**

- We present a taxonomy of security-related VM events used to enable focused monitoring to ensure that monitoring does not affect the performance of the running applications in the VM. To the best of our knowledge, no such taxonomy that focuses on security-related VM events exists. Most of the existing monitoring approaches provide general monitoring which may not be suitable for virtual machines. Our taxonomy covers VM life-cycle events, e.g. migration events. These groups of events are not normally deemed as security-related. However, we argue that hiding them from the consumer could pose privacy concerns. We believe that this contribution could be useful in designing VM-specific security monitoring systems.
- Based on the proposed taxonomy of security-related VM events, we present the design, architecture and a prototype implementation of a system to detect and monitor security-related VM events. Our system (*VMInformant*) monitors, records, analyses and *informs* the VM owner/consumer of the various discovered security-related events so as to support the decision-making process. The system takes many requirements into consideration such as: ease of use, informative nature, customizability. We believe that the proposed design and architecture could see use in developing security-focused monitoring tools more suitable for VMs.
- To help the owner of VMs draw conclusions about the security status of monitored VMs, we propose the design and architecture of a system to aggregate, manage, analyse and correlate security-related events observed from multiple VMs. Our system (*Inspector Station*) provides reporting and alerting features for the consumer to assist in making trust decisions and to find abnormalities across multiple VMs. We suggest a preliminary pattern finding framework which can help correlate security-related events that occur in multiple VMs. To the best of our knowledge, there is a lack of research in the area of correlating security-related events which happen across multiple VMs in public IaaS cloud systems. Our architecture can serve as a basis for designing bespoke centralized monitoring tools which offer event management and reporting the security state of a large number of VMs.
- We illustrate how some of the state-of-the-art open source tools can be used to aggregate events from multiple VMs in real time and represent them.

2. **Detecting VM migrations:**

- We propose a spatio-temporal taxonomy of migration detection techniques. It considers if migration of VMs occurs within or outside data centers and also whether the detection happens before, during or after migration takes place. We believe that considering this temporal feature when choosing suitable migration detection techniques will help achieve detection with minimized performance and storage overheads. This is because some techniques require that the recording of metrics is performed more frequently inside the VM. This contribution is considered an extension to the work in [46].
- We propose a prototype implementation of a tool to detect migration of VMs in IaaS using an approach called: Virtual Remote Landmark Fingerprint. This approach relies on periodically recording ICMP latencies from the monitored VM to a selected of remote servers. This tool was also used to evaluate the approach and generate the results.
- We developed an algorithm to detect VM migrations in public IaaS cloud systems. The algorithm combines the use of multiple migration detection techniques (referred to as: *migration metrics*), to estimate the likelihood of the migration event. The approach considers the importance of the metrics from the perspective of the cloud consumer. The algorithm uses a weighted decision function to generate the probability of the migration event. The use of the combined approach emphasizes that it may not be enough to rely only on a single metric in the detection process as this might lead to inaccuracies. To the best of our knowledge, our combined approach is the first which considers aggregating migration metrics in a decision function. The function works with any number of migration metrics, specified by the consumer.
- Following on from the need to collect events from multiple VMs and correlate them, we proposed a general algorithm to detect the migration of monitored VMs within a cluster of multiple VMs. The detection of migration considers the interactions between the monitored VM and a set of light-weight VMs whose sole purpose is to check whether the monitored VM has moved or not. This contribution could help learning about the migration state across multiple VMs owned by the consumer. We believe that it can also help to visualize migration by observing changes in the metrics.

3. **Experimental evaluation:**

- We provide a detailed evaluation of monitoring and detection of security-related VM events, in terms of performance, storage and usefulness.
- We present an evaluation of migration detection techniques by experimenting with various scenarios in common public IaaS cloud systems, e.g. Amazon Web Services (AWS) and Google Cloud Platform.

4. **Literature survey**: We present a literature survey, which covers and discusses various cloud security challenges and concerns. In addition, it highlights the state-of-the-art in the field of security event monitoring. Moreover, it provides a comprehensive survey of migration detection techniques, which is a field that is seldom covered from a consumer's and a security perspective.

1.5 Publications

Here are the list of publications related to this thesis. Some of the outcome of the publications is mentioned throughout the thesis:

Journal Articles

Al-Said, Taimur. Rana, Omer. and Burnap, Peter. VMInformant: an instrumented virtual machine to support trustworthy cloud computing. *International Journal of High Performance Computing and Networking* 8(3), pp. 222-234., article number: IJHPCN080303. (10.1504/IJHPCN.2015.071257)

This paper presents "VMInformant which is a system to monitor malicious security-related VM events and inform the consumer about them. It works in the background of the VM and does not take actions but it has some configurations to allow sending the list of selected events to the consumer according to his choice of granularity and the types of the events to be monitored. The paper also presents a taxonomy of security-related events which are specific to virtual machines. These events are the basis of the work of VMInformant. It discusses the usefulness of security monitoring against the importance of the performance and evaluates the system using several benchmarks.

Conference Papers

Al-Said, Taimur. and Rana, Omer. 2015. Analysing virtual machine security in cloud systems. *Lecture Notes in Computer Science* 8993, pp. 137-151. (10.1007/978-3-319-19848-4_9)

This paper highlights and discusses various cloud security issues, especially those which are related to virtualisation security. It sets a special focus on the status of users' data in IaaS public cloud systems with regards to deletions. Most cloud providers claim that disks are wiped after use by customers. The paper argues that this may not always be practical due to the associated complexities, which can also affect other consumer's running processes and the overall performance. Also, it shows that there is evidence of remaining traces of data after deletions which can be extracted using some data recovery tools.

Al-Said, Taimur. and Rana, Omer. 2015. Implementing migration-aware virtual machines. Presented at: *IEEE 2nd International Conference on Cyber Security and Cloud Computing (CSCloud)*, New York, NY, 3-5 November 2015. IEEE Conference Publications, pp. 54-61. (10.1109/CSCloud.2015.92)

Migration of VMs to a different region without the consumer's knowledge can create many security and privacy concerns. The paper presents a suggested migration detection system which is able to signal, or at least gives an estimate of when the VM migration event happens, without relying on the cloud provider. The migration event is one of the identified security-related VM events in the taxonomy. When that event happens, it should be captured by *VMInformant* which informs the owner/consumer of the VM. The paper provides a literature on migration detection techniques and argues that relying only on one detection technique may result in inaccurate outcomes. Hence, it suggests a combined approach where several techniques may be aggregated in a weighted function that factors the importance of techniques from the perspective of the consumer

---

Win, Thu Yein. Tianfield, Huaglory. Mair, Quentin. **Al-Said,Taimur**. and Rana. Omer F. 2014. Virtual Machine Introspection. In *Proceedings of the 7th International Conference on Security of Information and Networks (SIN '14)*, Glasgow, UK, 9-11 September 2014. *SIN '14 Proceedings of the 7th International Conference on Security*

of Information and Networks. ACM, pp. 405. (10.1145/2659651.2659710)

The paper presents a literature survey of virtual machine introspection (VMI). It highlights the concept, applications and some of the tools which can be used to introspect disk and memory and to learn about the processes of the VMs.

1.6 Summary of Thesis

- **Chapter 2**: Provides a general background about cloud computing and highlights the focus of the thesis.
- **Chapter 3**: Gives an overview of cloud security literature and a critique of the research work related to the thesis.
- **Chapter 4**: Presents an architecture of a system to monitor security-related VM events and its reference prototype implementation and evaluation.
- **Chapter 5**: Presents our research on detecting VM migrations in IaaS public cloud systems, from the perspective of the cloud consumer.
- **Chapter 6**: Highlights a general system/approach to manage the security-related VM events aggregated from multiple VMs, which belong to the same consumer.
- **Chapter 7**: Concludes and summarizes the thesis, and states future work.

Chapter 2

Background

2.1 Chapter Overview

This chapter will give the general context of the research problem this thesis is trying to address. Since this thesis is a focused document, the background chapter will only give an overview of the basics, to show why it is relevant and important in our research. Cloud computing and the concept of virtualisation will be briefly mentioned, then some of the cloud security incidents and issues will be highlighted, which will show why our research is important. More details on cloud security can be found in Chapter 3.

2.2 The Cloud Computing Model

Cloud computing is a concept which has gained considerable attention recently in a variety of fields, e.g. academia, research and enterprises. It can be traced back to the 1960's when the concept of time-sharing machines became popular, which allowed many users to use a single computer concurrently. In 1961, John McCarthy, the computer scientist, gave the opinion that "Computation may someday be organized as a public utility" [139] – much like water, electricity, etc. The term "cloud computing", however, has only been used heavily in the last decade.

Consumers in each field have different motivations for migrating their systems to the cloud [133]. According to Armbrust et al. [13], cloud computing refers to both the applications delivered as services over the internet and the hardware and software in the datacentres that provide these services. The use of datacentre-based hosting of systems and data, therefore, has significant similarities to cloud-based deployment (in

some instances, they are in practice synonymous). Jensen et al. [75] suggest that "the cloud computing concept offers dynamically scalable resources provisioned as a service over the internet." The NIST (the U.S National Institute of Standards and Technology) defines cloud computing as: "a model for enabling 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." [96]. This definition is by far one of the most accepted definitions of cloud computing.

The NIST specified five essential characteristics of the cloud computing model:

1. On-demand self-service: Computing services can be provided when needed without direct interaction with the service provider.

2. Broad network access: Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous client platforms (e.g. mobile phones, tablets, laptops and workstations).

3. Resource pooling: Computing resources of the cloud provider are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand.

4. Rapid elasticity: Capabilities can be elastically provisioned and released, in some cases automatically, to scale up and down according to the demand.

5. Measured service: Employing a metering capability such that resource usage can be monitored, controlled and reported, provides transparency for both the provider and consumer of the utilized service.

Cloud computing has three service models:

- **SaaS** (Software as a Service): The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure [96]. Examples of SaaS are email services like Gmail, Hotmail, etc. In this service model, the consumer does not manage or control the underlying cloud infrastructure, including network, servers, operating systems and storage. Users can access SaaS applications using web browsers.

- **PaaS** (Platform as a Service): It is the capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications

created using programming languages, libraries, services, and tools supported by the provider in such a way that does not give the consumer control over the infrastructure. Rather, the user can have control over the deployed applications and possibly configuration settings for the application-hosting environment [96]. An example of the PaaS service model is the Google App Engine[1] which enables consumers to deploy and scale Python and Java-based web applications and other applications. Both web services and web browsers may be used to access PaaS [75].

- **IaaS** (Infrastructure as a Service): This is the delivery of hardware (server, storage and network), and associated software (operating systems, virtualisation technology and file system), as a service [23]. An example of IaaS is Amazon Web Services (AWS), which allows consumers to rent pre-configured virtual machines (VMs) on a per-use basis. VMs are emulations of a typical computer. A VM was originally defined by Popek and Goldberg as "an efficient, isolated duplicate of a real machine." [56].

Notable benefits of cloud computing are: universal data access from anywhere via the internet; automated storage management; and no capital expenditure on hardware, software, personnel, etc. According to Perrons and Hems [110], cloud computing offers deployment flexibility and speed for small businesses. Cloud computing has four deployment models: *public*, *private*, *hybrid* and *community*. If the datacentre is accessed as a pay-per-use service over the internet then it is called "public". Alternatively, if an organisation utilizes its datacentre as a cloud to be used internally, then it is called a private cloud. A hybrid cloud is a combination of one or more clouds (private, community or public) that remain a unique entity but are bound together by standardized or proprietary access interfaces, enabling data and application portability [31]. A community cloud serves a group of consumers who have shared concerns, such as mission objectives, security, privacy or compliance policy [96].
This thesis focuses mainly on the IaaS public cloud infrastructure because:

1. IaaS allows the consumer/owner to deploy a full VM and set it up to host critical applications. With SaaS or PaaS, this level of control is not available to the consumer. Monitoring tools for SaaS and PaaS environments from the perspective of the consumer is an area that still needs research.

[1]Google App Engine https://console.developers.google.com/start/appengine

2. VMs hosted in private clouds belong to one organisation, so the multi-tenancy issue is not a concern. The fear of having one VM's data accessed by another tenant in the same infrastructure is not applicable in private clouds.

2.3 Virtualisation

2.3.1 Overview

Virtualisation is one of the most important enabling technologies for cloud computing. It is the technology that hides the physical characteristics of a computing platform from the consumers, instead presenting an abstract, emulated computing platform [94]. It lets a single physical machine simultaneously run multiple operating systems (OSs) or multiple sessions of a single OS in the form of VMs [149]. By freeing developers and users from traditional interface and resource constraints, VMs provide software interoperability, system impregnability and platform versatility [137]. In virtualisation, the physical server is called the *host* whereas the virtual servers are called *guests*. The VM manager or *hypervisor* is the component which makes different VMs independent of each other. There are three types of server virtualisation: full virtualisation, para-virtualisation and OS-level virtualisation, as described in Table 2.1.

Table 2.1 Types of server virtualisation & comparisons

	Full virtualisation	**Para virtualisation**	**OS-level virtualisation**
Availability of the hypervisor?	Yes	Yes	No
Guests knowledge of each other?	Unaware	Aware	Must be of the same OS type
Independence of guest servers	Yes	Yes	Yes
Description	The hypervisor interacts directly with the physical server's CPU and disk space.	As each guest is aware of the others and their demands, the hypervisor does not need more processing power.	Virtualisation capability is part of the host OS which performs the operations of a fully virtualised hypervisor.
Limitations	Physical server must reserve some processing power and resources to the hypervisor; this can impact server performance and slow down operations.	The guest OS must be tailored specifically to run using the hypervisor.	Guest servers must run the same OS.

IaaS consumers rent VMs according to their needs. Cloud providers give consumers many options on the choice of virtual machines by allowing them to choose the operating system type, memory size, hard disk size, etc. In general, VMs are classified into two categories:

- *System VMs*: They provide a complete system platform which supports the execution of a complete OS. For example, it provides a platform to run programs

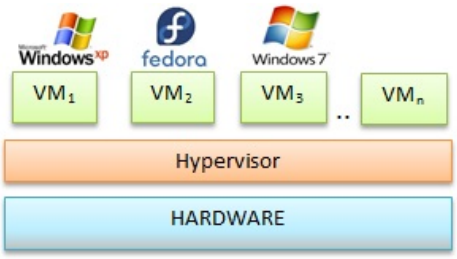

Fig. 2.1 Location of the Hypervisor (type 1-bare metal)

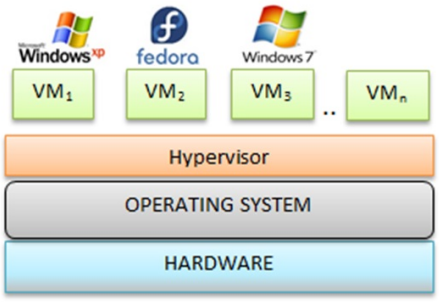

Fig. 2.2 Location of the Hypervisor (type 2)

where the real hardware is not available, e.g. legacy systems. Another aspect in the use of VMs is to improve utilisation of computing resources – the key reason for its use in cloud computing.

- *Process VMs*: Are designed to run a single program (support a single process) and are built with the main purpose of providing program portability and flexibility.

In IaaS, the virtual machines are considered *system* VMs as they run a complete operating system with its applications.

2.3.2 Hypervisor

The hypervisor or virtual machine monitor (VMM) is the basic abstraction layer that sits directly on the hardware of the physical server (Type 1 hypervisor or baremetal), see Figure 2.1. The hypervisor is responsible for scheduling and partitioning memory of the various VMs running on the physical device [157]. If the hypervisor is installed on top of the host OS, then it is said to be a "type 2 hypervisor" –Figure 2.2.

In addition to abstracting the hardware for the virtual machine, the hypervisor controls the execution of VMs as they share a common processing environment. It has no knowledge of networking, external storage, video, or any other common I/O functions. It serves as a platform for the VMs hosted on the physical server. This allows the independent running of various types of OSs which are not compatible with each other. Some hypervisors are heavily used in the virtualisation and cloud computing field, such as: Xen [19], KVM [82], VMWare, and Microsoft Hyper-V. Some cloud providers use only one type of hypervisor. For example, Google uses *KVM* in their

Table 2.2 Some of the famous virtualisation management software

Virtualisation Software	Description
VMware workstation	VMware is the leading product in desktop virtualisation. It acts as a virtual computer on which any OS can be installed.
Virtual Box	is an open source application to manage and run VMs. It can be used to run a virtual (guest) OS on any host machine that runs Windows, Linux, Mac OS X or Solaris operating system.
Microsoft Virtual PC	It allows a complete "virtual computer" to be created – and is free to download and use
Citrix XenServer	is an open source virtualisation platform for managing Cloud deployments, server and desktop virtual infrastructures. It is based on the Xen hypervisor.

cloud datacentres while Amazon uses *Xen*. A list of some potential VM management software are shown in Table 2.2.

2.3.3 How commands are executed in the VM

VMs do not interact directly with the host. The only way of communication between a VM and its host is through *system calls* and this is done via the hypervisor. A *system call* is how a process requests a service from the *kernel* of the OS it is executed on. System calls reflect every activity occurring inside the VM. Analysing system calls is one of the techniques used to monitor the activities of VMs without directly accessing them. This is called: virtual machine introspection (VMI). Chapter 3 will cover this in detail.

2.3.4 Virtualisation in cloud computing

Virtualisation is the basis of cloud computing because it simplifies the delivery of services by providing a platform for optimizing complex IT resources in a scalable manner [69]. It can be applied to memory, networks, storage, hardware, OS and application. In a typical scenario, we can imagine having a powerful physical machine with large amounts of storage, memory and a fast processor. Using a VMM, several virtual servers can be hosted on the machine. Storage, memory and processing will be divided among the virtualised servers. If a certain VM requires additional storage, then it will be easily acquired from the host machine. Public IaaS cloud providers such as Amazon AWS or Google Cloud Platform allow consumers to deploy VMs with different capabilities based on differing pricing schemes. Some cloud providers allow consumers to choose the region where they want their VM to reside. VMs in the cloud run on

Table 2.3 Some of the famous cloud management software

Cloud Management Software	Description
Apache CloudStack	an open source cloud computing software for creating, managing, and deploying infrastructure cloud services.Written in Java
Openstack	a free and open-source software platform for cloud computing. It began in 2010 as joint project of Rackspace and NASA.Written in Python.
OpenNebula	It is an free and open source cloud computing platform for managing heterogeneous distributed data center infrastructures. Written in many languages.

physical hosts. In order to manage the lifecycle of virtual machines efficiently, cloud providers use management software. Various cloud management software is available (see Table 2.3).

2.3.5 Live migration of virtual machines

Migration of physical servers requires creating backups, shutting down devices and the interruption of services. However, in the IaaS cloud, it is possible to migrate VMs from one physical server to another without explicitly shutting down and subsequently restarting the VM [38]. In the live migration of VMs, the entire state of the VM, including memory pages, are transferred to the destination host and the VM can resume its execution from its state prior to migration [14]. Clark et al. [38] provide a detailed description of the live migration process, which involves six stages: *pre-migration*, *reservation*, *iterative pre-copy*, *stop and copy*, *commitment* and *activation*. In the *pre-migration* stage, a remote hardware node is selected and resources are allocated, while in the *reservation* stage the resources are used to initialise a new empty VM container which is used to deploy the VM [24]. In the *iterative pre-copy* stage, the memory of the VM is copied from the source to the destination node. In this stage the hypervisor iteratively checks for modifications in the running VM memory, and only identified dirty pages (modified) are transferred to the destination node until few dirty pages are left. In the fourth stage, *stop and copy*, the VM is stopped and the last dirty pages are transferred to the destination node. According to [24], the interruption is only for milliseconds and running processes or established remote connections are not terminated. In the *commitment* stage, the VM in the source node is terminated and the resources are released [38]. Finally, in the *activation* stage, the VM is activated on the new server and the live migration process is successfully finished.

Fiebig et al. [46] generalized the stages into three phases: *memory copy*, *CPU copy* and *switch*. In the *memory copy* phase the VM memory state is copied in order to start the migration process. In the *CPU copy* phase the CPU is stopped and the registers are copied from the source server to the destination server. In the last phase, the VM is fully removed from the source server and started up on the destination server.

2.3.6 Virtual disks

When it comes to VMs, and especially the migration aspect, it is important to talk about virtual disks because this is where the operating system and applications will run. A virtual disk is a file or set of files that appears as a physical disk drive to a guest OS. These files can be on the same physical host machine as the VM or on a remote computer or a central network storage system. Some cloud providers like Google Compute Engine offer several types of data disks that can be used as primary storage for virtual machine instances: persistent disk storage and local SSD storage. Persistent disks can be attached and detached to any instance, while local SSD disks can be physically attached to a server that is running the instance. Hence, data stored in persistent disk storage remains intact regardless of the state of the instance to which it is attached, while with a local SSD, the data does not persist beyond the lifetime of the instance. Throughout the thesis, persistent disk storage attached to the instances is considered.

2.3.7 Virtualisation API

In order to facilitate the communication between applications and the virtualization capabilities of recent versions of Linux and other operating systems, Libvirt can be used. Libvirt [25] is an open source API, daemon and management tool which has support for a wide range of hypervisors and bindings to many programming languages. Developers can use the language of their choice to develop applications which can manage the lifecycle of VMs. For example, they can create, stop, snapshot, remove or migrate VMs. The Virt-manager tool developed by Redhat, also known as Virtual Machine Manager, provides a graphical tool for administering virtual machines, and it uses the Libvirt library as the management API.

2.4 Containers in the cloud

Another technology which is based on the concept of virtualisation, is containers. This is an OS-level virtualisation method for running multiple isolated Linux systems (containers) on a physical host by using a single Linux kernel [18]. Containers were once typified by FreeBSD jails and chroot jails [152], but are now more commonly associated with Docker [97]. Unlike virtual machines which can run different operating systems regardless of the operating system of the host, containers have to run the same operating system as the host. The discussion of containers with regards to security is beyond the main scope of the thesis. However, we will highlight some aspects of it in Chapters 4 and 5.

2.5 The Multi-tenancy Issue

IaaS public cloud services are cost effective due to their multi-tenancy nature. Multi-tenancy simply means that the hosting infrastructure and the address space are shared among thousands of consumers. In practice, this implies that a consumer's data could reside on the same physical machines as a competitor. Consumers theoretically have no clue as to exactly where their data and applications reside nor do they have a clue about the other tenants sharing the infrastructure with them. Cloud providers generally rely on hypervisors and access policies to isolate consumers from each other and ensure that no one can see data of another tenant. Due to the dependency on the security of the hypervisor, flaws in the implementation or access policies management could affect all aspects of consumers' security [7]. This will be discussed in more detail in Chapter 3.

2.6 Cloud Computing Actors

The NIST Reference Architecture document [88] describes five major actors with their roles and responsibilities, namely: cloud consumer, cloud provider, cloud auditor, cloud broker, and cloud carrier. According to the document, each actor is an entity (a person or an organisation) that participates in a transaction or process and/or performs tasks in cloud computing. Table 2.4 shows the five different actors and describes their roles.

The actors are the same for all cloud service models. However, since the focus of this thesis is on the public IaaS cloud, throughout the thesis, when we refer to cloud

Table 2.4 Cloud Actors according to the NIST Reference Architecture [88]

Cloud actor	Description
Cloud Provider	an individual or organisation or entity responsible for making a service available to consumers
Cloud Consumer	an individual or organisation that acquires and uses cloud products and services.
Cloud Auditor	conducts the independent assessment of cloud services ,operations, of performance and security
Cloud Broker	An intermediary between consumer and provider which helps consumers through the complexity of cloud service offerings and may also create value-added cloud services.
Cloud Carrier	Provides connectivity and transports cloud services from the provider to the consumer

consumers, then we mean IaaS public cloud consumers. Public IaaS cloud consumers could be individuals or organisations. In practice, there are providers of services who depend essentially on offerings from public IaaS cloud providers. For example, Netflix[2], which provides streaming media and on-demand video over the internet is a consumer of Amazon Web Services (AWS). AWS enables Netflix to quickly deploy thousands of servers and terabytes of storage within minutes, so that users can stream Netflix shows and movies from anywhere in the world [15]. Another example of a service provider (also a consumer) which uses AWS is Airbnb[3], a marketplace for people who want to list or rent houses.

In this thesis, our main focus is on cloud consumers who could be individuals or organisations requesting IaaS services in the form of virtual machines from public cloud providers. Those cloud consumers could have different types of users depending on the privileges given. Throughout this thesis, we use one term "consumer" to refer to the *owner* of hosted VMs in public cloud systems.

2.7 Practical IaaS Cloud Scenario

Since the focus of this thesis is on the public IaaS cloud, it is important to describe a typical scenario to scope this thesis properly. Consumer of different needs and requirements approach IaaS cloud providers requesting virtual machines with certain specifications. Some cloud providers, e.g. Amazon, ask consumers about the region they would like their virtual machines to reside. Some regions have different zones which include different datacentres. When the requested virtual machine is launched on one of the physical servers, it can be up and running in few minutes. The cloud

[2]Netflix www.netflix.com
[3]airbnb: www.airbnb.com

consumer is given a private key which is associated with the user account. This private key is very important and has to be kept secure otherwise it would not be possible to access the VM. To access the VM, the cloud consumer could either: (1) use the direct online shell that is usually provided by the cloud provider; (2) use some third party SSH tools to connect to the virtual machine; or (3) use some remote desktop connections to access the virtual machines. Using SSH tools, such as Putty[4], to access a VM is the most common way. Each VM is given a private IP address and an external/public IP address. The private IP is mostly used for internal communications within the platform and does not usually change even if the VM is migrated to another zone or region. Developers can use the private IP to write code if they need to address the VM directly. The public/external IP is used to communicate with the VM from the outside world. Consumers can configure some security settings to govern the roles of users accessing the VM and to set up network ports. They do not know in which physical host their VM is located and they do not know about other VMs or the other consumers who share the physical resources with them.

Physical hosts need to be maintained from time to time or patched. Some cloud providers such as Amazon notify the VM owners beforehand, requesting them to turn the VM off and prepare for the scheduled maintenance. During the maintenance, VMs will be restarted but on a different host. On the other hand, Google, has the ability to move/migrate the VMs lively from a host to another with no downtime. This degrades the performance of the VMs, as mentioned on their website [57], but it allows the processes to run as usual without interruption.

2.8 The Shared Security Responsibility and the Conflict in Perspectives

Security is a major issue when using a cloud system, and for many consumers it is one of the most significant barriers to moving services to the cloud [65]. Surprisingly, security can also be one of the reasons a consumer may want to move to the cloud, especially if they are an SME without experienced technical staff.

[4]Putty: http://www.chiark.greenend.org.uk/ sgtatham/putty/

Cloud providers provide all the services needed for consumers to run their businesses. The cloud provider has a security responsibility: protecting the infrastructure; ensuring availability; and adhering to SLAs and other legal requirements. Their main priority is to ensure that: (1) their systems will not be attacked; (2) their infrastructure is not going to be used to create attacks; (3) consumers' data or virtual machines are safe from attack either from outside the cloud or by one of the co-resident VMs in the host. While they have these security responsibilities, there are things which they cannot control such as how the consumer uses the VMs. For example, they cannot control the insider effect, where a dishonest employee steals data from the VM or change configurations. Also, they can not interfere if there is any misuse due to the negligence of the consumer or their employees. The lack of awareness from the consumer end could result in severe problems which the cloud provider cannot be held accountable for. Some cloud providers offer additional security features, while some allow third party companies to offer these security features to the cloud consumer for a fee. Allowing third party companies to offer value-added security services to the cloud consumer also means that data has to be exposed, so the privacy of consumers may be at risk. For consumers who require a great deal of security and privacy, handling this issue by themselves is very important.

We have to differentiate between security services which can be provided for the use of the API in the cloud environment — e.g. to access the dashboard and change the settings — and the security services which can be provided for the VM itself. For example, if a new user account is created *in a VM* which is running in the cloud, the cloud provider will not know about it, nor should they. This is entirely the responsibility of the consumer. The same applies if the consumer does not protect their credentials or private keys properly. At the same time, the cloud consumer may not have the expertise or the tools to detect such events and if the creation of this user account was malicious, the privacy of the consumer would be at stake. Many SMEs have little or no knowledge about security. According to Ion et al. [73], most companies mitigate risks by negotiating legal terms with the cloud provider to share liability in case of security breaches and the unavailability of data.

This brings us to two perspectives:

- The cloud provider's perspective: ensure that their services are used lawfully. Some cloud providers may check to see if the consumers are doing anything illegal. This may require them to examine code running on VMs or analyse the processes which run in them either intrusively or in a non-intrusive manner. They know

that they have contracts and agreements to satisfy and they cannot sacrifice the security of their consumers because of one unlawful consumer. Also, in some countries, cloud providers have to obey government's orders if they request an investigation of any kind on the platform.

- The cloud consumer's perspective: consumers have signed a contract or agreement with the cloud provider for services. Of course, since the VMs or the consumer's data in general are hosted offsite in the cloud provider's facilities, there must be a level of trust between consumers and providers. Consumers trust that the cloud provider will protect the infrastructure and will not violate their privacy by interfering with what they do or by disclosing confidential information or selling their data to other parties.

Many researchers in the industry or academia have focused their research on these perspectives. For example, there exists much research which discusses security in favour of cloud providers whereas there also exists research which discusses security from the perspective of cloud consumer.

In this thesis, we focus on cloud consumers and on ways we can give them some visibility on how their VMs are processed and managed in the cloud. One reason for this is the limited control consumers have on the cloud's infrastructure. Also, to promote the importance of transparency between the consumer and the provider. We argue that our research supports trustworthy cloud computing. Although our research is not from the perspective of the cloud provider, we argue that getting the consumer to trust the provider will help the provider eventually. If the trust between the consumer and the provider is enhanced, the business relationship between them will continue for longer periods. Other consumers will also be willing to deal with the same provider based on this trust.

2.9 Security in the Cloud

Information security in general is the practice of protecting information from unauthorized access, unauthorized use, disclosure, theft, modification, recording or damage. Confidentiality, integrity and availability are considered the main principles of information security. Confidentiality means protecting sensitive information from unauthorized disclosure. Integrity is the property of safeguarding the accuracy and completeness of

assets, while availability is the property of being accessible and usable upon demand by an authorized entity (ISO 27001:2005) [103]. Any evaluation of a system against security must cover these three general principles in order to reduce the risks surrounding the system. In our discussion we will mainly focus on these principles.

When it comes to the cloud, security is one of the most often-cited objections to cloud computing [13]. Although security is the biggest concern in cloud computing, there is always an argument that data could be safer in the public cloud than in privately-managed facilities of organisations which do not specialize in IT [110].

In a report by Alert Logic[5], research shows that there is an increase in *attack frequency* on organisations with infrastructure in the cloud [89]. The reason for this is the increased trend of migrating data and applications to the cloud. In 2011, the Sony PlayStation Network was hacked and sensitive data for 77 million subscribers were compromised. This breach was reported to be the second-largest online data breach in U.S. history [87].

In 2012, Dropbox[6] was subject to a breach and about 68 million account credentials were leaked online [100]. Companies could get bankrupted due to attacks on their services hosted in IaaS clouds as well. In 2014, a company called "Code Spaces" went out of business due to hackers attacking the console for Code Spaces' AWS account and deleting all of the company's files [156].

It was argued by Chen and Katz that only few cloud security issues are new or specific to the cloud, by trying to filter out what is new and what is not [33]. According to [88], security is a cross-cutting aspect of the architecture that spans across all layers of the cloud components ranging from physical security to application security. This implies that it is not enough to look only at one aspect of security when it comes to the cloud. We argue that the physical security of the cloud infrastructure is similar to the security of classic datacentres and it would be more practical to look for security concerns which are cloud-specific. In its "Cloud Computing Reference Architecture" document, the NIST states that cloud-based systems must address security requirements such as: authentication, authorisation, availability, confidentiality, identity management, integrity, audit, security monitoring, incident response, and security policy management.

[5]Alert Logic https://www.alertlogic.com/
[6]Dropbox: an Online storage platform www.dropbox.com

While there are many security benefits of using the cloud, such as centralization of security, data and process segmentation, redundancy and high availability [167], there are many other security issues. Centralization of security reduces the complexity and burden on security administrators, but since the data is outside the consumer's premises, the consumer has no control over it. Redundancy may help in the case of catastrophic events where backup data could be retrieved from another site. However, this may lead to privacy issues if the replicated data is not removed from the cloud when the consumer wants. Perhaps the biggest cloud security threat is the fact that consumers/tenants are all using the same physical infrastructure of the cloud provider. Armbrust et al. [13] argue that outside threats facing cloud consumers' data are similar to those facing current datacentres, but the responsibility in the cloud computing setting is potentially divided among several parties. For example, cloud consumers are responsible for application level security while cloud providers are responsible for physical security. Cloud providers cannot be held responsible for security breaches which occur due to vulnerabilities in consumers' applications. Countermeasures must be put in place to avoid such incidents from the consumer's side. It is not enough to only consider outside threats, but there could be inside threats as well. Dishonest employees of both the consumer and the provider could steal, manipulate or damage data, and if there are no detection mechanisms in place, this may not be discovered.

To the best of our knowledge, there are no reported incidents which involved attacks caused by the migration of VMs from a region to another in public IaaS cloud systems, without notifying the consumer/owner. Although this scenario is possible and cloud providers may perform this for a variety of valid reasons, they may wish to hide it from the consumers; to avoid SLA breach claims. One instance of migrating VMs without notifying consumers (initially), was when Google used the live migration feature to start a transparent maintenance and move VMs to other hosts within zones. This was to recover from the *Heartbleed Bug*[7], which affected the OpenSSL cryptographic software library. Although this was to the benefit of the consumers whose VM workloads were not affected or interrupted, the fact that migration occurred without notifying the consumers shows that this is possible. If Google did not publish or report what happened, perhaps none of the consumers would have even noticed.

In general security-related challenges in the cloud may include:

- *Privacy Challenges:* Keeping consumers' data private is important, especially given the fact that the platform is shared among many tenants.

[7] http://heartbleed.com/

- *Trust Challenges:* There is a lack of trust between providers and consumers [7]. Consumer data is hosted outside its premises and under the control of the cloud provider's staff. This means that the data and applications could be viewed and altered. Consumers are not sure if the cloud provider is protecting the infrastructure properly or following the best practices in security.

- *Legal & Auditability Challenges*: An SLA (Service Level Agreement) is a written contract between the cloud provider and the consumer. Providers have to adhere to it. Breaching the SLA requirements could result in legal issues for the cloud provider. We argue that it may also affect cloud consumers who provide services to consumers by using services from the cloud provider. In the case of incidents, due to the shared platform, it may be difficult to audit. Ensuring the adherence to SLA by monitoring the performance and availability has been highlighted in many research papers, such as in [141].

- *Availability Challenges:* Consumers trust that providers will keep their services running all the time so that end users can access them anytime from anywhere. Outages are common in the cloud and it can be costly for both consumers and providers.

- *Virtualisation Challenges:* Perhaps most of the security challenges in the cloud are virtualisation-related. Even though virtualisation is a security mechanism, it has introduced many security challenges.

- *API & Browser Challenges:* Users access cloud services using APIs (application programming interfaces) or web browsers. The security of these interfaces and browsers is important when it comes to secure cloud services.

In Chapter 3, we will cover the state-of-the-art on these challenges with a special focus on virtualisation security. The reason for this focus is that virtualisation serves as the basis for cloud computing, and in the IaaS if the isolation mechanism based on virtualisation fails, all VMs are compromised. We also discuss how these challenges are addressed in research and the industry.

This thesis presents internal monitoring as a solution to some of the privacy, security and trust challenges. Internal monitoring means that selected security-related VM events are monitored and recorded internally from the VM itself, without reliance on monitoring provided by the cloud provider. This way, the cloud consumer can ensure

that the data processed in the VM has not been tampered with and the the VM itself has not been migrated, cloned, etc by the cloud provider.

Monitoring events in VMs hosted in public IaaS cloud systems can be similar to monitoring them in a classical physical server or a laptop. However, there are major differences. First, physical servers in the consumer's premises are controlled and used only by the consumer, i.e. no sharing of resources. If the data is processed in a VM outside the consumer's premises, then it is under the control of another party and it can be located anywhere in the world. This could make it under different legislations and data protection laws, especially if the VM was migrated later by the cloud provider. According to Gul and Hussain [61], in case of reported incidents, cloud providers may not like to inform the user about the loss and can hide the information for the sake of protecting their image and reputation. Second, consumers pay for VMs hosted in public IaaS cloud systems. Monitoring events in VMs and reporting them to the consumer require the usage of virtual storage and computing resources. Since VMs have limited resources compared to classic physical servers on premise, exploiting the available computing resources in VMs require a more focused monitoring, i.e. recording efficiently according to the need. This is also to cut cost of VM rental. Another difference is related to the life-cycle events of VMs, which are not applicable in the medium of physical servers. VMs can be migrated lively to another physical host in the cloud without notifying the owner of the VMs. They can also be snapshotted, cloned, deleted, paused, etc. This brings a different breed of events which can either be triggered by the consumer or the provider. We argue that detecting these events as they occur is useful to ensure the privacy and the integrity of data hosted in the cloud.

We argue that internal monitoring can indirectly address legal & auditability challenges. Addressing API & browser challenges is out of the scope of this thesis. However, in Chapter 3, we will give an overview of them.

2.10 Chapter Conclusion

Cloud computing has become popular in the last few years, mainly because of the various advantages it provides. This chapter highlighted some basic cloud computing concepts to set the scene for the subsequent chapters. We concluded that the hypervisor in each physical host provides a layer of isolation for the VMs. We indicated that in this thesis our focus is on public IaaS cloud systems because: (1) IaaS gives more control to the consumer and allow them to deploy full VMs to host application; (2) Security/privacy concerns in public cloud systems are more evident than they are in

private clouds; private clouds are used by only one organization. We also highlighted the importance of information security in cloud computing because of the *multi-tenancy* and *co-residency* features. Data of consumers may reside in the same physical machines as their competitors. This may cause some privacy concerns. We explained how the responsibility of security is shared between consumers and providers in the public IaaS cloud system. The cloud provider is responsible for: protecting the infrastructure; ensuring availability; and adhering to SLAs along with other legal requirements. The consumer is fully responsible for deploying and securing their VMs. For example, patching of VMs is normally performed by the cloud consumer. Managing the users accessing the applications in VMs is also the responsibility of the cloud consumer. This leads to a conflict in the perceptions of security for both consumers and providers. This conflict drives the issue of trust between the consumer and the provider. For example, to protect the infrastructure, some cloud providers attempt to analyse code running in VMs. We indicated that our focus in this thesis is on the perspective of the *consumer*– not the provider. We argued that consumers of public cloud services have to have visibility on how their data are accessed and processed. We presented monitoring of security-related VM events as a solution to support the trust between the consumer and the provider. Chapter 3 discusses cloud security challenges in detail, while Chapters 4, 5 and 6 highlight our research into monitoring security-related VM events.

Chapter 3

Cloud Security: Literature Survey

3.1 Chapter Overview

This chapter contains a detailed literature survey about cloud security challenges. It starts with coverage of the wide range of cloud security challenges discussed in the literature, in order to clarify the motivation behind research into cloud security and to understand the relationships between these challenges. The focus of the discussion in this chapter is on cloud virtualisation security challenges. However, many of the challenges overlap. For example, some virtualisation challenges could directly affect the privacy of consumers' data in the cloud. Then, approaches to address the cloud security challenges are surveyed from the literature, with an emphasis on monitoring. This chapter compares some monitoring tools against our work to clarify why our work is different. Some of the work formed the basis for publications in [4] and [155] .

3.2 Cloud Privacy Challenges

Privacy is one of the most cited concerns in cloud security literature especially due to the sharing of resources by a huge number of consumers. Also, this is due to the fact that data of the consumers will be hosted away from their premises and the cloud provider will have full control over it. Shifting critical services to the cloud means accepting the risk that data could be viewed, manipulated, stolen by the cloud provider or any other malicious consumers. Privacy complexity increases in federated clouds, where multiple cloud providers may collaborate with each other to provide services to the consumer. It is not always a question of the cloud provider's dishonesty; the

provider's infrastructure itself could be compromised and the data of all consumers could be at risk. Similarly, terms have to be clear to the consumer about the case when the cloud provider is out of business due to bankruptcy or legal matters. Having no clear terms on what happens to the data in such a situation may lead to unpredictable privacy implications. Some cloud providers use introspection techniques for the purpose of securing the whole virtualisation platform. While this is a useful application of introspection, it may lead to disclosing critical information that the consumer does not want to disclose. Notorious cloud providers or their dishonest staff could use these methods to know what the consumer is doing and sell this information to competitors. Various cloud provider are already making use of the data stored by consumers to display customised adverts, which means that they are performing some sort of analysis on the data, e.g. email services. Cloud management software used by cloud providers is not bug free. Wang et al. [150] mention that OpenNebula (an open source cloud system) had a bug which leaves user passwords accessible by anyone on the network. This means access to confidential data can be easily done using APIs or web browsers.

3.3 Trust Challenges

According to Zissis and Lekkas [167], entity A is considered to trust entity B when entity A believes that entity B will behave exactly as expected and required. In the cloud computing environment, when the consumer approaches the cloud provider for services, trust is assumed. However, the trust between the cloud consumer and provider should not be absolute. It is not uncommon for some malicious users to try to brute force passwords illegally or control botnets using the offerings of the cloud itself. Chen et al. [34] suggest that the consumer code in the cloud must be examined. In addition, believing that thinking cloud services will always be running and available may lead to the consumer not taking backups, conducting risk assessments, etc. [33]. To improve trust in the cloud, Zissis and Lekkas [167] propose a TTP (trusted third party) based on PKI (public key infrastructure) to ensure confidentiality, integrity and authentication of data.

We argue that the transparency between the cloud provider and the consumer could enhance the trust between both parties. The NIST cloud reference model document suggests that there should be an auditor entity which conducts an independent assessment of cloud services, such as IS operations, performance and security of the cloud implementation [88]. This is also to support the trust and distribute the responsibilities accordingly. Another dimension of the trust has to do with malicious insiders from

both the cloud provider and the cloud consumer side. Malicious insiders from the provider side could view, manipulate or expose consumers' data. From the consumer side, they can leak login data or private keys which allow attackers to access VMs remotely.

3.4 Virtualisation Security

According to Armbrust et al. [13], virtualisation is the primary security mechanism in the cloud, and is intended to isolate the system and data of one user from another. However, not all resources are virtualised and not all resources are bug free. Vaughn-Nichols [149] argues that organisations are facing a challenge in securing virtualised systems, which are vulnerable to the same threats as physical systems, including intrusions and malware infection. The problem could be even worse, as it was estimated that 60% of VMs in production are less secure than their physical counterparts, mainly due to not involving the information security teams in the initial architecture and planning stages [116].

Despite being also a security mechanism that is intended to isolate services and applications from each other, there are many security challenges associated with virtualisation. There is significant literature which covers these challenges in detail [149, 114, 108, 92, 85, 63, 37, 11]. According to [48], the threats that work in a physical world could also work in a virtualised world and could be more devastating. The reason is that these threats could propagate much more rapidly within a virtualised environment to affect the other guests of the physical host.

In the discussion of virtualisation security, one has to consider various aspects. According to Anand et al. [11], the security of any virtualisation solution is heavily dependent on the individual security of each component, from the hypervisor and host OS to the guest OS, applications and storage. In any virtualisation system, the hypervisor allocates the host machine's resources to each virtualised OS or to each program running on a virtualised OS. It emulates a hardware device for each VM and handles the communication between the CPU, the storage medium and the network via the OS [149]. In the IaaS model, attacking the hypervisor would mean compromising all the running VMs (managed by the hypervisor). Perhaps the biggest threat to virtualisation security is that hypervisors have more privileged access to hardware resources than typical applications [149]. Studnia et al. [143] provide an overview of the use of virtualisation and demonstrate that the widely-used VMM or the hypervisor

cannot be considered fully secure. They argue that bad configuration or design flaws in the VMM could lead to a denial of service, system halt or *VM escape*. Some researchers argue that the security of the cloud provider in general also needs to be considered [79].

To provide a discussion on virtualisation-specific issues, the following sections will focus on three components: the physical host, the hypervisor, the VM and the guest OS. Common attacks on these components will be described. Classic security issues which are related to the cloud provider's datacentre are out of scope. The discussion of the issues below does not try to be exhaustive; rather, it aims to survey the important issues affecting security and privacy in the IaaS public cloud, mainly from the consumers' point of view.

3.4.1 Hypervisor hyperjacking

This type of attack is where a malicious user takes control of the hypervisor. This represents a central point of failure, as compromising the hypervisor means compromising all the VMs under it. "Hyperjacking" involves an attacker running a very small footprint hypervisor that takes complete control of the host OS [114].

Examples of hyperjacking are "virtual machine rootkits" (VMBR) or "Subvert" [81], which was developed as a proof of concept. The Subvirt rootkit injects itself between the hardware and the original OS and then runs the original OS as a guest OS. Taking control of the main OS may lead to the compromise of the whole system. Installation of SubVirt on the target system requires sufficient privileges to alter the boot sequence, which can be done by exploiting a known vulnerability or exploiting phishing methods to trick administrators into executing malicious code [136].

Another example of hyperjacking, introduced by the security researcher Joanna Rutkowska, is the "bluepill" malware that executes as a hypervisor to gain control of a computer's resources [126]. The bluepill starts a thin hypervisor under the main OS. The OS can still maintain reference to the devices but has no control over it. Joanna demonstrated a subvert of Windows Vista running under AMD-V such that it runs as a VM under the control of the thin hypervisor. The way it works is by bypassing Microsoft Vista's digital signature protection for kernel drivers and then manipulating the kernel memory and AMD-V SVM privileged resources [136]. Dai Zovi [39] introduced Vitriol, which is a similar rootkit that exploits MacOS X running on Intel VT. Neither technique requires modifications to the BIOS or boot sequence

prior to installing the rootkits [136], which makes their detection harder.

The idea behind the aforementioned rootkits lies in them being undetectable or *transparent*. King and Chen [81] claim that their rootkit cannot be detected easily, and Rutkowska [126] suggests that the bluepill is 100% undetectable. This claim was proved to be wrong by Garfinkel et al. [51] when they surveyed the various VM detection techniques which can be used to differentiate between virtual and real hardware.

Since most hyperjacking techniques were demonstrated as a proof of concept, a question that can be asked is how serious such types of attack are on a real IaaS cloud. For example, can one inject the bluepill into one of the VMs running in a physical host in AWS and control the hypervisor? To the best of our knowledge, there has been no incident reported which involved attacking IaaS cloud infrastructures using this method. The reason could be the dependency of such rootkits on specific processor architectures. According to Skapinetz [136], the threat of virtualised rootkits is low because their success largely rests on the existence of hardware virtualisation. Otherwise, it can be detected easily.

3.4.2 Insecure management console

In the baremetal hypervisor (type 1) as covered in Chapter 2, a management console is usually used to configure the hypervisor remotely and carry out management tasks on the VMs. According to [11], some virtualisation products offer multiple ways to manage hypervisors, so if the management interface is not secured, the hypervisor will be under threat. Also, if the management console is accessed remotely, communications must be protected.

In December 2013, the openSSL website was breached [42]. Instead of attacking the website itself, the attackers targeted the hosting company which stored the website on a host machine with an insecure management console. This also emphasises the effect of human error because using weak passwords to access the management console could result in compromising all of the services. In [92], it was suggested that poor isolation or an inappropriate access control policy would cause an inter-attack between VMs or between VMs and the hypervisor.

3.4.3 Side and covert channel attacks

A covert channel is a channel that is not intended for communication [151] and that is intentionally hidden, while side channels act like a backdoor to leak sensitive information. The basic idea of a side-channel attack is to determine the secret key of a cryptographic device by measuring its execution time, its power consumption, or its electromagnetic field [93]. Both types of channels can result in information leakage by examining cache memory, which is a small high speed section of memory built inside and outside the CPU [160].

Side-channel attacks are useful when co-residency of VMs is achieved. *Co-residency* means that a given VM is hosted on the same server as another VM in the IaaS cloud. Ristenpart et al. [120] experimented on AWS how they can identify where a particular VM resides in the cloud and instantiate VMs until one of the VMs is placed co-resident with the target VM. Then, they used cross-VM side channel attacks to extract sensitive information from the target VM.

Zhang et al. [165] demonstrated that it is possible to reconstruct private keys of certain VMs from another VM on the same host using side channel attacks. Using the private key, attackers could access the VM and manipulate its data. Although they demonstrated the attack in a lab setting, the same threat exists in the public IaaS cloud system. To mitigate side-channel attacks, Zhang et al. suggest avoiding co-residency with other VMs in high-security environments. However, this may not be feasible in public cloud systems where sharing of resources is one of the features. Harnik et al. [64] illustrated how data deduplication happen in cloud storage, a technique used to store only a single copy of redundant data, can serve as a covert channel to communicate with attackers and a side channel to leak sensitive information.

Much research was done in the area of detecting co-residency by exploiting side channel attacks. Zhang et al. [164] introduced “Homealone”, which is a system that inverts the usual application of side channels so that they can be used as a defensive detection tool rather than being exploited as a vector of attack. They argued that tenants using Homealone could detect the activity of a co-resident VM by using an L2 memory-cache-based side channel. This is particularly useful for consumers to check the cloud provider’s adherence to the terms of the SLA, especially if it states that the host is exclusively reserved for the consumer.

Bates et al. [21] followed another approach to check co-residency without reliance on internal side channels. They presented co-resident watermarking, a traffic analysis attack that allows a malicious co-resident VM to inject a watermark signature into the network flow of a target instance. They were able to confirm co-residency with a target VM instance in less than 10 seconds. Younis et al. [160] gave a comprehensive survey of cache side-channel attacks and how they benefit from multi-tenancy and virtualisation in cloud computing.

3.4.4 Denial of service attacks: DoS

In the context of VMs hosted in an IaaS public cloud, a denial of service attack could mean several different scenarios: (1) one of the VMs uses all the computing capacities of the host preventing other VMs from running correctly. As a result, VMs will not be available for end users to access its resources and applications. (2) The automatic live migration feature of VMs is exploited by malicious users by triggering multiple unnecessary migrations. This is called *over-committing*. (3) Because the VM is accessed like a normal computer, changed login information by a malicious intruder will deny access to the VM.

The decision for automatic migration of VMs in an IaaS cloud is taken depending on some parameters or thresholds, which are usually hidden from the consumers. To trigger the migration of VMs, Alarifi and Wolthusen [8] used a stress test tool which is widely available and can easily be used in IaaS clouds. Later, they designed a much more sophisticated and harder to detect attack [6]. It works by trying to estimate or discover the migration triggering parameters in order to cause the VM to migrate. The attack exploits two features of the cloud, which are: migration and over-commitment. 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 at the same time [163]; much like what happens in hotel reservation when they allow more rooms to be reserved than are actually available. As a result of the described attack, VMs could enter a continuous migration state and the entire cloud will have difficulty functioning, affecting the availability and leading to a breach of the SLA and a loss of reputation and potential customers [7]. Although this type of attack mainly affects the cloud provider, consumers' VMs are also affected as they can be migrated to a host which cannot fulfil their demands, leading to a VM crash. Although Alarifi and Wolthusen [6] tested the attack in a controlled environment as a proof of concept, this type of attack can also be executed in a public IaaS cloud.

Varadarajan et al. [148] defined a resource freeing attack. Cloud computing is based on a shared pool of resources where many VMs share the host's available resources. In a resource freeing attack, a VM consumer could suffer from performance degradation due to the activities of a malicious user co-resident on the same physical machine. In their paper, Varadarajan et al. found that the performance of a cache-sensitive benchmark can degrade by more than 80% because of interference from another VM.

3.4.5 VM escape

VM escape is an exploit in which the attacker runs code on a VM that allows an OS running within it to break out and interact directly with the hypervisor [92]. Hence, an attacker may gain access to the memory located outside the region allocated to the corrupted VM; in an environment which has access to the host OS. The attacker could monitor the memory being allocated to the other VMs from the host or just terminate the hypervisor, causing unavailability issues [74]. Many bugs which allow escaping from a VM have been found in famous virtualisation software, such as Microsoft Virtual PC/Virtual Server, VMware and Xen [63]. All of these attacks are possible because there is a possibility to detect whether there is a hypervisor running underneath the OS, as well as its type.

3.4.6 Inter-VM attack or VM hopping

This attack is from one VM to another VM that resides on the same physical host in the IaaS cloud. Thus, achieving co-residency is a pre-requisite. If the attack is successful, the attacker can monitor the target VM's resource usage, modify its configuration, and delete data, endangering that VM's confidentiality, integrity and availability [146]. Attackers can have full control of the VM once they get inside it [134]. This type of attack can make use of poor isolation between VMs and in its more sophisticated form, it can also be facilitated using side-channel attacks as described earlier.

3.4.7 Live migration attacks

Live virtual machine migration is a virtualisation technology innovation which allows the moving of a running VM from one physical host to another without causing services running in the VM to shut down. Despite the obvious benefits (which include: freeing hosts to carry out maintenance, distributing the work load in datacentre, and reducing energy consumption) live migration could be exploited by malicious people. Live

migration is susceptible to many attacks, like "man-in-middle", "denial-of-service", and "stack overflow" [10]. Oberheide et al. [102] highlighted the importance of securing the migration process, and they listed three threats to the process: (1) the control plane, which are the communication mechanisms employed by the hypervisor to initiate and manage live VM migrations; (2) the data plane, which is the medium across which migrations happen; and (3) the migration module, which is the component that implements live migration functionality. All of these components must be secure. They developed a tool, called *Xensploit*, to perform man-in-the middle attacks on the live migration of VMs. They showed that Xen and VMware are vulnerable to attacks that exploit their live migration feature.

In addition, managing VM migration could add another level of complexity to the security process, especially when the VM is migrated to an unsecure host. It is suggested by Garber [48], that virtualisation is very dynamic, with systems constantly creating and shutting down VMs or moving them to different hosts; and so the entire security process must also be dynamic. It has to take into consideration changes that occur to the platform all the time. In this thesis, we look at how migration of VMs can be exploited to breach the SLA and affect the confidentiality of consumer's data. Our main focus is to enable the consumer to detect the migration of VMs in order to make trust decisions.

3.4.8 Remnants of data

Data in the cloud is stored in disks of different kinds. Consumers using VMs in the cloud are accessing virtual disks which are attached to the VMs. When data is deleted from these disks, remnants of data are not permanently deleted and users could use data recovery tools to recover the data. A security assessment done on Rackspace — a famous cloud provider — indicated that some virtual servers contained data processed on other virtual servers. This was due to the improper wiping of disks and to the way the hypervisor was configured to read/write from the disk [76]. In [4], we experimented with the recovery of data from VMs using data recovery tools. An explanation for the ability to retrieve deleted data touches on the structure of file systems. For example, DOS and Windows file systems use fixed-size clusters, so even if the actual data being stored requires less storage than the cluster size, an entire cluster is reserved for the file. This unused space is called slack space. The slack space may contain data which has been previously deleted. Since the allocated space for any new VM is not necessarily given in successive disk locations, there is a possibility for previous data to be within

the allocated space. This data may not be related to the VM itself; it could be left out of past VMs.

3.4.9 VM sprawl

VM sprawl is another issue in virtualisation environments. It happens when the number of VMs is continuously growing, while most of them are idle or never return back from sleep mode [92]. It is a result of the ease with which new VMs can be created and proliferated on virtualized servers [130]. Creating VMs can be a matter of just few clicks and simple configurations. In a public IaaS cloud, every VM incurs a cost, even if it is not used or running, if storage is reserved. With time, consumers may end up with many VMs which are not utilized. VM Sprawl can also be a problem for cloud providers, but the effect of this on cloud providers is out of this thesis' scope. According to Commvault[1], about 30-40% of the VMs created end up being unused and about 10% of them have an impact on cost as well. This could lead to overuse of the infrastructure. Bourdeau [26], also identified VM sprawl as a problem, as well as its root cause. He adapted the concept "Reduce, Reuse, Recycle" of VMs as a mitigation plan.

3.4.10 Guest OS-related attacks

It is widely observed that traditional OSs have vulnerabilities – hence attacks which exploit these vulnerabilities may also work against virtualised OSs with the same vulnerabilities [149]. However, securing VM operating systems cannot be performed in the same way as securing a typical OS. For example, typically, security for a system of machines is enforced over the network by placing physical hardware, such as firewalls, between devices [48]. In contrast, in a virtual environment, hardware cannot be placed between VMs.

Cloud providers usually provide public images from which consumers can instantiate VMs. For example, Amazon provides an Amazon Machine Image (AMI) which provides the information required to launch a VM (instance). A considerable number of vulnerabilities were found in instances running AMI images. A large number of AWS public images were examined in [29, 17, 162] by launching instances and analysing them using security tools. Bugiel et al. [29] examined about 1,255 AMIs using privacy tools that they had developed. They found private keys and credentials, private data

[1]Commvault: VM Sprawl http://tinyurl.com/nxukpm4

and application source code. Balduzzi et al. [17] developed an automated system to download and analyse about 5,000 instances of AMI public images from four different datacentres. They discovered that some of these VM images issued unsolicited connections to suspicious sites. Also, they discovered backdoors, left out credentials and malware which monitored the browsing habits of the user. Recovery of deleted data from public user images was possible even from the AMI images. In addition to analysing a large number of AMI instances from Amazon and finding similar results, Zhang et al. [162] also analysed the cost and effectiveness of exploiting and defending prevalent vulnerabilities in traditional and cloud environments. An example of one of the attacks they tried on AWS is when they ran Metasploit[2] with the number of packets sent to the target VM (running Apache server) for a DoS attack. This resulted in crashing the Apache server running in all the targeted VMs. This could have been avoided if the OS was patched.

The fact that vulnerabilities could be found in VMs instantiated from public images is due to either:

1. *Purely patched operating systems:* In most OSs, vulnerabilities are discovered after a while. If the OS is not patched, it is just a matter of exploiting that particular vulnerability for the malicious intruder to get access to or destroy the VM.

2. *Malware left intentionally in the VM:* : This could lead to the creation of backdoors and bots that call home (the attackers machine), and allow VMs to be controlled remotely.

Securing the host OS and the guest OS against malware infections is very important. However, since antivirus products running in virtualised environments use agents to scan each VM instead of the individual instance of the product, this can slow the performance of a VM by creating antivirus storms [48].

Another dimension of complexity is that the security configuration for large numbers of VMs of a consumer can be very difficult. In the public IaaS cloud, the consumer has to be aware all the time of newly discovered vulnerabilities in the OSs running in the VMs and to follow the procedures to patch them. Soundararajan and Anderson [140] surveyed the various operations performed in different datacentres, including VM patching operations, and found that this operation may occur a lot. According to them, patching a VM requires the VM to remotely mount an ISO image that is located

[2]Metasploit: Penetration Testing Framework https://www.metasploit.com/

on the patch server. Obviously, in the case of a consumer renting hundreds of VMs from a public cloud provider, it is his task to handle patch management procedures.

3.4.11 Insecure hosts

While patching VMs is important, patching the physical host is far more important. In virtualisation, if the host is not secure, then the VM is not secure [33], even if the latter was patched effectively. The responsibility of patching the physical host is upon the cloud provider. Sometimes, this requires shutting down the host and affecting the availability of the VMs. However, if the cloud provider supports live migration, VMs can be moved to another healthy host without having to stop its operation – e.g. as the case with Google [58].

Razavi et al. [115] demonstrated an attack which allows an attacker to completely compromise co-hosted cloud VMs with relatively little effort. They developed a technique called "Flip Feng Shui", which manipulates deduplication operations that many cloud hosts use to save memory resources by sharing identical chunks of data used by two or more VMs. As a result, the attacker could gain full control over the target VM.

3.5 Availability Challenges

Availability is one of the main information security principles. Because cloud consumers data will be kept on the provider's premises, it is important that the service will be available when needed. It is not uncommon for cloud services to have gone off in the past, and also during these days. In [13] and [113], some famous cloud services outages were surveyed. DoS attacks can cause availability issues. Botnets are networks of bots (infected machines) used by malicious users for targeted attacks, such as DDoS (Distributed Denial of Service) attacks. In this scenario, attackers utilise a huge number of machines to cause the attack. Cloud computing services and the unlimited processing power they offer is a cheaper alternative to using botnets. In addition, it is difficult to detect cloud bots because of the transient nature of clouds [34]. For example, if a VM is used for an attack, it can be destroyed immediately afterwards. Bloomberg News reported that the hackers who breached Sony's PlayStation Network and gained access to sensitive data for 77 million subscribers used AWS to launch the attack [9]. Another example of availability challenges is when IP addresses of the consumer machines are

blacklisted for sending SPAM. In this case, cloud services cannot be accessed until the IP addresses are white-listed again [33]. This may also lead to a denial of service where consumers may not be able to use cloud services.

3.6 API & Browser Challenges

In general, most security concerns which are facing software applications also affect the cloud, especially in the SaaS model. Some of these issues are: viruses, password brute forcing, phishing, SQL injection, etc., [33]. The SaaS consumer interacts with the cloud provider using a web browser. A vulnerable web browser may cause threats from the consumer side. There has been extensive research on browser security such as [12, 75] where the focus was on the vulnerabilities which exist in web browsers and the scripting languages. Cloud computing providers a set of software interfaces or APIs that customers use to manage and interact with cloud services. An example of these APIs is Amazon's API tools which allow the cloud consumer to register and launch VM instances. Another example is Google SQL API. The Cloud Security Alliance (CSA) argues that the security and availability of general cloud services is dependent upon the security of these basic API tools [60]. Furthermore, some third party companies often build upon these APIs to provide value-added services, which introduces additional complexity [60].

3.7 Legal & Auditability Challenges

Cloud computing has brought a debate on its legal and auditability aspects. In general, laws which govern the use of information technology (IT) and electronic data take a long time to be implemented. In cloud computing, providers tend to store the data in different regions for the purpose of backup and disaster recovery. Those regions may have different jurisdictional laws. Some regions may not have IT-related laws at all. The absence of these laws could affect the security and privacy of data. Having different non standardized legalisations may lead to the same thing. Ruan et al. covered this issue in detail [124]. Some cloud providers, such as Amazon, allow customers to choose the region they would like their data to reside in [120]. Auditability, according to some regulations, must be provided in order for corporate data to be moved to the cloud. According to Chen and Katz [33], mutual auditability needs to be addressed in order to distribute the blame between cloud consumers and the providers when something happens. This, has to be addressed clearly in the SLA. De Chaves et al. highlighted

the SLA perspective in security management for cloud computing by providing an insight into the security metrics which need to be addressed in the Sec-SLA between the cloud provider and the consumer [40].

In reality, even though there may be SLA terms which state that the privacy of consumers' data is protected and assured, government laws may force cloud providers to disclose or turn down cloud services run by consumers. A famous example is the WikiLeaks website case. To resist a DoS attack on WikiLeaks servers, they moved all their data to AWS which resisted the attacks. Amazon was forced by the US government to terminate all WikiLeaks operations on AWS [152]. Regardless of the authenticity of what WikiLeaks was doing and the fact that it was not welcome, as a cloud consumer, this shows that data is not entirely under the cloud provider's control; governments may exert power to control the cloud provider.

3.8 Relationships Between the Attacks

From the literature, it appears that some of the attacks and challenges highlighted above are related to each other, either directly or indirectly. Figure 3.1 illustrates some of the observed relationships. For example, the figure shows that the unnecessary migration of VMs caused by the over-committing attack, may lead to co-residing of VMs next to the target VM. In turn, this could cause side-channel attacks and sensitive information about VMs will be leaked. Co-residing with other VMs in the public IaaS cloud could result in the resource freeing attack where VM resources can be wasted unnecessarily, which in turn can lead to the DoS attack due to VM starvation. The DoS attack can also be caused by poor isolation of VMs, which enables VMs to hop from one server to another. The poor isolation of VMs can cause VM escape, which can also be caused as a result of an insecure guest OS. These relationships are derived based on the literature and on understanding how the various attacks work. The relationships in the figure are not comprehensive and can be extended to include more links. We argue that understanding the relationships between the attacks can help in deterring them.

3.9 Addressing Cloud Security & Privacy Challenges

As discussed in the previous sections, there is a considerable amount of security and privacy challenges in IaaS public clouds which stem mainly from the multi-tenancy and co-residency features of the cloud computing model. Some of the issues can be

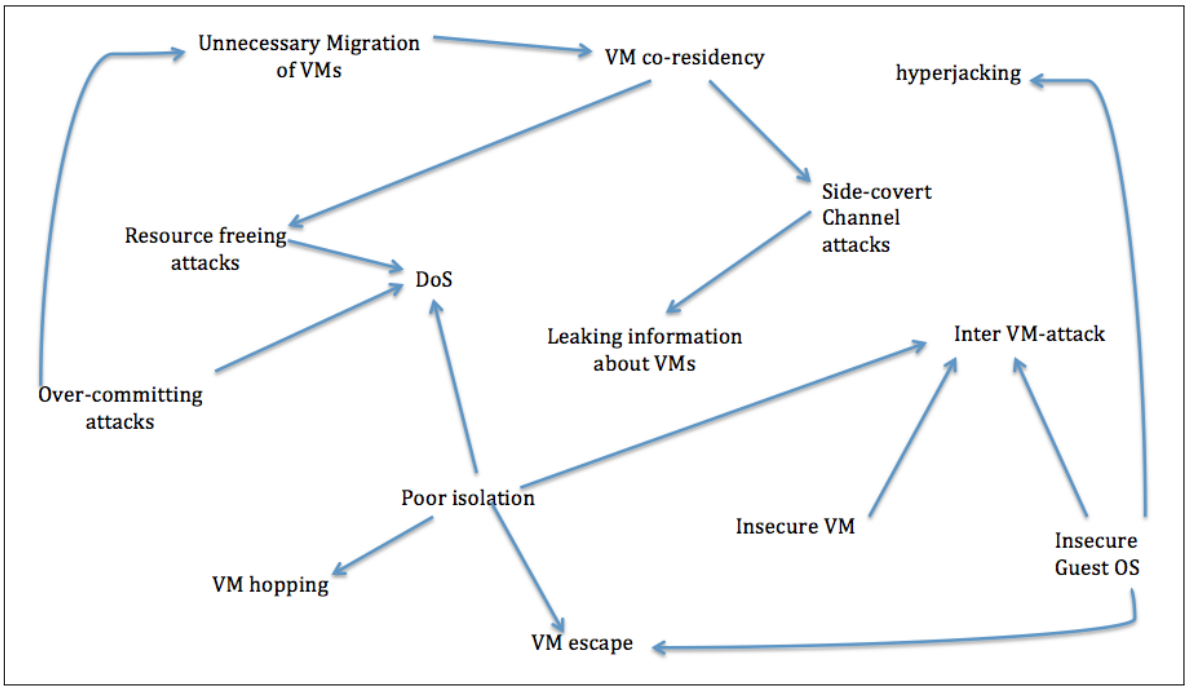

Fig. 3.1 Relationships between some of the attacks)

severe but have a low likelihood of occurrence, e.g. hyperjacking [136]. Yet, security measures and solutions have to be implemented to deter all kinds of potential attacks. Chen and Sion [35] stated that "the whole point of cloud computing is economy", so solutions to address such issues have to be economic and practical. Researchers have approached the challenges from different angles. In the following, we briefly summarize some of the approaches to address security and privacy challenges in the cloud:

- **Patching systems**: Attackers can exploit vulnerabilities either in virtual machines or in the virtualisation environment in general, including hypervisor vulnerabilities. Thus, for the consumer, it is important to regularly apply security patches to the VM. The same applies to the provider, who should patch all software systems, especially hypervisors and OSs. There is much research which covers the importance of patching [140, 162, 166]. Patching may require the physical machines which host the VMs to be terminated temporarily until maintenance is done. Luckily, in cloud centres which support live migration, VMs can be migrated to another healthy host without any downtime. However, since the VM also needs to be patched, a consideration has to be made on whether patching of VMs can be done online or offline. Zhou et al. [166] argued that patching VMs while they are running is unpredictable and creates performance challenges and high operational costs. They introduced a tool called Nüwa that enables efficient

and scalable offline patching of dormant VM images by analysing patches and converting them into offline patches. Patching cannot solve issues related to the misuse of the VM as it is intended only to fix software vulnerabilities.

- **Data encryption:** As discussed before, consumer data is susceptible to disclosure by unauthorised parties in a variety of ways. Even if the data was deleted, it was shown that it is possible to recover data using recovery tools. A question can be asked about whether encryption of files or VMs may solve the problem of data privacy. Martucci et al. [95] argue that cryptographic techniques are essential to provide information separation and data confidentiality in a cloud environment. Data may be encrypted at rest in the cloud provider's storage or in motion while it is being used by the consumer. Consumers are recommended to encrypt the data before moving it to the cloud. There are a variety of commercial products which offer encryption of files before moving them to the cloud, such as BoxCryptor[3], Viivo[4], etc.

  When performing the encryption, a key to encrypt the data will be generated and the same key can be used for decryption. In the agreement between the provider and the consumer, different levels of data encryption may be requested, and so the encryption process will be done by the cloud provider [70]. It is often debated whether the keys should be kept with the provider or the consumer. If the provider keeps the keys, then sensitive data belonging to consumer could be viewed by this provider. On the other hand, if the consumer loses the keys, he will not be able to access his own data. So, managing and securing the keys is one of the issues related to using encryption in the cloud environment. According to Gartner, if the keys are managed by the cloud provider, then businesses should require hardware-based key management systems within a tightly defined and managed set of key management processes [91].

  Trend Micro argued that encryption with policy-based key management can help ensure that data is only accessed in permitted locations to abide to existing privacy regulations [98]. Many research papers have focused on encryption in the cloud environment. Hwang et al. [70] proposed a business model for cloud computing based on the concept of separating the encryption and decryption

[3]BoxCryptor https://www.boxcryptor.com/en
[4]BoxCryptor https://viivo.com/

service from the storage service. The idea is to keep the cloud provider, which provides the storage, away from the encryption process to avoid potential data confidentiality breaches from the provider's side. A novel security scheme for encrypting virtual disk images in the cloud computing environment was introduced by Kazim et al. [78].

In 2009, an IBM researcher managed to achieve what is called "Fully Homomorphic Encryption", which makes the deep and unlimited analysis of encrypted information possible without sacrificing confidentiality [55]. Martucci et al. [95] argued that this encryption technique may solve, in principle, some problems, but it is still rather unexplored and untested in practice. Mukhopadhyay et al. [101] proposed a file encryption mechanism using AES, which is an encryption algorithm. Their methodology allows access to the encrypted data based on successful authentication.

Although encryption of cloud files is essential and has many benefits, it has some drawbacks as well. Data encryption restricts the consumer's ability to perform keyword searches, and thus makes the traditional plaintext search methods unsuitable for cloud computing [86]. Another issue is the effect on performance, as the encryption and decryption processes require complex computation power. Chen and Sion [35] provided estimates for the cryptographic costs of some encryption algorithms. They argued that making use of cloud storage as a simple remote encrypted file system is unfeasible if considering only core technology costs.

- **Secure Isolation:**There are some approaches which focus on prevention of attacks by strengthening the isolation software. Examples include sHype and Qubes OS. Shype is a secure hypervisor architecture by IBM which aims at providing functions such as: strong isolation, mediated sharing and communication between VMs [129]. Qubes OS [127] is a security-focused desktop operating system that aims to provide security through strong isolation. It exploits the isolation capabilities of the baremetal hypervisor (Xen), together with modern hardware technologies, such as Intel VT-d and Trusted Execution Technology. A question remains on how consumers will benefit from such solutions which rely mostly on cloud providers.

- **Detecting co-residency**: In the discussion of co-residency, we have explained that attackers could misuse this feature by spying on another VM using side channel attacks. There is research which argues that verifying exclusivity of host usage may enhance the trust between the cloud provider and the consumer [7, 164]. While it can be considered one of the solutions which can be used by the consumer to overcome some of the trust issues, it can also be an attack that may affect the privacy of other innocent consumers.

- **Information security management:** Similar to how information security challenges are faced in some classic IT environments, cloud security challenges can be managed using management frameworks or procedures. Cloud consumers are encouraged to carry out a security risk assessment before shifting their services to the cloud, in order to find potential risks and to decide a mitigation scheme or just accept the risk. The European Union Agency for Network and Information Security (ENISA) developed a cloud computing risk assessment report which can help organisations realize the various risks to be considered [30]. Popović and Hocenski [111] argued that a formal information security risk management process should pro-actively assess information security risks, as well as plan and manage these risks periodically or when needed. This goes well with Schneier's view of information security, as he considers it "a process, not a product" [132]. This means that even if there are products used to assure the security of data, without proper management of the whole process, attempts to secure data assets will fail. The Cloud Security Alliance (CSA) also produced a report which listed a number of important threats to cloud computing. Their document can help consumers assess the various risks and decide whether they can tolerate them or whether they should implement strategies to combat them. ISO Standards such as ISO/IEC 27017:2015 [105] and ISO/IEC 27018:2014 [104] are implemented especially for cloud services and can help consumers manage the various potential risks using the suggested controls.

- **Monitoring:** In the IaaS cloud, monitoring has been looked at and approached from different perspectives and for many purposes. Since utilization of physical servers is one of the main aims of virtualisation, tools to monitor the performance and utilisation of servers were implemented. For the consumer who is renting virtual machines, there are also some solutions to monitor performance metrics and provide alerting mechanisms, as in AWS's Cloud Watch[5]. Some of the

[5]Cloud watch https://aws.amazon.com/cloudwatch/

monitoring approaches target monitoring API use. For example, it monitors and records AWS API calls for the user account and sends logs to the consumer; as in Cloud Trail[6]. Most of the described monitoring tools do not consider what happens inside the VMs themselves. In another words, they do not tend to capture or record malicious events which occur in VMs. While some monitoring techniques, such as intrusion detection systems (IDSs), have been extensively researched for decades, their application in the cloud still faces some challenges. According to Lonea et al. [90], IDS sensors have the limitations that they yield massive amount of alerts and produce high false positive rates and false negative rates. Since VMs hosted in the cloud have limited capabilities, a more focused monitoring is required to ensure that computing resources are not heavily affected. Most approaches to implementing IDS solutions for cloud services have been from a cloud provider's perspective such as in [8, 50, 121] . Generally, this requires implementing IDS on the hypervisor level or the host (where IaaS cloud consumers have no control over). In [123], Roschke et al. argued that cloud providers need to enable possibilities to deploy and configure IDS for the user.

This thesis considers *monitoring* as a solution to some of the security and privacy challenges facing VMs hosted in IaaS public clouds. Specifically, it looks at achieving a focused monitoring of the malicious events that occur in the VM in order to inform the consumer/owner about them. Keeping the consumer informed about these malicious events as they occur could help the decision-making process. In the following sections, we cover monitoring in the cloud in more detail.

3.10 Monitoring and accountability in the cloud

The need for monitoring capabilities and accountability in cloud systems has been highlighted in many research papers. This occurs due to a lack of transparency of operations in many cloud providers [159, 125, 128]. There are many motives for monitoring and accountability in the cloud including (but not limited to): performance checking, anomaly detection, SLA enforcement, checking location of services, forensic analysis, etc. Chow et al. [36] indicate that data owners are often interested in understanding how their data is being managed by the provider in order to better understand potential for abuse or leakage of data. It is noted in [125] that one way to address this issue is by making the cloud more accountable.

[6]Cloud Trail https://aws.amazon.com/cloudtrail/

An approach to a more accountable cloud is discussed by Gehani et al. [54]. They argued that the cloud can be more accountable by proactively collecting potential forensic evidence, and they highlighted the steps towards achieving this. An architecture to monitor personal data transfers in a cloud system is highlighted in [106]. Rong et al. [122] argued that there is a need for further work in accountability in public clouds in order to provide transparent services that can be trusted by all users. The Cloud Accountability Project (A4cloud) focuses on the accountability for cloud and other Future Internet Services[7]. It aims at increasing trust in cloud computing by devising methods and tools to enable stakeholders apply accountability for the information held in the cloud.

Most of the research into the issue of accountability in the cloud has focused on the entire cloud infrastructure including not only the virtual disks of customers, but also the potential logs generated by a provider. In addition, the main driving factor in such approaches has been the need to collect evidence from prior incidents. Our approach attempts to support accountability by monitoring events from inside the VM, primarily as an evidence trail that might give greater confidence to the customer.

3.10.1 Provenance of data in the cloud

Provenance generally refers to information that "helps determine the derivation history of a data product, starting from its original sources" [135]. In [1], a distinction between provenance and log-file-based data is made, where logs provide a sequential history of actions usually relating to a particular process. In that sense, logs can be a source for generating provenance data [2].

In [128], a provenance logging system was highlighted along these challenges and benefits. However, this work focused on a provenance system which includes the cloud provider as well in a step towards supporting accountability in the cloud. Sources for collecting logs from the cloud computing infrastructure was highlighted in [125]. Abbadi et al. [1] argued that the key elements in the foundation of such a provenance system are: standardization of log records, availability, structured and centralized repository of data, security of logs, and trust establishment. CloudTrail is a web service that records AWS API calls for the account holder and delivers logs to the consumer. Such a data trail helps a consumer better understand the operations that were carried

[7] http://www.a4cloud.eu/

out on their VM instance. Our approach in this thesis is similar, but it extends the functionality to what is happening inside the VM and not just operations that take place on general cloud resources.

3.10.2 Monitoring VMs from the outside: VM introspection (VMI)

Garfinkel and Rosenblum first used the term VM introspection and developed an architecture to analyse the memory of a VM [50]. VM introspection involves examining the active state of a VM during execution, including memory or disk [117]. Introspection techniques attempt to bridge the *semantic gap* caused when the hypervisor cannot see what is inside a VM because of the isolation imposed by virtualisation technology. An example VMI tool is the *VMST* system, which supports the generation of VM introspection tools [47]. Also, *XenAccess* introspects both the memory and disk of the VMs and infers file creation and deletion events [107]. In [8], VMI was used to monitor system calls inside VMs in order to detect some attacks, such as: DoS and malware infection. Pathogen [121] is a system which can be used to monitor multiple VMs through introspection. Hizver and Chiueh [68] introduce a kernel data structure for supporting VM monitoring, which makes use of the *Volatility*[8] framework to simplify or automate the analysis of VM execution states.

In general, VMI has many applications including:

- ***Malware Detection:*** Since VMs are usually exposed to the internet in order to be used by customers, they could be vulnerable to malware attack just as normal physical machines would be. Many papers have covered the use of virtual machine introspection to detect malware such as [131, 22, 121]. Robert et al. [121] developed a system called Pathogen to monitor multiple VMs against malware infection. They tested the system by injecting a known malware to a VM running Windows 7 and *Pathogen* detected the creation of some malicious files.
- ***Intrusion Detection:*** VMI is useful in detecting and reporting such intrusion incidents by monitoring the memory in real time and inspecting any suspicious behaviour. Many research papers have touched on this topic in detail [8, 16, 50, 84].
- ***Checking Against PCI Compliance:***

[8]https://code.google.com/p/volatility/The Volatility Framework

Credit cards are very important in financial transactions. Merchants usually use credit card payment systems to process customers' transactions. Because of that, the merchant's system may become a target for intruders who aim at stealing card details. For this reason, the PCI (payment card industry) has set a mandatory pre-requisite security compliance standard[9] for businesses that process credit cards . Specifically, they are interested in the credit card data flow or how credit card details travel through production servers. VMI has been observed as a valuable technique to check virtual servers against PCI security standards. Hizver and Chiueh [67, 68] designed an introspection tool to automate the task of identifying credit card data flow in commercial payment systems running on virtualized servers hosted in private cloud environments. In [68], the introspection tool inspected the memory spaces of the communicating processes for the card data. Once the processes whose memory contained credit cards data are found, the machines involved in the credit card flow are identified.

- ***Detecting DoS Against the Host:***

  Availability is one of the essential principles in information security. Cloud providers strive to reduce the risk of threats which could cause availability issues. Because most cloud providers employ an automatic load balancing mechanism to move VMs across physical servers for utilization purposes, an attacker could lure the system into migrating VMs without a real need for that. This could result in wasting hardware resources or the disturbance of services. Thus, there is a need to detect DoS attacks coming from VMs, especially attacks which will not normally be detected by intrusion detection systems. In [8], a virtual machine introspection approach was described to detect DoS attacks in IaaS environments. Their method relies on monitoring system calls using machine learning techniques.

- ***File-level Integrity Checking:***

  Most VMI techniques focus on inspecting memory. It is also possible to inspect virtual disks. Despite the possibility, not much research highlights the introspection of virtual disks to monitor file-level updates in VMs without having to inject an agent inside them. An example is the work by Richer et al. [118] when they introduced a mechanism called Distributed Streaming VM introspection (DS-VMI), which can infer file system modifications from sector-level disk updates

[9]PCI Security Standards https://www.pcisecuritystandards.org

in real-time and stream them to a central monitoring facility. They provide external applications near real-time visibility into the file-level updates of a running VM guest. Their experiments showed that the performance overhead of this approach is modest except for write-intensive workloads; where the overhead is considerable.

- ***Application Whitelisting:*** Virtual desktop infrastructure gives users their own virtual desktop and manages all the VMs in a central manner. In this environment, it is important that only approved applications can run on VMs. For that, tools leveraging VMI are used to govern the execution of applications on virtual servers. In [68], a tool was implemented to check an executable file or a library module against a whitelist of known good programs before it was loaded into the address space of a user process, and aborted the program load operation if the executable file or library module was not on the white list.

Significant work in this area therefore attempts to inspect and monitor VMs from outside, trying to infer events that have taken place within the VM. This usually requires control from the provider side and traditional VMI cannot be done by the consumer since he has no control over the physical server hosting the VMs. Our approach is based on monitoring events as they occur inside the VM while keeping the collected event data secure. Therefore, we use introspection techniques but directly from inside the VM. We argue that this gives an increased visibility on the events that occur in the VMs. Besides, it eliminates the need to bridge the semantic gap. The work in [118] is closely related to our work in the sense that it provides file-level monitoring of events. However, their approach is agentless and it is mainly used to meet quality of service constraints from the perspective of a cloud provider. In addition, we do not only focus on file-level updates. Our approach makes use of VMI techniques for developing a taxonomy of security-related events. Also, it makes use of log file analysis techniques to find useful and informative patterns of particular security concerns.

3.10.3 Monitoring VMs from the inside

The classic way of monitoring physical servers has been to observe the logs generated by various system components. Applications running in the servers can also contribute application-specific logs which can be monitored and analysed. The generated logs can be very complicated and difficult to make use of directly. Because of that, on top of these logging systems, there are other tools which facilitate searching these logs and that provide alerting mechanisms. These tools may reside on the same local host or in

another remote central server which gathers logging data. Just like physical servers which run operating systems, virtual servers also run full OSs. The generated logs on the VMs can be a valuable source of information. In guest OSs which run Linux, for example, many of the valuable logs are stored in `/var/log`. Currently, there is a lack of tools which can run on VMs and make use of the generated logs in an informative manner; i.e. easier to understand and less complicated for consumers. Most of the monitoring tools include software agents which gather the logs as they are generated in the VM and ship them to a central processing server. If the agent sends the data to the server, the collection method is called "push". If the server *pulls* the data from the monitored hosts, the collection method is called "pull". Once logs are received on the central server, they are either displayed as they are or processed and analysed. Some of the tools contain plugin APIs which enable specific application platforms to make use of the received data. For example, a tool may provide a plug-in to load the data directly into an SQL database. There are an extensive number of such monitoring systems which include but are not limited to:

- **Collectd**[10]: It is an open source tool for collecting monitoring state which is highly extensible and supports all common applications, logs and output formats. Many cloud providers use Collectd as part of their own monitoring solutions, including Rightscale[11]. Collectd uses a push model, where the monitoring state is pushed to a multicast group or single server. It has a client/server mode where monitored VMs can be configured as clients. It contains an extensive number of plugins which enables data to be pushed to many storage services and applications including Redis, MongoDB and MySQL. Collectd is concerned mostly with the collection and transmission of monitoring state. It does not perform any actual monitoring of system or analysis of the logs. As mentioned, this can be performed by other applications and front-ends using the plugins.

- **Splunk**[12]: It is a commercial software tool that is used to analyse a large set of system logs. It allows a distributed analysis of logs gathered from remote hosts using the concepts of *forwarders* and *indexers*. Forwarders are splunk agents deployed inside remote machines and configured to send selected logs for monitoring. Indexers are the machines which receive logs and perform the analysis or search on them. Similar to Collectd, Splunk does not perform actual monitoring on monitored client machines. It just collects the data from the client

[10] https://collectd.org/
[11] http://www.rightscale.com/
[12] http://www.splunk.com

and sends them to the server (indexer). The indexer performs the analysis and provides alerting services. Splunk facilitates the visualisation of data.

- **Logstash**[13]: It is an open source, server-side data processing pipeline that ingests data from a multitude of sources simultaneously, transforms it, and then sends it to other applications in order to perform search and alerting tasks. Thus, Logstash is only concerned with shipping logs and event data from clients, then processing and formatting them. It does not collect metrics, although it does provide a plugin for Collectd in order to send metrics. Logstash is part of the ElasticSearch family, a set of tools intended for efficient distributed real-time text analytics. Kibana[14], another tool in the ElasticSearch stack, is responsible for analysing and visualising the collected logs' data. Similar to Collectd, Logstash runs a small agent on each monitored client machine, referred to as a *shipper*. The shipper is configured to take inputs from various sources (stdin, stderr, log files, etc.) and then "ship" them an indexer. The indexer parses and categorise logs in the pipeline using filters. Later, parsed log data can be exported to other tools for further analysis. Logstash uses the *pull* method for collecting the logs.

- **Stackdriver**[15]: It is an intelligent commercial monitoring tool for AWS, Google Compute Engine and Rackspace Cloud that provides resource monitoring and anomaly detection. It pulls data from monitored VMs and provides a dashboard for administrators to monitor the resources and some logs. It allows for the configuration of alerts and notifications. By default, Stackdriver is configured to stream specific logs types but it can be configured to stream others. Since it runs as a service in the cloud, consumers cannot run Stackdriver locally on their physical servers to pull logs from monitored instances.

- **Riemann**[16]: It is a monitoring tool that aggregates events from hosts and applications and can feed them into a stream processing language to be manipulated, summarized, or actioned [147]. It is mostly concerned with processing network metrics. Data is pushed to Reimann and on receiving an event, Riemann processes it through a stream [152]. Stream functions can handle events, merge streams, split streams and perform various other operations. Through stream processing, Riemann can check thresholds, detect anomalous behaviour, raise

[13]https://www.elastic.co/products/logstash
[14]https://github.com/elastic/kibana
[15]http://www.stackdriver.com/
[16]http://riemann.io/

alerts and perform other common monitoring uses [154]. It can be configured to send emails if certain events occur.

- **Munin**[17]: This is a networked resource monitoring tool that can help analyse resource trends. It can monitor the performance of various client machines by remembering the monitoring state and comparing it in order to help administrators make decisions. It uses the pull method to collect resource metrics. Thus, it does not perform actual monitoring on the monitored hosts.

- **Monit**[18]: This is a utility for managing and monitoring processes, programs, files, directories and file systems on a Unix system. It is particularly useful for monitoring daemon processes. If the process does not respond, Monit can be configured to restart it automatically. Monit can be used to monitor files, directories and file systems for changes. It generates its own logs and it can provide alerting features. The user can access the GUI via a web browser. Since Monit operates locally on monitored machines, there is no support for pushing the events/data to a central server. All the analysis is performed locally.

- **Varanus**[153]: This is a peer to peer monitoring tool designed for monitoring large scale cloud deployments. Designed to handle scale and elasticity, Varanus makes use of a k-nearest-neighbour-based group mechanism to dynamically group related VMs and form a layered gossip hierarchy for propagating monitoring state using the push model [154]. By itself, Varanus does not perform any actual monitoring on the monitored instances, but it does provide analysis of the collected data. However, its main focus is on collecting metrics. The collected metrics are comprehensive; it includes all the metrics which the Collectd tool can gather. In the implementation, it uses *Collectd* to collect the monitoring state. In [153], Varanus was compared to *Nagios*[19] which is another monitoring tool specialized in monitoring IT infrastructure.

There are many other monitoring tools similar to the aforementioned tools. In our work, we focused on monitoring only security-related events from VMs. Thus, monitoring resource usage or performance metrics fall beyond the scope of our research. Our tool, *VMInformant* performs customized monitoring by allowing the consumer to decide the level of monitoring and the type of monitored events. This is mainly to

[17] http://munin-monitoring.org/
[18] https://mmonit.com/monit/
[19] https://www.nagios.org/

minimize the performance and storage overhead that may result from the monitoring process. Our monitoring tool generates its own logs which are formatted in an informative and easy to understand way. VMInformant's logs from multiple VMs are pushed to a central server for further analysis. Thus, unlike most other tools, our tool performs actual monitoring on VMs and ships the data to a central sever for further analysis. This server is called the inspector station (see Chapter 6). The inspector station performs analysis of the collected refined logs to find patterns across the cluster of monitored VMs. One thing that the VMInformant architecture is concerned about is monitoring VM lifecycle events, particularly migration events. None of the other surveyed monitoring tools detect migration events. In addition, it provides alerting features when specific events are encountered. Table 3.1 provides a comparison between the discussed monitoring tools and our work. The following chapter, covers the architecture and the reference implementation of VMInformant in greater detail.

Table 3.1 Comparisons between some monitoring tools against our work

Monitoring tool	**Collection Method**	**Actual Monitoing in clients**	**Monitoring focus**	**alerting**	**event analysis?**	**VM life-cycle events**
collectd	push	no	comprehensive	no	no	no
splunk	push	no	comprehensive	yes	yes	no
Logstash	pull	no	comprehensive	yes	yes	no
Stackdriver	pull	no	resource usage	yes	no	no
Riemann	push	no	network metrics	yes	yes	no
Munin	pull	no	network/ performance metrics	yes	yes	no
Monit	local only	yes	monitoring running services	yes	no	no
Varanus	push	no	comprehensive	yes	no	no
Our work	push	yes	security-related events	yes	yes	yes

3.11 Literature summary

Table 3.3 provides a summary of some of the key research papers from the literature. Most of papers focused on specific cloud security challenges. The categorization of the papers was based on many features. Due to the vast number of compared features, Table 3.2 provides the list of keys to the features to make it easier to compare literature sources. In this chapter, we have covered an extensive number of literature sources. However, the table only highlights the major ones.

Table 3.2 Key to the table of the literature survey

Feature	Key	Feature	Key
Hyperjacking	H	*Insecure management console*	C
side channel attack	S	*Technical approach?*	T
Denial of Service	D	*Management approach?*	G
VM escape	E	*Demonstrating an attack?*	J
Inter-VM/VM Hopping	I	*Experiment platform*	X
live migration attacks	L	*Patching as solution?*	P
Remnants of data	R	*Encryption as solution?*	Y
VM sprawl	V	*Monitoring as Solution?*	M
Guest os-related attacks	O	*Info sec Management solution?*	Q
Insecure hosts	Z	*concept/Tool name:*	N

3.12 Chapter Conclusion

This chapter highlighted some of the security and privacy challenges of cloud computing with a focus on virtualisation security. The discussion of virtualisation security takes into account the whole virtualisation platform, including the host, the hypervisor, guest OS and the VM as a whole. We conclude that there are many possible ways that attackers could use to get to consumer's data. While some of the explored attacks in literature were demonstrated in a laboratory setting, they can also be applied to real public cloud systems. For example, reconstructing private keys of a VM from a co-resident VM was tested in laboratory setting environment. Attackers could apply the same attack, which utilizes side-channels, on a public cloud, thereby affecting the privacy of consumers. Some attackers exploit useful features of the cloud to create attacks. For example, they might exploit the features which allows providers to load balance requests on their infrastructure by triggering migration of VMs unnecessarily. This may lead to freeing resources of the physical servers hosting consumer's data. By the studying the attacks on the virtualisation platforms, we found that some of the attacks could be related; one might cause another. For example, the unnecessary migration of VMs caused by the *over-committing* attack, may lead to *co-residing* of VMs next to the target VM. We illustrated the potential relationships between the attacks in Figure 3.1. We also highlighted some of the approaches used to address the challenges and explained that we use monitoring of malicious events as the main solution in this thesis. We argued that there is a need for monitoring capabilities and accountability in cloud systems. This occurs due to a lack of transparency of operations in many cloud providers. We covered two approaches for monitoring: from the outside and from the inside. Monitoring VMs from the outside is usually performed from the perspective of the cloud provider; since they have control over the hypervisor. An example is a technique called: virtual machine introspection (VMI), where monitoring of VMs occur at the hypervisor level. Monitoring VMs from the inside is the classic method of monitoring which benefits from direct access to logs and operations in the

Table 3.3 Summary of literature Survey

	H	S	D	E	I	L	R	V	O	Z	C	T	G	J	X	P	Y	M	Q	N
Ray and Schultz [114]	-																			
King et al. [81]	✓											✓		✓						VMBR
Rutkowska [126]	✓											✓		✓	Win Vista					Blue pill
Skapinetz [136]	✓																			
Dai Zovi [39]	✓											✓		✓	MacOS X					Vitriol
Anand et al. [11]																				
Luo et al. [92]				✓	✓			✓			✓									
Ristenpart et al. [120]		✓										✓		✓	AWS					
Zhang et al. [164]		✓			✓							✓								homealone
Harnik et al. [64]		✓										✓		✓	dropbox					
Bates et al. [21]		✓			✓							✓		✓	lab-based					
Younis et al. [160]		✓										✓								
Alarifi and Wolthusen [6]			✓		✓							✓			Eucalyptus			✓		
Varadarajan et al. [148]			✓											✓	AWS					
Gurav and Shaikh [63]				✓																
Shoaib and Das [134]	✓			✓																
Oberheide et al. [102]						✓						✓		✓						Xensploit
Jordon and Forshaw [76]							✓					✓					✓			
Sarathy et al. [130]								✓				✓								
Bourdeau [26]								✓					✓						✓	Reduce,Reuse,Recycle
Vaughan-Nichols [149]									✓											
Bugiel et al. [29]									✓					✓	AWS					
Balduzzi et al. [17]									✓					✓	AWS					
Zhang et al. [162]			✓						✓					✓	AWS	✓				
Soundararajan and Anderson[140]									✓				✓			✓				
Garber [48]									✓											
Chen and Katz [33]	✓	✓							✓	✓										
Zhou et al. [166]													✓			✓				Nüwa
Martucci et al. [95]													✓				✓			
Hwang et al. [70]													✓				✓			
Lowans and MacDonald [91]																	✓			
Gentry [55]																	✓			
Catteddu and Hogben [30]																				
Popović and Hocenski [111]													✓						✓	
Oliveira et al. [106]																		✓		
Rong et al. [122]																		✓	✓	

VM. We indicated that in this thesis, we follow this type of monitoring. We compared several monitoring tools which can be used in the cloud. We explained that our work is different because: (1) it offers a focused monitoring of security-related VM events; (2) unlike some tools, in our work, we perform actual monitoring and analysis of the events– not just collecting events from logs; (3) we focus on life-cycle VM events, i.e. migration of VMs. Finally, we conclude that while cloud computing has offered various advantages, many cloud security and privacy challenges still remain. The challenges need to be managed and addressed using suitable technical or management approaches.

Chapter 4

Monitoring Security-related VM Events

4.1 Chapter Overview

This chapter highlights our research on monitoring security-related VM events. It proposes our taxonomy of security-related VM events used to enable focused monitoring that does not affect the performance of the running applications in the VM. The taxonomy forms the basis for the design and architecture of a system to monitor VM-specific, security-related events. The system is called VMInformant. Specifically, the system monitors events which occur inside virtual machines and informs the owner about them. This chapter also discusses the reference implementation of *VMInformant* and its subsequent evaluation, and forms the basis of the work published in [5].

4.2 Introduction

As discussed earlier in Chapter 3, multi-tenancy and co-residency are two features of IaaS cloud systems. The first feature refers to the hundreds or thousands of consumers that may share the cloud infrastructure, while the second feature refers to the fact that a VM of one owner may reside on the same physical machine as a competitor. In Chapter 3 we highlighted that achieving co-residency of VMs in IaaS is possible, and so is verifying exclusivity of host usage of VMs– where cloud consumers can verify whether they are the only tenants on a physical cloud server or not. Vulnerabilities in VMs mean that they could be exploited in various ways by a malicious tenant, and the fact that the cloud provider has full control of the hosted virtual machines means that trust

between the provider and the consumer could be violated. Much research has indicated this trust problem and emphasized the importance of monitoring malicious events in virtual machines hosted in IaaS to catch *privilege based attacks* such as in [152, 7]. A rogue system administrator with root privileges on the VM hosts can undermine all security mechanisms and obtain access to other users' applications and data [152].

To overcome these concerns, it is necessary to keep the owner of a VM informed about what has happened to their VM once it has been deployed within a public cloud system. It is therefore not enough for a provider to offer guarantees about data and VM privacy or pre-negotiate an SLA with a consumer. A more pro-active process is needed to ensure that the consumer is kept aware of operations that are carried out on their deployed VM. We propose the monitoring of security-related VM events.

In general, the need to monitor security-related events in virtual machines hosted in public IaaS is motivated by a number of issues which include:

- **Limited transparency**: In general, all cloud service models suffer from a lack of transparency from the cloud provider side. They do not disclose how exactly they are securing the infrastructure or whether they have carried out risk assessments, or even the exact location of stored data. Hiding such information may be reasonable given the fact it could be exploited in a notorious manner by intruders. Still, trust cannot be fully given to the cloud provider.
- **Rapid discovery of attack vectors:** Every now and then, new vulnerabilities are discovered in operating systems – which run in guest VMs and on physical hosts. The search for possible novel attacks to the virtualisation platform is continuing, and even the cloud provider may not be ready for them. Thus, there need to be ways for consumers to find symptoms in their critical mission VMs which are hosted in the cloud.
- **SLA compliance**: Consumers pay per use of cloud services and guarantees are given by the cloud provider. If the consumer requires that their VMs be located in a certain region, then cloud providers must comply. Otherwise, consumers may choose to switch cloud providers to protect the confidentiality of their data.
- **Government intrusion**: Governments are increasingly exerting control on cloud providers, which may lead to unauthorized disclosure or tampering with stored data. This may happen without consumers noticing such events if there are no mechanisms to record and alert to this from the consumer side.

- **Separated security responsibilities**: In IaaS, consumers manage the security of their deployed VMs by themselves. They have to manage security patching of OSs and applications to make sure systems are free of vulnerabilities which may lead to intrusions. In some ways, monitoring security-related events which occur in the VM helps support this, but it does not replace the need for other security counter measures. What it does is complement the consumer's effort to secure VMs.

Defining a malicious activity or event is debatable, but in our proposal we create a taxonomy of virtual machine security-related events to help design security-focused monitoring tools. Usually, operating systems running in virtual machines generate logs containing millions of events. Some of these events are related to system performance, application execution, etc. Not all events are security related. To the best of our knowledge, no such taxonomy that focuses on security-related VM events exist. Most of the existing monitoring approaches provide general monitoring which may not be suitable for virtual machines. Besides, our taxonomy covers VM life-cycle events, e.g. migration events. These groups of events are not normally deemed as security-related. However, we argue that hiding them from the consumer could pose some privacy concerns. We believe that this contribution could be useful in designing VM-specific security monitoring systems. Virtual machines also have limited capabilities in terms of storage, processing power, etc. The role of the monitoring system is to capture only security-related events from the VM and aggregate them in a file in order to be sent to the owner. Therefore, it does not take actions on the events. Rather, it sends them to a central location for final analysis.

Based on the taxonomy that we propose, we present the design, architecture and prototype implementation of *VMInformant*, which is an agent-based monitoring system that sits in the deployed virtual machine and can be configured to record selected events which occur within a VM. The consumer/VM owner is therefore able to achieve better visibility on how their data is being accessed or processed – thereby giving them greater potential trust that their VM or data has not been compromised. The system stores the events in an encrypted area so that only the owner can access the data. It allows the consumer to choose the level of detail at which events are recorded. The selected events are valuable from the perspective of the VM owner/consumer as they may indicate malicious activity, whether this activity is the consequence of an intruder or malware. We believe that the proposed design and architecture of *VMInformant*

could see use in designing security-focused monitoring tools suited to VMs.

This chapter begins by explaining the problem in detail in a scenario-based approach, discussing research questions from which we derive requirements for the monitoring mechanism. We then explain the strategy which can be used for monitoring security-related VM events. Next, we cover the taxonomy of VM related-security events, outlining different event groups and types. We also introduce the architecture of VMInformant by highlighting components and the interactions between them. We then go into the reference implementation of VMInformant and show results of usage. Finally, we introduce the evaluation and the methodology behind it.

This chapter makes the following contributions:

1. Taxonomy of security-related VM events, which helps to achieve focused monitoring of VM security events.
2. The design and architecture of VMInformant and its subsequent prototype implementation.
3. A detailed evaluation of monitoring and detection of security-related VM events, in terms of performance, storage and usefulness.

4.3 Problem Specification

The focus of monitoring security-related events in VMs is on the Infrastructure-as-a-Service (IaaS) cloud delivery model, where a consumer deploys his own VM on a public cloud, hosting a web or a database server for instance. Consider the following scenario as an example. A European company makes use of an IaaS cloud provider to use either pre-configured virtual servers or attempts to deploy their own. They decide to use pre-configured ones. There will however still be additional configuration to be undertaken, but this is generally less time consuming than purchasing and installing a new physical server (in addition to the management of software and subsequent maintenance). They deploy three virtual machines: the database server, the mail server, and the web server. The IaaS cloud provider has multiple data centres in different continents, including Europe. The company requests that their data stays within Europe to ensure that the privacy concerns of European customers can be adequately addressed. The company realizes that the IaaS provider has data belonging to many

other tenants, from which one of them could be their competitor. In addition, since the data will be held remotely, they are concerned that confidential files may be read inadvertently or tampered with without their knowledge – particularly documents which contain sensitive customer and operational data. The company would be more confident if they had a mechanism which would allow them to record events which took place on their deployed VMs, such as read, written, created, and deleted operations on files within the VM. They would have some level of confidence and trust in the infrastructure if such visibility was available. However, they would still have concerns about the overhead resulting from the recording process – even if such event recording was possible – to ensure that these additional operations would not significantly impact the performance of the cloud system. Also, they would be uncertain about what to record and how often to address their security and privacy concerns.

This scenario attempts to clarify the particular research questions we are attempting to focus on in this work: (i) what monitoring/recording of events can be undertaken inside the VM; (ii) what level and granularity of information can be recorded; (iii) how to determine which events to record to address particular security/ privacy and trust concerns; (iv) is it possible to also detect events based on interaction with the hypervisor?; (v) what is the performance and storage overhead resulting from monitoring of such events? In particular, can we reduce the number of events we record (thereby reducing potential storage overhead) but still address particular security concerns of a customer? The scenario and the research questions also highlight requirements for the mechanism:

Secure: the event logs gathered by the VM monitoring infrastructure will be kept encrypted inside the VM so that only the VM owner can access the data using their private key.

Flexible: allowing a VM owner to decide the level of detail/granularity of monitoring required.

Economic: due to the VM virtual disk size, event data should not exceed the storage limit by which the VM can operate normally. In another words, the monitoring process should not affect VM operations.

Informative: the monitored events should be related to the security/privacy issues of a particular customer, i.e. it should address a particular requirement and not be undertaken randomly. Also, a consumer is able to specify (as a configuration parameter) the maximum size or time period for creating the event log, after which the system

sends the logs to the VM owner.

Given the scenario covered in the previous paragraphs and the requirements identified above, we consider the following events to be of interest:

- Has a particular folder/file in the VM been tampered with, deleted, moved, modified or reproduced?
- Have the permission bits of a particular folder/file(s) changed suddenly?
- Have any new users been added or removed from the system?
- Are there any failed logins using a certain user name?
- Are there any newly opened ports or suddenly closed ports that were expected to be open?
- Has the shutdown or reboot of the VM occurred due to a user or hypervisor intervention?
- Has the VM been migrated to new hardware or a new region (data centre)?
- Has the VM been snapshotted?
- Is there a malicious application running inside the VM?
- Is there a legitimate application running inside the VM which should not be there due to policies or regulations?
- Has a certain critical process been deactivated in the VM?
- Are there any USB or other devices connected to the VM during its life cycle?
- Who did what (which user?) and at what time?

4.4 Monitoring Strategy

To effectively monitor events within a VM which has limited resources, we argue that there must be a strategy based on the choices made by the consumer. We outline here some aspects of the strategy to be considered:

- Mode of detection: Whether detection of malicious events should be *periodic* or *continuous*. While periodic checking would lower the performance overhead, continuous monitoring and detection would increase it.
- Mode of analysis: Whether analysis of detected events should be local or central. Again, performing analysis of recorded events locally in the VMs may affect the performance, yet speed up the process.
- Storage location of events: whether recorded events are stored locally in the virtual disk before being sent to the owner, or stored *remotely* in another location, e.g. another storage cloud provider.
- Granularity: the level of detail at which events are recorded and the ability to follow a custom approach that is based on consumer requirements. This will lower the impact on performance and storage.

We argue that the monitoring strategy of VMs in IaaS should aim at reducing the number of recorded events (as much as possible), while at the same time addressing the particular security concerns of consumers. This way, performance and storage overheads could be reduced considerably.

4.5 A Taxonomy of Security-related Events for VMs

Based on the discussions in the previous sections which highlighted requirements for monitoring security-related events in VMs, it is evident that there is a need to specify security-related VM events from the perspective of the VM owner. Therefore, in this section we propose a taxonomy which involves selected groups of relevant events.

4.5.1 Event groups

Operating systems, in general, generate many log files which record a number of internally generated events. The volume of data stored in these log files can be very large and not all of it may be important when it comes to analysing security/privacy/trust issues related to virtual machines. Therefore, there is a need to create a taxonomy for these events as a basis to enable aggregation. Many research papers have touched on log analysis for checking events and on intrusion detection by analysing system calls, e.g. [118, 8]. However, none have covered a VM-specific security-related taxonomy of events. In this section we propose a taxonomy for such events (these are generic

and not operating system specific). To demonstrate how such a taxonomy would be used, we utilize events from Ubuntu Linux. We classify the events into six groups: File-centric events, VM life-cycle events, process events, user-access events, network-related events and attached devices events. In the following, each group will be discussed with examples. The list of events in each group is not comprehensive. Rather, it is meant to give an idea of critical events that should be recorded in order to determine actions that may be undertaken by a resource owner. Also, such an analysis is used as the basis for the design and implementation of *VMInformant*, as will be covered later.

File-centric events: This group of VM events is related to the folders or files being opened, updated, created, deleted or modified in the specified path. In a virtual server where critical services may be hosted, if critical files are newly created in a certain folder, deleted suddenly, copied, or even viewed by unauthorized people, this could indicate a possible threat. A mechanism to report or detect such events is required so that the owner of the VM can input the path containing the file/folder to be monitored. For example, if the critical folder contains seven files, three of which were opened, and one of them was newly created, details of the opened files and the time of creation will be logged. Monitoring file access and updates in this way provides a useful basis for understanding and logging of unauthorized access events. A number of tools which allow the monitoring of file level updates already exist. Tools that check the integrity of files are commonly called *file integrity checkers*. These types of file-level updates or manipulations are usually initiated by the VM by performing system calls which are propagated to a hypervisor. The hypervisor executes the system call instructions on the VM. If the files are encrypted, the same mechanism should detect attempts to brute-force a password.

VM life-cycle events: The hypervisor controls the lifecycle of the VM. Creation of the VM is undertaken either by the hypervisor directly or by using a public API (via a consumer application). The same is true for rebooting, pausing, resuming, shutting down, creating snap shots, and migrating the virtual machines. All these VM-lifecycle events can be carried out from the provider's site. This can expose the VM to availability as well as confidentiality and legal issues if the VM has been migrated to another region than the one specified in the initial agreement or where the VM was initially deployed. Biederman et al. [24] stress that if the VM is migrated across national borders during a live migration, internal data can become subject to different legislation. Furthermore, VMs can be susceptible to manipulation during the live

migration procedure [102]. We argue that a mechanism to detect or make transparent the event when the VM was shutdown, rebooted or paused by an entity other than the owner may be of a great importance. Also, reporting a possible "physical" migration of a VM can be useful to the owner and allows potential legal issues associated with data access to be known and responded to. Chapter 4 focuses on the detection of migration of VMs from the consumer's perspective.

Process-related Events: Normally, a malicious application may run new processes or deactivate critical processes in order to control the system. If there is no mechanism in place to detect this, the main user of the VM may not notice such attempts. Anomaly detection research such as [8] compares processes which execute under *normal* operation, compared to processes generated in a particular usage scenario. Some research also tends to find missing processes, as in [32]. We argue that the monitoring system should indicate to the consumer whether a malicious process has been created or a critical process deleted or deactivated. To achieve this, the system can use one of the widely available anomaly detection techniques in a way that would not cause impact on the routine processes.

User-Access Events: Usually, only a specified number of users may have access to virtual servers, especially critical ones. If a new user is added to the system without the knowledge of the VM owner, this could indicate a possible breach. Generally, when attackers gain access to a VM remotely, they tend to create new users with privileges. They can then perform malicious activities that will go un-noticed if there is no detection mechanism in place. Failed logins can also indicate a possible brute-force attack. A command such as: `grep ‘failed‘ /var/log/auth.log` can display the failed attempts. We argue that such events have to be reported to the VM owner.

Files and folders have many permission modes which identify allowed oper- ations. Changed permissions on files or folders within a cloud environment could indicate a potential attack or data breach. Richter et al. [118] identify that Google reported an outage in early 2011 that affected 15-20% of its production servers. This outage was caused by a permission change to a folder in the path of the dynamic VM loader. We argue that such events must be monitored in order to capture incorrectly set permission values within a user file space. A statement such as: `find /lib -maxdepth 0 -not -perm 755` can easily find misconfigured files and hence support secure user access.

```
taimur@taimur-VirtualBox:/var$ grep "failed" /var/log/auth.log
Apr 30 00:14:04 taimur-VirtualBox unix_chkpwd[3185]: password check failed for u
ser (taimur)
Apr 30 00:43:25 taimur-VirtualBox unix_chkpwd[3506]: password check failed for u
ser (taimur)
Apr 30 01:30:53 taimur-VirtualBox unix_chkpwd[3960]: password check failed for u
ser (taimur)
Apr 30 23:31:22 taimur-VirtualBox unix_chkpwd[6830]: password check failed for u
ser (taimur)
May  1 11:03:45 taimur-VirtualBox unix_chkpwd[8218]: password check failed for u
ser (taimur)
May  1 11:38:20 taimur-VirtualBox unix_chkpwd[12175]: password check failed for
```

Fig. 4.1 Illustrating failed login monitoring

Network-Related Events: Ports provides end to end communications for applications running locally and other services in the internet. This group of events in the taxonomy can indicate if certain ports have been incorrectly left open for remote access. There exist many open-source tools which allow the detection of open ports such as: Nmap[1], Nessus[2], etc. There are also some lightweight tools which just check status of ports. Similarly if a port that is normally used by a legitimate application is closed, this can cause availability issues and therefore should also be flagged.

Attached Devices Events: The ability to attach devices such as a USB memory stick to a VM should not be permitted, especially in critical VMs, due to the large attack vector it may impose. Attaching such devices may indicate a possible attempt to steal data or infect the VM. It is therefore imperative that such events are reported to the VM owner, to enable them to take any corrective action or provide an audit trail to potential customers who own the data in the VM. The file `/var/logs/syslog` could have information about connected USB devices. There are other places from where such information can be acquired.

4.5.2 VM security events

Table 4.1 shows a list of security-related events that we have identified, grouped according to the taxonomy introduced in Section 4.5 and according to their relationship with the CIA triad (Confidentiality-Integrity-Availability). From the table, we notice that most of the events affect the availability and confidentiality of VMs. We also note that some of the identified events are cloud- specific, e.g. life-cycle events, but some are

[1] Nmap https://nmap.org/
[2] Nessus www.tenable.com

not. However, they are still a major concern when it comes to the analysis of threats. As outlined in Section 4.8.1, only selected groups of events were gathered in our initial attempt to develop a reference implementation of VMInformant (1-7,17-19). Due to the importance and complexity of detecting the migration event from a consumer's perspective, we have dedicated Chapter 5 to highlighting our contribution.

4.5.3 Scope of the events to be monitored

It is important to note that the taxonomy of VM security-related events highlighted above is not related to the applications consumers run in the VMs. For example, applications such as SQL server or Apache server produce their own logs and event reporting tools. Some of the applications contain tools to manage the users who access them. VMInformant monitoring is focused on specific types of events which are mostly tied to operating systems (as a proof of concept, we focus on Linux Ubuntu OS).

4.6 The Rationale Behind VMInformant

Our proposed system, *VMInformant*, is a monitoring system that is embedded in virtual machines to monitor, record and alert consumers about malicious events that occur on their system/VM. In classic IT environments where physical servers are used, there exist systems which monitor such events for many purposes. Therefore, given that we are concerned about VMs in IaaS, it makes sense to compare these systems to determine where VMInformant can be positioned and why such architecture is needed.

4.6.1 Intrusion Detection Systems similarity

VMInformant can be compared to an Intrusion Detection System (IDS), which is a device or software application that monitors a network or system for malicious activity or policy violations. The most common categorization of IDSs in terms of detection methods are *signature-based* or *anomaly-based*. A signature-based IDS checks against pre-defined rules, policies or symptoms. It is also sometimes called a misuse IDS detection system or knowledge-based IDS. Anomaly or behavioural IDS usually employs learning techniques to determine the normal state of the system as a baseline from which to identify anomalies and raise alarm. Some systems use a *hybrid* approach which combines signature and anomaly detection methods.

Table 4.1 List of the event according to the taxonomy and their relevance to the C-I-A triad

Event Group	Seq	Event Description	Confidentiality	Integrity	Availability
File-Centric	1	A file/folder was created		x	
	2	A file/folder was modified	x	x	
	3	A file/folder was opened/viewed	x		
	4	A file/folder was deleted			x
	5	A file/folder was duplicated/copied	x		
	6	A file/folder was moved	x		x
	7	A file/folder had permissions changed	x	x	x
VM Life Cycle	8	The VM was started			x
	9	The VM was rebooted			x
	10	The VM was suspended			x
	11	The VM was resumed			x
	12	The VM was snapshotted	x		
	13	The VM was migrated	x		x
Process-related	14	A malicious process was run	x	x	
	15	A malicious application was installed	x	x	
	16	A critical process disappeared			x
User-access	17	Failed login attempts detected	x		x
	18	A new user account added to the system	x		
	19	user account is removed from the system			x
Network Related	20	A malicious port was opened	x		
	21	A Legitimate port was deactivated			x
Attached-Devices	22	A USB device was attached to the VM	x	x	

Table 4.2 shows a comparison between the two methods. While the anomaly-based IDS is able to detect unknown attacks (Zero day), signature-based IDS fails –as information about new attacks has to be added regularly to the knowledge base. Signature-based systems have lower rate of false negatives because they search for the exact pattern of attack, whereas anomaly-based IDSs have a higher rate of false negatives. This is because there is a chance that a true attack will go undetected due to being classified as normal [49].

Table 4.2 Comparisons between Signature-based and Anomaly-based IDS systems

Detection Mechanism	**False positives rate**	**Detecting attacks**	**OS dependent?**	**Computational Cost**
Signature/knowledge based	low	only known attacks	yes	low
Anomaly/behaviour based	high	known and unknown attacks	no	high

The most common types of IDS systems are:

1. ***Host-based intrusion detection systems (HIDS):*** located in the host machine or can also be placed in VMs. Sometimes these are classified as a family of in-guest monitoring systems. In IaaS clouds, providers may deploy them in physical hosts, while consumers may place them inside the VM itself. Being inside the host or the guest machines provides better visibility of activities inside a VM. The accuracy and computational cost will be based on whether an anomaly or signature-based detection method is used. Because HIDSs are placed directly in the host, they can be susceptible to attack if the host is compromised.

2. ***Hypervisor-based intrusion detection systems:*** located at the hypervisor level and sometimes classified as the family of out-guest monitoring systems, these allow the user to monitor and analyze communications between VMs, between hypervisor and VM, and within the hypervisor-based virtual network [50]. This technique is suitable for the public cloud provider's side as they have control over the hypervisor. Cloud consumers cannot utilize the hypervisor to monitor their hosted VMs because they do not have access to it.The most common technique used by hypervisor-based IDS is Virtual Machine Introspection (VMI), which is based on inspecting the memory and disk of a guest virtual machine *usually* from the outside, as first introduced by [50]. VMI-IDS is different from traditional HIDS since it directly observes hardware states, events, the software states of the host, and offers a more robust view of the system than HIDS [99]. Because VMs execute operations via system calls, many VMI- IDS techniques involve analysing

captured system calls to find anomalies. A hypervisor-based IDS has relatively good visibility on VM behaviour, and since they are placed at the hypervisor level, they can be relatively resistant to attacks. As highlighted in Chapter 3, hypervisor attack is possible although not very easy or common as no direct incidents of hypervisor attack have (generally) been reported.

3. ***Network-based Intrusion Detection Systems(NIDS):*** They monitor network traffic to detect malicious activity such as Denial of Service attacks (DoS) or port scans. Because they are placed outside the host, hypervisor, or the hosted VMs, they have strong resistance to attacks. However, when it comes to finding out what happens in VMs, they have poor visibility, but have no effect on the running of VMs.

4. ***Distributed Intrusion Detection Systems (DIDS):*** consist of several IDSs which communicate with each other or a central server that enables network monitoring. The intrusion detection components collect the system information and convert it into a standardized form to be passed to a central analyzer [99]. This central analyzer can consist of different types of IDSs, so the visibility is based on the IDS type employed. Also, in terms of the effect on VM performance, this depends on the IDS; as using HIDS may influence the performance. This type of IDS can be applied from the provider side or the consumer side.

Table 4.3 presents a comparison between the four common types of IDSs. Intrusion Prevention Systems not only try to detect attacks but attempt to prevent them also; e.g. they may block an attack from a specific IP address. In this case, they are called *active* as opposed to *passive* systems, which do not react to attacks.

Table 4.3 Comparisons between IDS systems which can be used for the cloud

IDS type	**Applicability**	**Visibility of what going in VMs**	**Resistancy to attack?**	**Performance of running VM?**
Host-based	consumer/provider	excellent	weak	may be affected
Hypervisor-based	provider	good	strong	hardly affected
Network-based	provider	poor	strong	not affected
Distributed	consumer/provider	depends on types of IDSs	average	depends on types of IDSs

As can be seen from the comparisons, determining the appropriate intrusion detection methods relies on several factors, such as: applicability, visibility, performance

overhead, and resistance to attacks. Other criteria may include possibility of evasion by the attacker by luring the detection system so alarms will not sound. From the consumer's perspective, only HIDS and DIDS are applicable – assuming that distributed IDSs are all host-based and running directly in the VMs.

4.6.2 VMInformant specialities

VMInformant is a *passive* monitoring system which runs *inside* the VM itself. In that sense it is similar to HIDS. The whole point of VMInformant is to reinforce the trust between the consumer and the cloud provider by checking for malicious events which may involve tampering from the provider side. Therefore, it is not comparable with hypervisor-based IDS or NIDS as both of these require administrative privileges and control from the provider side. The architecture of VMInformant is distributed in the sense that monitored events could be sent to a central owner platform/location for further analysis. However, this distributed nature does not mean that monitored VMs collaborate directly with each other.

A classical Host IDS was not originally designed for VMs. Traditionally, it was deployed directly in physical hosts. IDS could generate massive amount of alerts and produce high false positive rates and false negative rates [90, 161]. The VMInformant checks for the occurrence of specific events which are related to VMs and security. It bases this on a taxonomy of events – Section 4.5. This is also to reduce the effect on the performance of VMs, as performance overhead can be incurred if detection is local.

To arrive at the decision that an event has occurred, VMInformant may incorporate both signature and anomaly checks. Hence, it can be considered hybrid. It is worth mentioning that not all the captured events VMInformant is interested in are related to intrusions – e.g. some of the events are related to the life-cycle of the VM, and capturing these maybe useful to protect the privacy of VMs and checking adherence to SLA terms. A classical Host IDSs do not consider life-cycle events neither do they consider VM-specific events in the first place. In general, VMInformant reports listed, *pre-defined* events. In general, VMInformant advantages over classic HIDS can be summarised in the following points:

- The ability to provide a more focused security monitoring which spans multiple event groups. This will allow the user to control recording of events according to the need. Thus, the design is made with VMs in mind and the monitoring is performed from the cloud consumer's perspective.

- The inclusion of life-cycle events, which are not necessarily deemed as *intrusions* but they could affect the privacy of data and the adherence to SLA terms.

4.7 The Architecture of VMInformant

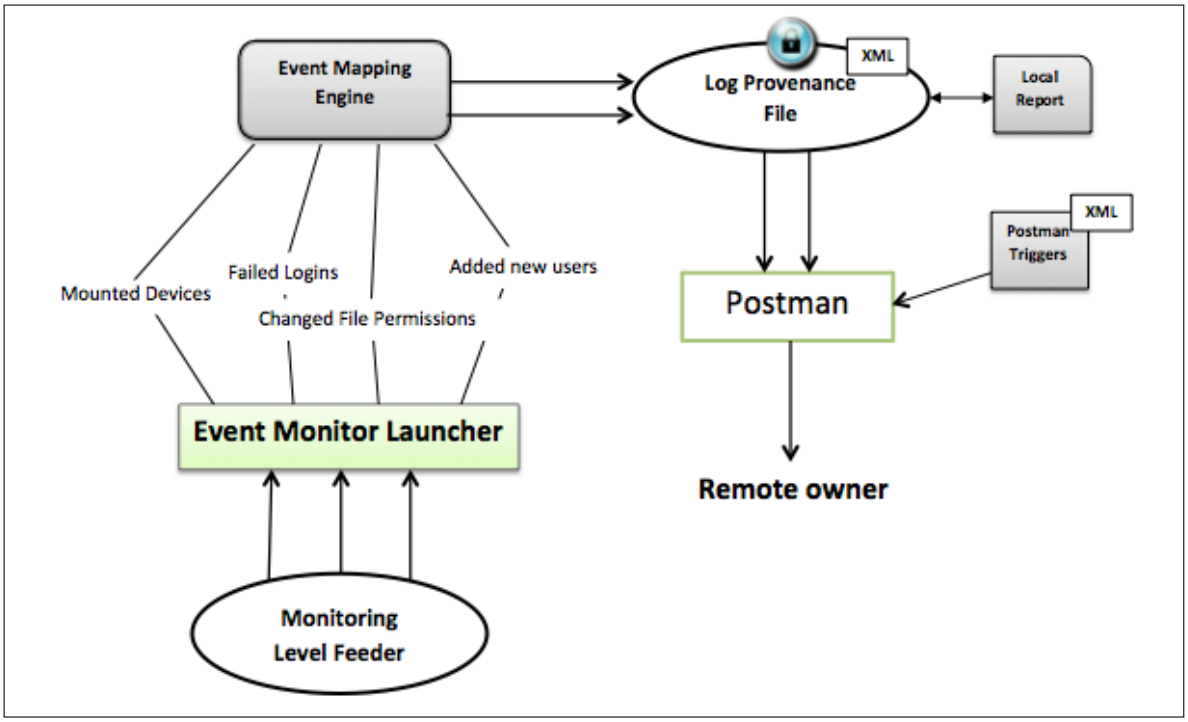

Fig. 4.2 Architecture of the proposed system

Figure 4.2 shows the architecture of VMInformant. It consists of the following main components: *Monitoring Level Feeder, Event Monitor Launcher, Events Mapping Engine, Log Provenance File* and *Postman.* In the following sections a description of each component and its role in the architecture of *VMInformant* will be highlighted.

4.7.1 Monitoring Level Feeder

The Monitoring Level Feeder component acts as an interface for the owner/consumer of the VM. It enables the consumer to determine what security-related events need to be monitored and to change the granularity of event recording (i.e. how many events should be recorded based on available storage and potential impact of event recording on the overall performance of the application running in the VM). Section 4.5 discussed a taxonomy of security-related events in further detail. The interface could

be either: graphical or command line. The graphical interface may contain check boxes identifying which events need to be monitored. For instance, for file-level updates, the consumer is required to identify the file/folder to be monitored along with the event types to be monitored for this particular file/folder (open, delete, update, etc.). In the case of monitoring process-related events, for example, the consumer can decide the granularity of the recording of process data.

4.7.2 Event Monitor Launcher

This component launches event monitors based on the configurations acquired from the Monitoring Level Feeder so that they run as a daemon process to capture events as they happen and provide data for other components. The system has to take into consideration scalability in terms of the number of monitors which can run concurrently. For example, the consumer may decide to monitor many different folders for changes or to start periodically monitoring new users. This would require launching more than one monitor. Also, there could be different monitors for different categories of events. Another example is monitoring the log file `/var/log/syslog` in Linux for USB devices attached to the VM.

4.7.3 Events Mapping Engine

This component attaches semantic labels/ descriptions with observed events in order to store them in a provenance log. It is an optional component, although it might be necessary in order to reduce the quantity of logged data. An example of the use of the Events Mapping Engine is after observing successive failed logins to the system; the mapping engine maps or translates this into a "possible breach of the system by brute forcing passwords". In short, this component cleans the data so that only useful information will be stored in the log, rather than storing every possible event which could be monitored. There must also be an ability for the consumer to examine details related to the event– i.e. the raw event log should still be available if further investigation is needed. An alternative and preferred approach to mapping and analysing the recorded event inside the VMs is to send the logs periodically to a central analyser.

4.7.4 Provenance Log

The provenance log will contain information obtained from the Events Mapping Engine. An XML-based schema is used to describe the data contained in this file, enabling other externally developed components to also make use of this data. In this way, the provenance log can be used to produce a VM-specific report, so that the owner/consumer can explore any critical events. This file is encrypted and only readable by the owner of the VM. This prevents tampering or updates to this file by the cloud provider.

4.7.5 Postman

The Postman component fetches the provenance log and submits it to the VM owner. A configuration file identifying possible trigger rules is used to determine when this submission should be made. Examples of possible triggers include:

1. The size of the provenance log file exceeding a threshold previously identified by the VM owner.
2. The occurrence of a possible breach –i.e. the existence of an event trigger.
3. Periodic delivery of the file to a consumer at a particular time, e.g. every day at 3 a.m.

It is worth mentioning that there are tools which allow the sending of log entries in near-real time. This is useful for detecting malicious events as soon as they occur. Such tools usually require an agent to run as a daemon process that monitors updates on the generated provenance log file. Examples of potential tools are *splunk*[3] and *collectd*[4].

The VMInformant system architecture described above assumes live monitoring of events. However, critical security-related events that occurred before launching the monitors may be present. In this case, the VMInformant requires an optional capability which allows the scanning of event logs for security-related events whenever this is needed, and to display them in an informative way. A number of possible visualisations of such events can be provided based on the level of expertise of the consumer.

Figure 4.3 shows the interactions between the different components in the system. From the figure, the owner/consumer of the VM configures how they want the

[3] http://www.splunk.com
[4] https://collectd.org/

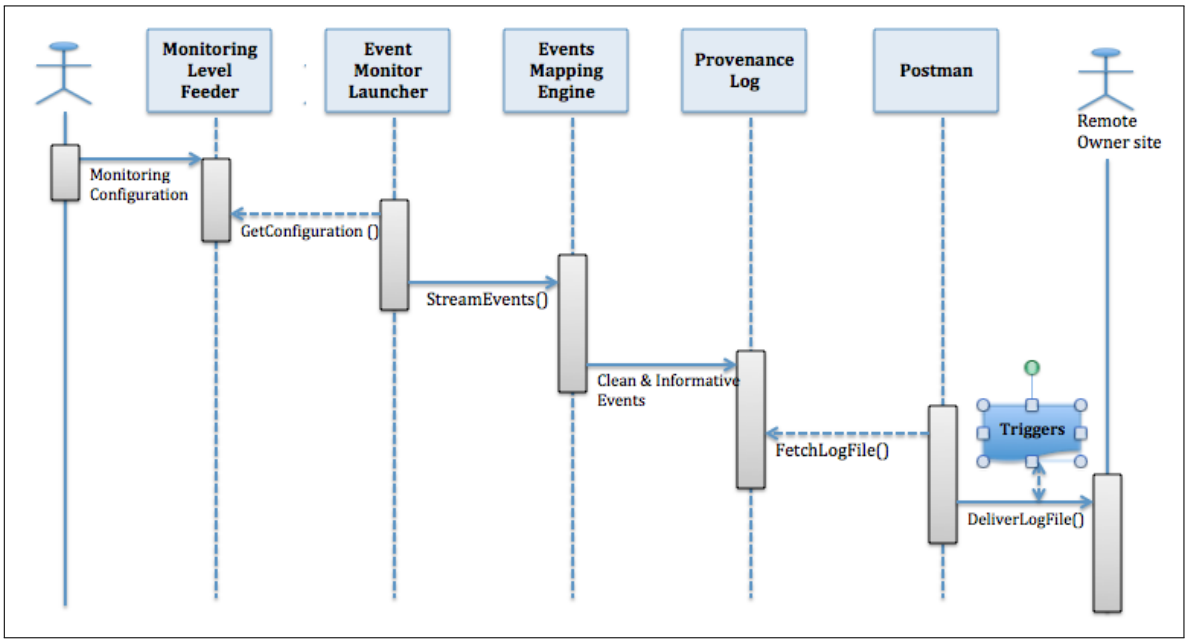

Fig. 4.3 Sequence Diagram showing interaction between components

monitoring to be performed in the Monitoring Level Feeder, while the Event Monitor Launcher fetches the configurations and launches the monitors. It then forwards the recorded events to the Event Mapping Engine, where further processing is performed to produced clean and informative data. It then stores the data in the provenance log in XML format. The Postman component fetches the provenance log file and delivers it to the consumer based on the trigger rules. The consumer is expected to have this log from various VMs.

4.8 Reference Implementation of VMInformant

The architecture of VMInformant was partially implemented and evaluated. Not all features of the proposed system are implemented in our current prototype. Some features were added later as individual scripts which perform specific monitoring tasks. Our first priority was to generate a working data set of security-related VM events from the VM (in a log file). The next stage was to send these events to a central location (consumer side) for processing. The following sections describe the implementation stages in detail.

4.8.1 VMInformant's interface design

We focus on the first two components of the VMInformant architecture from Sections 4.7.1 and 4.7.2. Our current prototype has been implemented in Python using the PYQT[5] library, which is a Python binding of the cross-platform GUI toolkit Qt. A GUI was implemented to:

1. Make it easier for non-experienced users to start monitoring tasks; as the tool will take care of complex configurations.
2. To reduce the chance of making errors or missing important monitoring tasks.
3. To facilitate the dynamic monitoring of locations in case monitoring plans change; as the tool will allow easy addition of monitors as and when needed.

In the implementation of the prototype we focused on two security categories highlighted previously in the taxonomy: *file-centric* and *user-access* events. File-centric events are expected to generate more data because actions on files or folders may trigger multiple events. The tool allows:

- Choosing a file or a folder to be monitored for changes (modification, deletion, movement). If it is a folder then it monitors the creation of new files or folders in it recursively.
- The monitoring of permission changes for files or folders.
- The monitoring of new users added or removed from the system.
- The monitoring of failed attempts to log in to a VM.
- Turning off/on the monitoring options.
- Launching of event monitors, in addition to the ability to start or pause them at any time.
- Choosing where to store the provenance log file.

Our implementation serves as the basis for achieving most of the monitoring activities of security-related event categories described in the taxonomy in Section 4.5. Figure 4.4 shows a snapshot of the tool.

[5]PyQT https://wiki.python.org/moin/PyQt

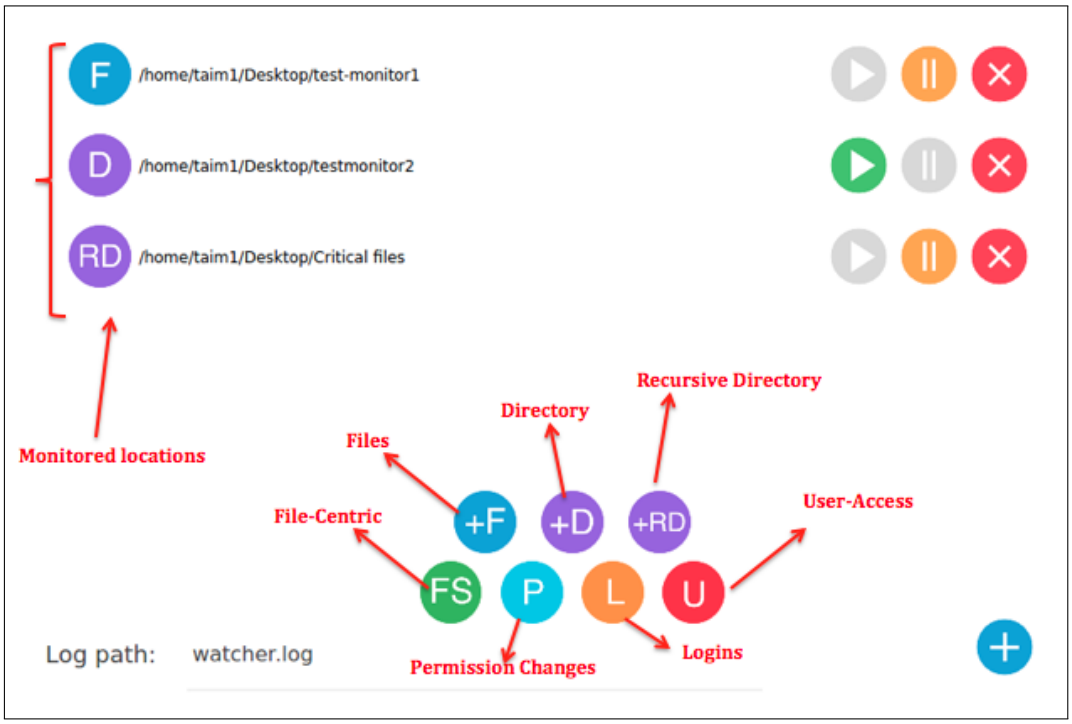

Fig. 4.4 VMInformant user interface

The GUI of the current prototype contains seven main icon options for the consumer to select. Each icon has been labelled as follows:

- **"+F"** : This will open a file system dialog window to allow the consumer to choose the file to be monitored for changes.
- **"+D"** : This will open a file system dialog window to allow the consumer to choose the directory/folder to be monitored for changes. The monitoring will be applied to all the files in the that folder only.
- **"+RD"** : This will open a file system dialog window to allow the consumer to choose the directory/folder to be monitored for changes. The monitoring will be applied to all the files in that directory and in the directories inside it recursively.
- **"FS"** : This will apply monitoring file system changes to the selected files/directories.
- **"P"** : This will allow the consumer to check when the permission bits of the selected files/directories changed. When enabled, the provenance file will contain permission changes events.

- **"L"** : When this is selected, successful and failed login attempts are logged.
- **"U"** : When this is selected, new user accounts that are added to or removed from the system will be logged.

Other features that allow the monitoring of other events from the taxonomy in 4.5.2 can also be added to the GUI. Whenever a file/directory location is selected for monitoring, a thread will be created and the full path will appear in the interface window with an icon to indicate the level of monitoring. In addition, three icons will appear to control the monitoring: start, pause, and remove. This is to enable flexible monitoring according to the demands of the consumer. The provenance log file is stored by default in a specific location, but the GUI will enable the consumer to choose the save location of the file. This is useful in cases where an application needs to make use of the log, and storing it in a specific location is a requirement; e.g. to be shipped to a monitoring or visualisation tool like Stackdriver[6].

4.8.2 Working environment

The reference implementation of VMInformant was designed to work on virtual machines hosting the Linux Ubuntu OS – as a proof of concept. Since it was implemented in Python, the interface will be displayed in various operating systems. However, because some of the operations require making use of tools which are Linux-based, VMInformant currently support Linux-based OSs. The subsequent sections are based on a testing environment where VMInformant runs in VMs hosted on an Ubuntu host. We use KVM as a hypervisor installed on a Ubuntu host. KVM[7] is an open source hypervisor developed as an extension to the Linux kernel – and made use of in a number of different cloud systems such as OpenNebula. The VMs run Ubuntu 12.04 as a guest OS.

4.8.3 Skills of the user

VMInformant is a monitoring tool which operates from the cloud consumer's perspective. Since an easy-to-use interface design was used to implement the prototype of VMInformant, it is assumed that no particular technical skills are required in the users working on the tool. Any user with basic understanding of monitoring needs could easily use the tool. However, deciding the priority of monitoring tasks and designing

[6]stackdriver: http://www.stackdriver.com/
[7]KVM: http://www.linux-kvm.org/page/Main_Page

a monitoring strategy require people who understand security and the implications of the choices around it. For example, as will be highlighted in Chapter 5, choosing parameters for the migration metrics require an experienced party which understands security. As well as understanding the security issues, users need to understand the effect of monitoring options on the performance of the VMs.

4.8.4 VMInformant operations: how does it work?

The VMInformant tool automatically identifies the required missing dependencies and installs them in the VM in order to start the monitoring process. It makes use of the *Watchdog* library. This is a python library that allows the monitoring file system events. The library enables the launching of observers that monitor file system events. It also enables handlers for these events; e.g. to start an action if certain events were observed. VMInformant also makes use of *auditd* – which is a subsystem for monitoring and accounting of the Linux OS developed and maintained by RedHat. It produces log entries which are stored in `/var/log/audit/audit.log`. Rules to govern the monitoring in auditd are stored in `/etc/audit/audit.rules`. Adding auditd rules can be done using the auditctl[8] tool, which controls the behaviour, get status, and adds or deletes rules in the auditd sub-system. VMInformant dynamically adds monitoring rules to *auditd* by passing shell commands using the *sub-process* python module. Auditd was mainly used to check for new users which have been added to the system and for associating events with user identities. To facilitate the use of the logs generated by auditd, the *ausearch* tool was used in VMInformant as a sub-process to search for added and removed users. Ausearch[9] is a tool that can query the audit daemon logs for events based on different search criteria. VMInformant also checks: `/var/log/auth.log` for information about failed logins by searching for "authentication failure" messages. Most of the logs contain user and user group data in the form of IDs which are not translated into meaningful information. VMInformant maps the IDs to the corresponding meaningful user names and user groups. It also formats the time and date and makes them more readable.

The tool currently gathers all the collected events into one log file. If the generated event is file-centric, we record the following:

- (Timestamp) (Type of Event) (path to file)
  by (user-name) from (user-group)

[8] auditctl: https://linux.die.net/man/8/auditctl
[9] ausearch: https://linux.die.net/man/8/ausearch

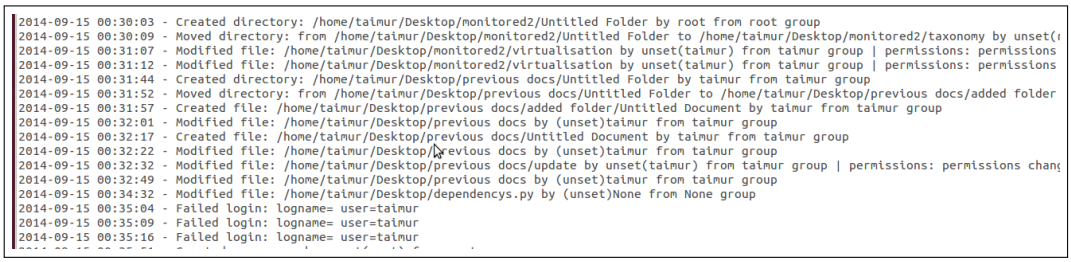

Fig. 4.5 A snapshot of the log file containing some of the monitored events

- (Timestamp) (Type of Event) (path to directory/folder) by (user-name) from (user-group)
- Permissions: changed permission from (old permission set) to (new permission set) by (user-name) from (user-group)

The type of the event can be: modified, deleted, moved, opened, etc. If a user-access event is observed, we record according to:

- (Timestamp) login failed, user=(user-name)
- (Timestamp) created user=(new user-name) by (user-name) from (user-group)
- (Timestamp) removed user=(user-name) by (user-name) from (user-group)

VMInformant was tested initially by adapting a scenario where a sequence of events are triggered. A snapshot of the resulting log file can be seen in Figure 4.5. It is worth mentioning that although the prototype of VMInformant was run on a VM hosted in a physical machine, the same can be applied to a VM which is hosted on a public IaaS. In Chapter 6 we cover this in more detail, as well as covering how events from multiple VMs can be aggregated and analysed. Also, given that we had to check operations on files, there exist tools which check file level updates, but we preferred to implement a dedicated tool using the available open source libraries to control how the provenance file is formatted and the level of detail.

4.9 The Attacker Model

The VMInformant architecture allows the consumer to monitor and record specific security-related events inside the VMs. The VMs are hosted in a public IaaS cloud system, which means that the provider exerts full control over the infrastructure. Attackers may try to compromise the monitoring application or destroy/manipulate the recorded events to cover their trails. Table 4.4 shows the various potential attacks and

whether our system provides protection against these attacks. It is important to note that the main assumption in this research is that the provider cannot be fully trusted. Thus, the attacker model is based on the provider trying to evade detection/recording of the events and informing the consumer. Also, it may be possible that the attacker is one of the cloud consumer's employees. In such cases, the consumer/owner needs to regulate who can access their production systems hosted in VMs. Also, to control who can configure the monitoring applications.

Table 4.4 The potential attacker model

Type of attack	**Protected against?**	**Method/suggestions**
Disabling the monitoring application	Not directly	Because events are to be send to a central analyser VM (Ch 6), if no events are received, the consumer will know. Hence, investigating the matter would be possible.
Viewing the provenance log file	Yes	The provenance file is encrypted. Besides, if the events are to be sent immediately (or after a specific time interval), prior event data can be discarded.
Manipulating the provenance log file	Yes, partially	The provenance file is read-only (only the monitoring application can modify it).
Deleting the provenance log file	Yes	The provenance file is read-only and hidden in the VM (cannot be found easily)
Flooding the system with events	Not directly	Luring the system into triggering events unnecessarily may be controlled by including rules, e.g. a rule to check for identical events fired within a short window of time.
Blocking periodic ICMP PING requests from the VM	Yes	Choosing a suitable ping interval, e.g. five seconds.
Crashing the monitoring application.	Not directly	If the monitoring application is crashed, restarting the VM will restart the application. Currently, there is no support for reloading configuration parameters.
Changing monitoring configurations parameters/options	Yes, partially	Currently, parameters are set using the GUI of the VMInformant prototype. The parameters are not stored in a particular file. However, the architecture of the Inspector station allows configuring the monitoring in the VMs by remotely changing the parameters (stored in a file). If the parameters are then stored in a file, it must be protected against tampering.

4.10 Evaluation

A key objective of this work is to investigate the trade-off between monitoring of security related events and the likely overhead such monitoring has on application performance. In VMInformant, a consumer is able to selectively choose how many events will be recorded (and with what frequency), to ensure that the size of the event log does not create a significant overhead within a VM. Therefore, as the process of monitoring events and writing to the log file, as well as sending the log file data back to a consumer/VM owner, may incur a cost in terms of the performance and size of the

log file, we evaluated the performance and efficiency of our technique. One objective of undertaking such an investigation is to better understand how many security related events are likely to be useful (in terms of the C-I-A triad) whilst reducing the potential performance penalty. Our discussion on the benefit of our technique spans three axes:

1. Usefulness: This is related to how important the monitored events are to a VM owner.

2. Performance: Given the types and number of monitored events, we would like to better understand how the recording of these events affect the performance of the VM. It is important to note that all VMs have limited processing capabilities.

3. Data size: a VM has a limited virtual hard disk. As the provenance log can grow over time, it is important to understand how this will affect other operations of the VM.

Hence, from the perspective of overhead analysis, we aim to determine the optimum combination of: (i) recording of useful events; (ii) keeping performance overhead minimal; (iii) reducing the virtual disk space required to store the provenance log. In this section we describe how we evaluated the performance of the proposed system (VMInformant prototype) using widely used benchmarks. When running inside the VM, normal running applications may use CPU or I/O. Running VMInformant, which is a monitoring application, alongside the standard application presents an additional overhead.

Consumers use virtual servers in public IaaS clouds to host their production applications. These applications are expected to function efficiently and smoothly without being affected by monitoring applications. Therefore, in order to evaluate VMInformant, we need to see if it affects the normal operation of the virtual machine. To perform this task we make use of benchmark applications which are both CPU and I/O intensive in order to see how many events are collected and what is the overhead of creating a log file for these events within a VM. In this evaluation we focus on file-centric events, i.e events such as created, deleted, moved , modified, modified file/folder permission bits. This can help address questions relating to the confidentiality and integrity of data within the VM. We used Povray[10] and Bzip[11] as benchmarks. Povray is an open-source ray-tracing application, which can vary in computation time based

[10]Povray http://www.povray.org/
[11]Bzip2 : http://www.bzip.org/

on the complexity of the scene being rendered. Povray generates a number of output files representing different frames of interest. It is widely used to create animated graphic scenes using pre-developed scripts. *Bzip2* is an open-source file compression utility which can be used to compress a number of different file formats. It can take large files as input.

We chose both Povray and Bzip2 as benchmarks because:

- They are both CPU/IO intensive: this means that running them to act as production applications of the consumers will mark the worst case scenario.
- Both of them take input files with variable size/complexity: which allows us to try different scenarios with variable file size and complexity.
- Running these tools in the VMs while monitoring is likely to trigger many file-centric events. This will enable us to check if the performance is affected by events written to disk.

4.10.1 Methodology

To carry out all the experiments we used an Ubuntu VM hosted on a KVM host. The VM has 1 GB RAM, 10 GB of disk storage and makes use of one virtual CPU.

Our evaluation methodology can be summarized as follows:

1. We run the benchmark application in the VM using a specific input file while no monitoring is taking place.
2. We record the total execution time of the benchmark. This includes the time to carry out the computations and the time to produce the output.
3. We then turn the monitoring on for selected folders where the benchmark will perform its operations.
4. Then we run the benchmark application using the same input files and record the start time of the process.
5. When VMInformant finishes writing the events resulting from running the benchmark in a provenance log, we record the size of the log and the time of the last entry in the log. We use this to calculate the total execution time while monitoring is on. This total execution time will include the total running time of the benchmark to produce the output plus the time taken to detect the events and log them to the disk.

6. To calculate the time overhead we deduct the total execution time of the benchmark without monitoring from the total execution time whilst monitoring.

Based on the Povray and Bzip benchmarks, we set up a number of different scenarios to validate the outcome across different input sizes (and complexities):

- **Running povray to render both a simple and a complex input files:** We used the *glasschess.pov* file as a simple input and the *benchmark.pov* file as complex input. The latter file is recommended by the Povray community to be used as a benchmark to measure the performance of systems. In our experiment we did not use it according to the recommended settings because we were more interested in the events that could be triggered during the execution of the benchmark.
- **Generating a high resolution output (800*600) from the simple and the complex input files:** in this scenario, we also used the *glasschess.pov* and *benchmark.pov* input files, but we configured Povray to render a high resolution version of the images. We were interested to see how many file-centric events could be generated and collected in order to better understand the duration of time taken to write these events to disk.
- **Generating a different number of frames (animated pictures) from the same input file.** We used the *glsbng.pov* input file to create 25 and 100 frames respectively. In this scenario we aimed at triggering both I/O and CPU related events to see how the recording of events would be affected. Generating more frames can trigger many instances of file-centric events within short periods of time, which will challenge the process of writing the provenance log to disk.
- **Using Bzip to compress two different files of different sizes:** in this scenario we used two input files: *Google Chrome file*, and *Ubuntu 13.4*. We wanted to see how the file compression process can be affected by the recording of events while the compression process was executing.

4.10.2 Results

Table 4.5 shows the result of conducting experiments using the benchmarks based on the scenarios discussed in Section 5.10.3. From the table, using the Povray input file 'glsbng.pov' as a benchmark to render a number of frames (25,100,500) caused a reasonable overhead. The overhead is reduced as the number of frames increases

Table 4.5 Results of running the benchmarks with and without VMInformant

Benchmark Name	execution time (monitoring off)	execution time (monitoring on)	Time overhead	Overhead (%)	Events	Storage Overhead
Pov-ray (glsbng.pov) 25 frames	11 sec	12 sec	1 sec	9.09%	75	9.8 KB
Pov-ray (glsbng.pov) 100 frames	47 sec	50 sec	3 sec	6.38%	300	37.7 KB
Pov-ray (glsbng.pov) 500 frames	230 sec	242 sec	12 sec	5.22%	1500	188.5 KB
Pov-ray (benchmark.pov) standard	148 sec	190 sec	42 sec	28.38%	18	3.0 KB
Pov-ray (glasschess.pov) standard	12 sec	13 sec	1 sec	8.33%	5	0.89 KB
Pov-ray (benchmark.pov) 800*600 px	828 sec	936 sec	108 sec	13.04%	71	11.6 KB
Pov-ray (glasschess.pov) 800*600 px	57 sec	69 sec	12 sec	21.05%	23	4.0 KB
Bzip (chrome.deb) size : 47.7 MB	6.6 sec	8.5 sec	1.9 sec	28.79%	9	1.3 KB
Bzip (ubuntu.iso) size 794.8	109 sec	141 sec	32	29.36%	79	11.1 KB

even though the number of written events to disk also may increase. This is due to events being generated over a long time-span and therefore reduction in the overhead of starting up VMInformant. For the same reason, using Povray to render a high resolution (800*600px) image from the input file (benchmark.pov) produces a relatively small overhead of 13%, compared to an overhead of 28.38% when rendering the standard benchmark.pov input file. It can also be observed that the number of events generated by executing the benchmark on a given input file is proportional to the number of events generated by the benchmark on the same input but with higher resolution output. This indicates that the benchmark follows a similar pattern, which can help us approximate the number of events when considering a different processing setup.

Running VMInformant after executing the file compression tool (Bzip) on variable file sizes yielded similar overheads of 28.79% and 29.36% for small and large file sizes. This is a considerable overhead which arises due the file compression operation that incurs both CPU and I/O costs and the monitoring of the file system where compression is taking place. This can lead to many events being dispatched while the CPU is still busy compressing the file. We can argue that the more time spent in computation, the more chance that VMInformant will be able to write the events to disk in the background. This will also delay the triggering of more events within a short time-span. The storage overhead was minimal in the experiments because the number of fired file-centric events were relatively small. Regardless of the number of events, based on our proposed VMInformant architecture, we can periodically migrate the provenance log to the VM owner. Figures 4.6, 4.7, 4.8, 4.9 also show a comparison between the execution of the different benchmarks.

From the experiments and the discussion above, we argue that it is hard to identify a specific performance hit associated with monitoring events. As mentioned before, the choice of the benchmarks was basically to mark the worst case scenario; when the applications running in the VMs are both I/O and CPU intensive. In practice, this is not usually the case. This means that the overhead observed in the experiments is unlikely to occur. Since the monitoring strategy of VMInformant follows a focused approach, where specific types of events are trapped and the user can determine the granularity of monitoring, VMInformant is unlikely going to have a notable impact on the running applications of the VM. In general, when it comes to deploying VMInformant in a real public IaaS environment, the consumer could decide an acceptable overhead incurred by monitoring (or a threshold). If the monitoring hits this overhead threshold, then the monitoring strategy can be revised. In some of the scenarios of the experiment, an overhead of more than 25% was observed. This is considered very high and has to be reduced. We argue that some of the overhead may be due to the way our GUI interface launches monitors. Using a classic command-line application or improving the code may improve the results. However, this is a secondary priority, as further enhancement can be made to the prototype implementation of VMInformant.

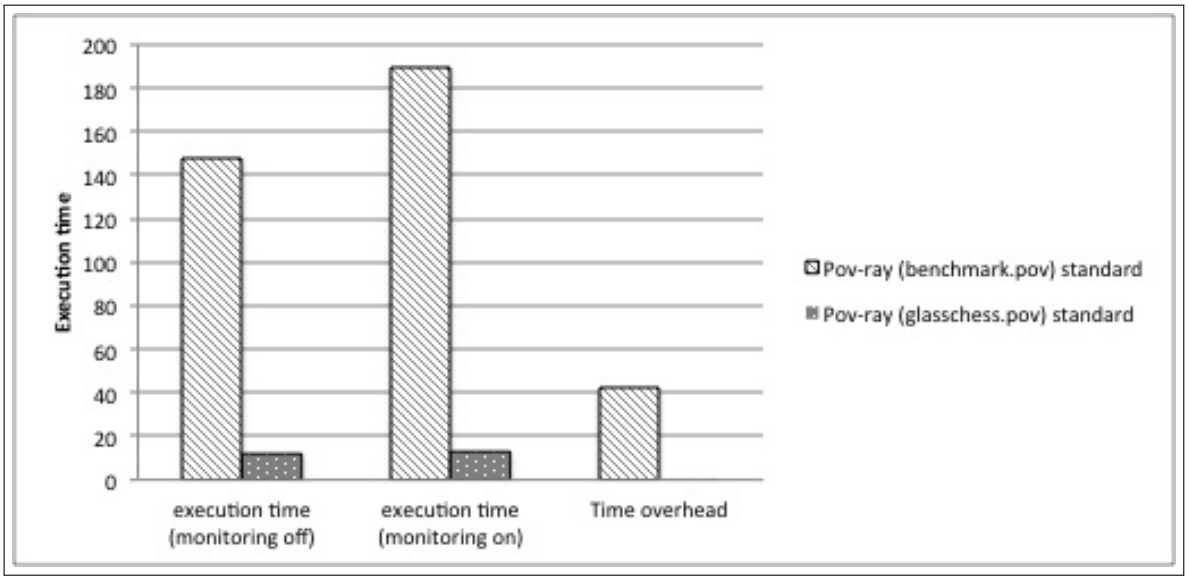

Fig. 4.6 Execution of the Povray benchmark using: benchmark.pov and glasschess.pov

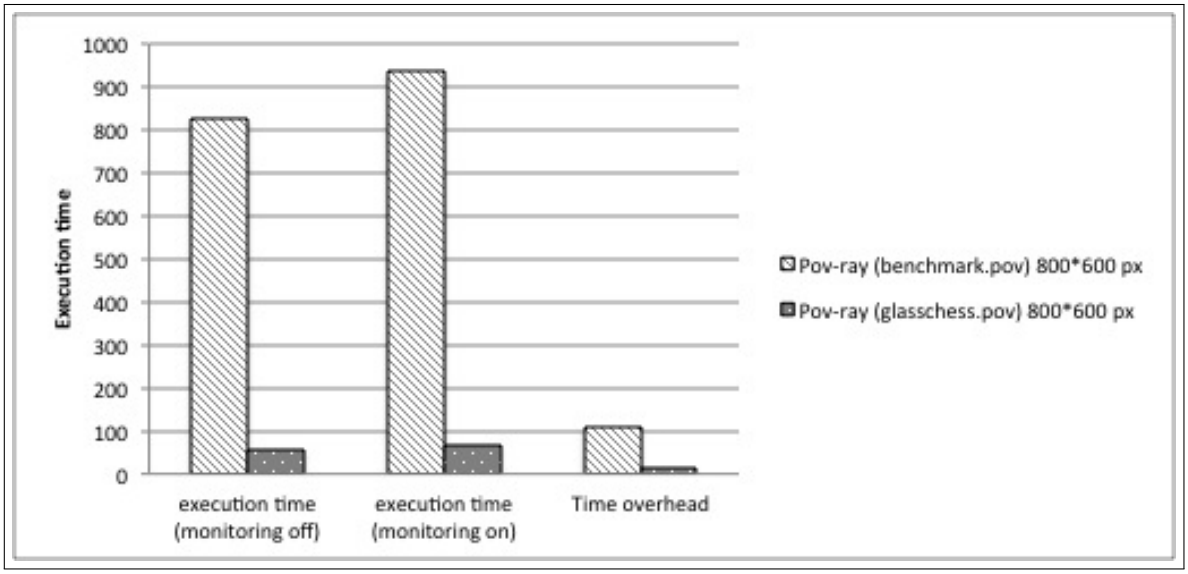

Fig. 4.7 Execution of the Povray benchmark using: benchmark.pov and glasschess.pov using high resolution image output

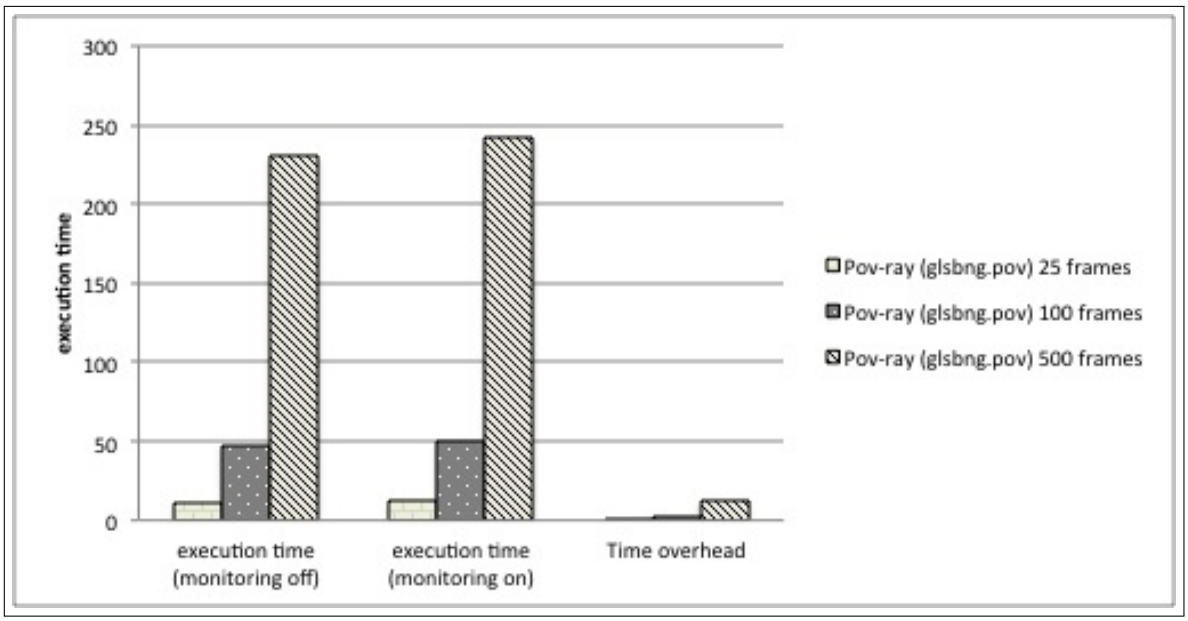

Fig. 4.8 Execution of the Povray benchmark using a larger number of frames with input file glsbng.pov

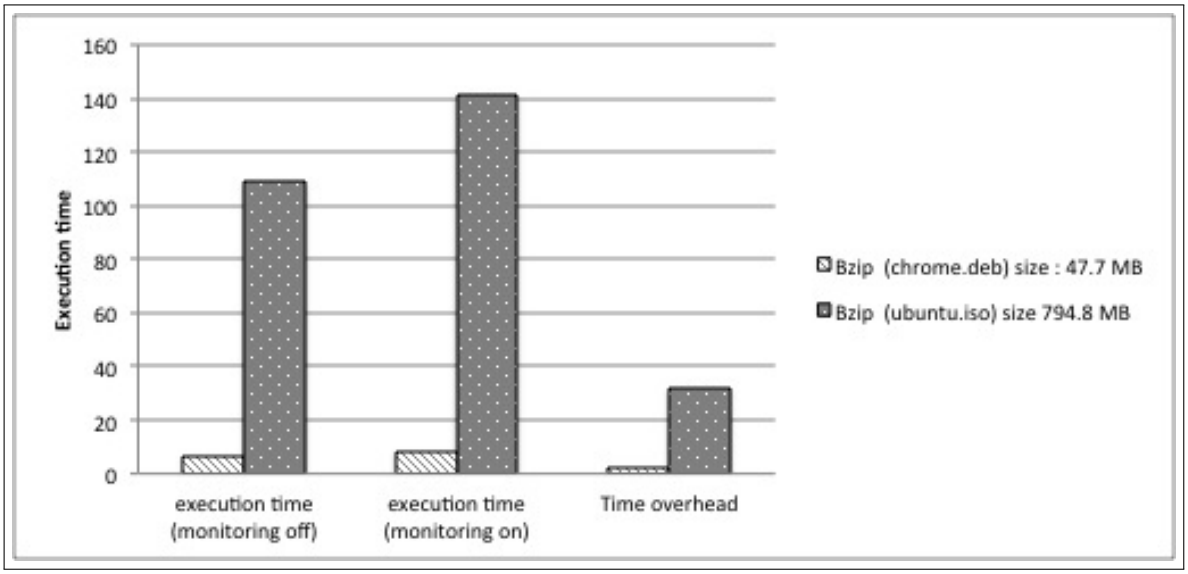

Fig. 4.9 Execution of the Bzip tool on two files (simple and complex)

4.11 Limitations and suggestions

We believe that the VMInformant architecture may have some limitations which need to be addressed:

- **Possibility of compromise**: VMInformant or the recorded provenance log itself could be compromised. As a partial solution we encrypt the log file so it cannot be tampered with. Compromise of in-guest monitoring systems is a common threat. Having logs sent to a central analyser may help detecting such compromise, as data will stop flowing into the analyser. When that happens, it may be possible to decide on further actions, e.g. stopping or restarting the VM. One solution to ensure the flow of event data from the VM to the central analyser is to send a dummy event periodically. Not receiving this dummy event for a long period of time could indicate an incident which need to be checked. The central analyser concept was implemented and discussed in Chapter 6. Another option to ensure that the monitoring tasks are performed as expected by VMInformant (i.e. not compromised), is to manually trigger events in the VM and observe if they are detected successfully.

- **Possibility of Cloning**: As mentioned before, the cloud provider has full control over the IaaS public cloud. Thus, employees of the cloud provider could just clone the VM (memory and disk), thereby gaining full access to the consumer's

data. If an intruder compromises the cloud providers servers, he could also achieve the same goal. In this chapter, we argued that cloning of VMs is one of the events which should be monitored and detected from the consumer side when it happens. Implementing techniques to detect the cloning event from the consumer's perspective is set to be one of the future tasks.

- **Processing text logs**: The type of malicious event will be mapped to a certain format before being written to the provenance log file. Eventually, as more events occur, the processing of large text data may incur a considerable overhead. A solution would be to store only indexes to events. Later, at the consumer end, events can be reconstructed by mapping indexes to a table that contains a list of all the events in the taxonomy. This was partially implemented in the architecture of the inspector station in Chapter 6.

- **Evaluation**: We evaluated VMInformant only on file-centric events. Also, it was evaluated based on the assumption that the observed file has not yet been sent to the owner site.

- **I/O overhead:** Writing to disk files (especially virtual disks) can be slow. If something triggers many events at the same time, this would cause the module that aggregates events in the provenance log to halt. A possible solution is to write the events to memory instead of disk. This will allow faster processing of events. Also, instead of storing the logs inside the virtual machine itself, dedicated public cloud storage can be used, e.g. Amazon Simple Storage Service (S3).

- **Program crash:** During the experiments, there were some occasions when the monitoring tool crashed. This was mainly due being overloaded with events waiting to be written to disk. Since VMInformant was just a prototype implementation, we believe an improved code and exception handling would fix the problem of crashing. Currently, if events are displayed in the standard output without writing them to disk, the crashing problem does not occur.

4.12 Scope of the implemented detection mechanisms for the events in this work

Table 4.6 shows the list of the event groups that we were able to detect and include in our work. Also, it shows what was the method of detection (in brief). Section 4.8.4, Chapter 5 and Chapter 6 include more details on how the events were trapped. As can be seen from the table, the migration event is the only event among the *life-cycle* group of events which we were able to devise mechanisms to detect. Other events such as: cloning VMs, snapshotting VMs, etc, are yet to be explored in future work.

Table 4.6 Listing the implemented features related to the taxonomy of events and how detection was done

Event groups	**Detected in our work?**	**Method of detection**
File-centric	Yes, in the prototype implementation of VMInformant	Using python watchdog, auditd. Check Section 4.8.4
Life-cycle	Only the migration event in Chapters 5 and 6	Using various techniques such as: monitoring ICMP Latencies, detecting hardware change, etc. Please check Chapter 5
Process-related	Partially to trap some specific system calls such as: execve (an executed process)	Using tools such as: auditctl, ausearch, snoopy
User-access	Yes, in the prototype implementation of VMInformant.	Using ausearch tool and by checking logs such as: auth.log
Network-related	Yes, partially to check closed/open ports	Using some independent tools such as: nmap
Attached devices	Yes	By searching syslog.log

4.13 Performance Implications of Security

From the experiments, it appears that there is no specific number of events to be monitored in the VM in order to cause minimal effect on the performance of the running routine applications. Thus, in order for the consumer to make the decision as to how to secure the hosted VMs while not affecting the performance of the running application, we believe they should base this on multiple factors, which include but are not limited to:

- How critical is the hosted VM, i.e. Does it really need to be monitored?
- What are the events which can be considered more severe or have greater security impact (from the consumer's perspective).

- What to monitor in the VMs (e.g. full folders or specific files).
- Frequency of monitoring (e.g. periodic or continuous).
- Storage mechanism (e.g. local or in a remote server).

This is strongly related to the monitoring strategy highlighted in Section 4.4. Deciding the granularity and the level of monitoring can have a great impact on the performance. The interface design of the prototype implementation of VMInformant allows determining the level of detail recording of events is needed. For example, it allows activating/deactivating the monitoring of file-centric events or user-access events. Choosing the events to be monitored according to the severity has to be performed by someone who understands security and its implications (from the consumer's side).

In the evaluation of the monitoring approach, we were interested in the the performance of the applications which run in VMs monitored by VMInformant. Thus, VMInformant performance evaluation was not carried out. This is partly because the functions performed by VMInformant requires events to be triggered by users or other applications. In general, if we are storing the events in a local provenance file in the VM itself, VMInformant performance will be a function of the number of detected events waiting to be written to disk. Thus, if the there is a large number of events to be written to disk, VMInformant, will spend more time running in the background. If the event data are to be displayed in the standard output, without having to be written to disk, VMInformant can be more effective. As will come in Chapter 5, VMInformant also receives input from other modules which perform calculations to determine migration events. The efficiency of these modules may affect the performance of VMInformant.

4.14 Discussion: Detecting Security-related Events in Containers

Containers make use of OS-level virtualisation to run isolated Linux-based systems which share a single Linux kernel. All relevant applications and their libraries are bundled together in one container. This segregation leads to faster application delivery, as developers will be concerned about applications running in the container, and system administrators will be concerned about deploying the containers in the server [77].

Unlike virtual machines which can run different operating systems regardless of the operating system of the host, containers have to run the same operating system as the

host. Using containers enhances the utilization of servers because they share a single kernel. This implies the possibility of running far more containers on a single physical host than is possible with VMs. This also means fewer physical servers are needed to run hundreds of application in containers. In addition, no booting is required with containers, which means launching them can take a matter of seconds. There are some existing containers implementations such as Docker[12], Lxc[13], lmctfy[14] and OpenVZ[15]. Docker has made it easier to create containers, particularly as it uses the concept of a hub, which enables users to build containers by downloading code from a portal and developers to contribute code and store it in the portal[16]

The main isolation factors in containers are:

- **cgroups:** a Linux kernel feature which provides for grouping of tasks and resource tracking and limitations for those groups [80]. Resource usage including: CPU, memory, disk I/O, network, etc. of a collection of processes.
- **namespaces:** a feature of the Linux kernel that isolates and virtualizes system resources used by a collection of processes. Examples of resources that can be virtualized include process IDs, hostnames, user IDs, network access, inter-process communication, and filesystems [80].

Because of this isolation method, unlike VMs, containers do not require hypervisors for isolation. While this can provide better utilization of resources [77], it can also lower the security of containers. For virtual machines, hypervisors provide a natural level of isolation. As mentioned before, compromising the hypervisor could mean compromising all the hosted VMs. In containers, gaining root access to the OS kernel could allow intruders direct access to all running containers [62]. Thus, containers may provide better utilization of resources at the expense of security. Since the main research in this thesis is concerned with detecting malicious events in VMs, it makes sense to ask some questions as to whether it is practical or possible to monitor malicious events within containers using the same techniques that we followed. In this section, we discuss this.

In general, container deployment in IaaS public clouds can be carried out in two ways:

[12]Docker www.docker.com
[13]Linux Containers https://linuxcontainers.org/
[14]lmctfy (Let Me Contain That For You) : https://github.com/google/lmctfy
[15]OpenVZ https://openvz.org/Main_Page
[16]Docker Hub https://hub.docker.com/

- *Directly running containers on top of an operating system which is hosted in a physical machine*: As mentioned before, the separation mechanisms in this depend mostly on *cgroups* and *namespaces*. Containers are transparent to the operating system. This implies that operating systems have a direct view of what the container is doing. Getting access to the OS means getting access to the containers under it. However, theoretically, containers can see only their own environment, and do not affect or have access to other containers because namespaces provide restricted access to file systems like chroot with a directory structure [77]. In other words, containers cannot access what they cannot see [44]. Some cloud providers support this, e.g. Amazon Web Services (AWS) has support for containers with their ECS[17] (Elastic Container Service), which allows the creation of a cluster of container instances (pre-configured instances with Docker installed and a special container agent).

- *Running containers from within a virtual machine that is managed by a hypervisor* – much like how typical VMs are hosted in the cloud: While VMs have to be run using real hardware which has an architecture that supports virtualisation, it is possible to run containers within virtualised servers. This can be useful as the hypervisor will provide another level of security to containers, in addition to the protection provided by *cgroups* and *namespaces*.

Figures 4.10 and 4.11 illustrate the deployment approaches. It is important to note the difference in the two deployment methods, as this will decide the implications associated with monitoring certain events. Also, in the discussion on the possibility of monitoring events in containers, it is important to distinguish between *system containers* and *application containers*. According to [44], system containers behave like a full OS and run init, inetd, sshd , syslogd, cron, etc., while application containers only run applications. Given that the container cannot access what it cannot see, this could mean that access to a system container may allow intruders more control over access to application containers. Application containers will only have access to the resources which allow running of the applications.

The monitoring techniques in this thesis have been dependent on the proposed taxonomy of security-related VM events. The taxonomy was general but it relied on

[17] Amazon Elastic container service https://aws.amazon.com/ecs/

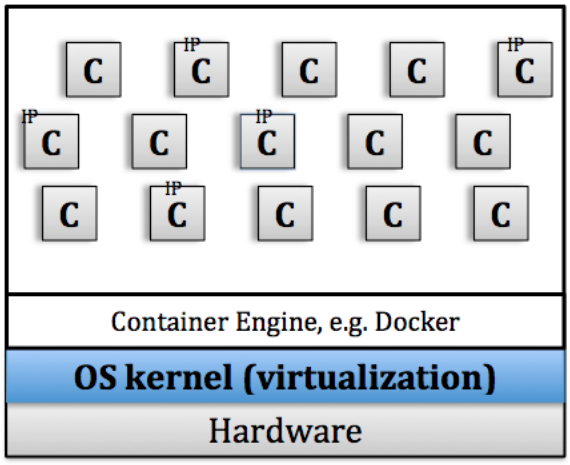

Fig. 4.10 First container deployment method: directly on top of OS

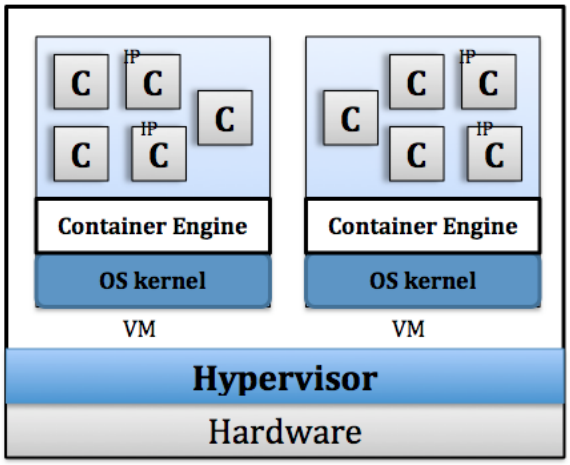

Fig. 4.11 Second container deployment method: from within a VM

events observed on full OSs; not on the application-level). Because of that, we argue that the mechanism may work with some system containers. Monitoring events of an application container may necessitate the development of new methods. We also argue that these methods could be application specific, as all applications vary in use and the resources they access or make use of. This challenge is similar to the challenge of monitoring applications in PaaS public clouds, especially if we consider the perspective of the consumer. If the containers are deployed in VMs, they can be susceptible to the same security challenges that we discussed in Chapter 3. For example, an intruder employing a side channel attack could spy on the user from another co-located VM, and the container's activities could be exposed.

We put forward that monitoring malicious events within containers is an interesting research area that is yet to be explored, especially when taking into consideration the growing popularity of containers. As limited research is available on concepts related to containers, an enhanced taxonomy of security-related events concerning containers would be useful.

4.15 Chapter Conclusion

The work in this chapter work arises from the observation that a consumer making use of a cloud system does not have access to operations that are performed on data or a VM hosted by a public cloud provider. There is an inherent degree of trust that a consumer must have in the provider. Other than supporting data encryption, it is hard for a consumer to know what operations have been carried out on their VM while it is deployed on the infrastructure of a cloud provider. The main aim of this thesis has been to monitor security-related events for the purpose of supporting trustworthy cloud computing, i.e. so the consumer can have a greater confidence in what is happening to their VMs which are hosted in the cloud. Thus, to achieve this aim, we first identified the security-related VM events of interest to us. This was mainly to provide a focused security monitoring; especially given that VMs have limited resources and monitoring should not affect normal operations of these VMs as much. Therefore, we proposed a taxonomy of security-related VM events which included six groups of events: *file-centric, life-cycle, network-related, process-related, user-access* and *attached-devices*. The *life-cycle* group of events involves events which are specific to VMs. Such events include: migration, creating snapshots, starting/stopping of instances, etc. To detect

and monitor VM security events, we proposed the design and architecture of a system to monitor, analyse and report the events to the consumer–referred to as *VMInformant*. The architecture allows launching monitors for the events according to the need of the consumer; by making use of the proposed taxonomy. In this chapter, we explained the rationale behind the VMInformant's architecture. Our aim is to enable a consumer to make an informed decision about which security related events are likely to be of interest (related to the Confidentiality-Integrity-Availability triad) and how recording these impact the overall performance of a VM. We have indicated why a classical Host IDS might not be suitable to run directly in VMs due to: (1) generating massive alerts; (2) higher rate of false negatives; and (3) higher rate of false positives. In our experiments, we focus particularly on events 1–7 discussed in Table 4.1, covering file-centric events. We compare both execution and storage overheads when generating log files. We evaluated the performance of VMInformant using the *Povray* and *Bzip* benchmarks, primarily observing file-centric events. From the results, we can see that the overhead is tolerable when recording events, but may become significant when these events are subsequently written to disk. If such events are kept in memory and only written on the completion of the benchmark execution, the outcome is almost equivalent to the original application. Using better and faster disks (or in-memory disks) can reduce the overhead considerably and at the same time achieve visibility about operations taking place inside a VM. Deciding how much data to record depends entirely on the type of security concerns a particular VM owner has. If the system is critical, it may become necessary to record all possible events.

We argued that if the cloud consumer is kept informed about security-related events a VM, this can significantly improve trust between the provider and the consumer. By coming up with taxonomy of security-related VM events and the architecture of VMInformant with its prototype implementation/evaluation, we were able to address two objectives. The first one is related to the types of security-related VM events which can be monitored from a consumer's perspective, while the second one is concerned with achieving a focused monitoring (technically) from the perspective of the consumer. In this chapter, we also provided a brief discussion on the possibility of detecting security-related events in containers. The discussion involved highlighting the key differences between VMs and containers in terms of deployment and security. We concluded that the mechanisms followed in this chapter to monitor security-related VM events may suit monitoring in *system* containers, while monitoring *application* containers might require devising new mechanisms. We also argued that if the containers are deployed

in VMs, they can be susceptible to the same virtualisation security challenges; that we discussed in Chapter 3. We concluded that monitoring security-related events within containers is an interesting research area that is yet to be explored. This is set as one of the future work to be undertaken.

Chapter 5

Detecting Migration of Virtual Machines

5.1 Chapter Overview

This chapter describes the detection of virtual machine migration from a user's perspective; surveying and comparing the potential detection techniques. It highlights our approach to detecting VM migration which combines multiple techniques (migration metrics) in a decision function. The result of the decision function gives an estimate of the occurrence of the migration. The chapter explains the algorithm used to apply the combined VM migration detection. To the best of our knowledge, this approach is the first which considers aggregating migration metrics in a decision function. It presents a prototype and a detailed evaluation of two approaches to detect the migration of virtual machines: (1) using one technique/metric; (2) aggregating multiple migration metrics. Part of the work in this chapter was published in [3].

5.2 Introduction

According to a survey by Rightscale which covered a large group of organisations, about 89% of the respondents were using public clouds [119]. IaaS public clouds have made it easy (and relatively inexpensive) for organisations to create virtualised servers (VMs) which can host their critical production services. We refer to these virtual servers that hold importance to the customer as *Critical VMs*. A critical VM is any VM that is crucial to the customer due to the tasks it performs or the data it processes or generates. For example, mission-critical defence applications used by governments do not tolerate

any leakage of sensitive data [41]. According to Gartner[1], there is an apparent trend for moving mission-critical applications to the cloud [52]. In public IaaS cloud systems, virtual machines can be migrated to another (jurisdictional) region/availability zone other than the region where the VMs were initially deployed. This is facilitated using live migration techniques [38] where VMs can be migrated with little or no downtime and without having to shut down services. There are many potential reasons for this, including: (1) load balancing requests across multiple physical data centres; (2) reducing the cost of energy in data centres by moving to regions where energy is cheaper; (3) performing maintenance of data centres.

This has often been identified as being of benefit for both the user and the cloud provider, in that the user does not see any interruption in service when the migration takes place, and for the cloud provider to benefit from reduced operational costs. The cloud provider generally does not directly engage the user in such a load balancing decision, so as to provide greater potential flexibility in undertaking such load balancing and reducing any overhead before such a decision can be made. However, such automated migration could also pose security concerns, especially when users are not informed about the migration event. With an emerging interest in supporting multiple cloud provider interactions whereby a cloud provider makes use of another provider as part of a service provision *chain*, suitable detection of migration events may become even more difficult to detect. We argue that hiding the migration event from users could introduce many risks and issues that include (but are not limited to):

- **VM theft:** the VM could be stolen after a targeted attack over the cloud provider's server. In this case, the VM and its data could have been migrated to a non-disclosed location as a consequence of the attack. It is possible that if the hypervisor is compromised, even the cloud provider will not know about the theft.

- **Violation of Service Level Agreement (SLA):** the SLA between the cloud provider and the cloud consumer could include clauses which stress the choice of a particular hosting region, otherwise terms of the contract could be violated. This could subsequently impact the established trust between the user and the provider.

- **Data privacy:** regions can have different laws which govern who has the right to access data in the case of incidents. Regions which differ in laws regarding

[1] http://www.gartner.com/

data privacy may use this as means to access the cloud provider's data. In this way, neither the cloud provider or the user will have control over the data, especially if it is in a geographical region that has differing data access regulations. This aspect assumes that VM carries its data as part of the migration action. Therefore, data made use of by the VM, which can be both application data and configuration information, needs to migrate with the VM.

- **Denial-of-Service (DoS) possibility:** migration could be triggered based on false alarms [8].In this way, the computing resources of the cloud providers can be under-utilised. This could impact on the overall quality of service offered and the potential revenue the provider could generate. Subsequently, if the migration is performed unnecessarily, the trust between the cloud provider and the consumer could be impacted.

In a survey of cloud consumers conducted by Fujitsu Research, it was indicated that 88% of such consumers are worried about who has access to their data [72]. In the case of migrating VMs to a prohibited region, data could be accessed by unauthorised entities. We argue that knowledge of migration events can help consumers make informed decisions as to whether to trust the provider or not, and whether action should be taken to terminate the VM. In Section 4.5 we developed a taxonomy of security-related events which included VM migration as an important event that needs to be reported. Although research literature mentions how live migration of VMs can be supported, such as in [38, 66], there is limited coverage about detecting live migration from a security perspective. Existing work primarily focuses on ways migration can be stopped, emphasising the need to detect migration in the early stages. Moreover, the motive for existing research is to prevent migration if it has already started.

Our research focuses on how we can make the VM itself *aware* of the migration event so that the VM owner/ user is informed of the location of their VM. We do not prevent migration of VMs (as this is an important capability within cloud systems to support performance and load balancing), but we do provide support for a user to track a VM after migration, or at least provide the facility to detect a probable VM location using a hybrid approach that combines many detection techniques.

In this chapter we highlight our work in detecting the migration of virtual machines from the consumer's perspective. We list the various potential detection techniques that could be made use of to detect or infer the migration, or provide a probabilistic estimate that VM migration has occurred. We compare these techniques and introduce a spatio-temporal taxonomy of these techniques which takes into consideration the point in time and location that migration can be detected. We argue that using only one technique or metric may be insufficient to indicate the migration event, so we also propose a decision function that combines several (measurable) metrics to detect migration. We refer to these metrics as: *migration metrics*.

We highlight and evaluate in detail two approaches to detecting migrations:

1. **First approach:** we apply a technique that is based on measuring the latency to specific "virtual" internet landmarks. An internet landmark is any reliable web server which could be used as a reference. In this approach, we selected a list of internet landmarks to maintain reference to VMs hosted on Amazon (AWS), by constantly record ICMP latencies. We found that this technique could be useful to estimate the probability of migration. However, due to its dependency on various network factors, it might result in inaccuracies. Hence, we argued that using this technique alone may not be sufficient from the perspective of the consumer to detect VM migration. This approach is highlighted in Section 5.9.

2. **Second approach:** We extended the previous approach with a different realisation, by considering ICMP ping interactions between the *critical* VM and *light-weight* monitoring VMs, and demonstrated how it can be made use of in practice. Light-weight monitoring VMs are small and low cost VMs whose sole purpose is to monitor or track the critical VM. In addition to the ICMP latency check, we consider three other metrics: *external/public IP address, hypervisor type,* and *hardware type*. We use the change in processor information to estimate the probability of hardware change. All these *migration metrics* are combined in function, which factors the *importance* of metrics from the perspective of the user. Therefore, the *importance* variable for each metric can be any value between 0 and 1. We highlight how to use the combined decision function to detect the migration of VMs within a *cluster* of VMs based on the input of the four metrics. A *cluster* of VMs consists of multiple virtual machines hosted together in public IaaS cloud systems, and are able to communicate with each other freely through

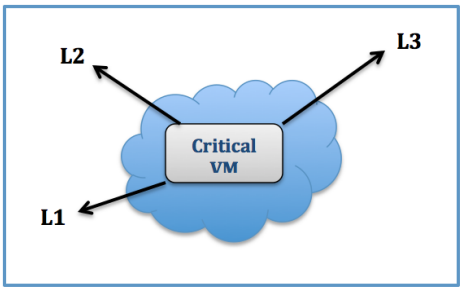

Fig. 5.1 First Approach: "virtual" network landmarks using few reliable internet locations to maintain reference to

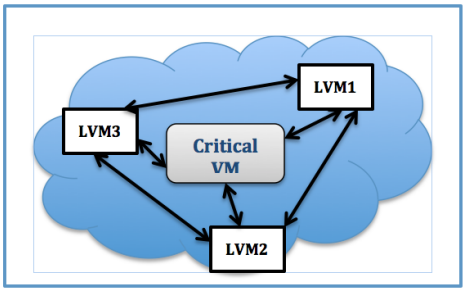

Fig. 5.2 Second Approach: combined approach using few lightweight VMs to monitor the critical VM

the virtual network interface card (NIC). This approach is highlighted in Section 5.10.

Both approaches use the concept of profiling to describe the state of the migration metrics of VMs. Information about the current state of the VMs is gathered based on the measured metrics, and we refer to this as the *normal profile*. We refer to the state of the migration metrics when they are calculated periodically as the *periodic profile*. The periodic profile is compared against the normal profile to find deviations in the metrics.

Figures 5.1 and 5.2 illustrate the two approaches. In the first figure, several internet landmarks are chosen, so the critical VM will use ICMP ping to form a *normal profile*. In the second, several light-weight VMs in the cluster are used as landmarks. However, there are ICMP ping interactions between all the VMs, and the normal profile will consist of three other metrics in addition to the ICMP latencies– as mentioned above.

This chapter makes the following contributions:

- We propose a spatio-temporal taxonomy of migration detection techniques. This considers whether migration of VMs occurs within or outside data centres, and also whether the detection happens before, during, or after migration takes place. This contribution is considered an extension to the work in [46]. In their work, they surveyed four types of migration detection techniques and focused on how monitoring of the migration event can be done: internal (inside the VM) or external (from a remote server). Our coverage of the techniques include five additional detection techniques. We focus on the stage at which migration can

be detected. Also, whether the detection technique is useful within the same cloud data center or across data center in other regions.

- We provide a comparison between different migration detection techniques. This can help in selecting the appropriate VM migration detection technique.
- We present a discussion on related work on exploiting VM migration and the detection of migration.
- We propose a prototype implementation of a tool to detect migration of VMs in IaaS using an approach called Virtual Remote Landmark Fingerprint.
- We develop an algorithm to detect VM migrations in public IaaS cloud systems. To the best of our knowledge, our combined approach is the first which considers aggregating migration metrics in a decision function. The function works with any number of migration metrics specified by the user.

5.3 Motivation for Detecting Migration from the Consumer's Perspective

Cloud providers often make use of live migration to satisfy the required availability in consumer Service Level Agreements (SLAs) or to improve the quality of service. As long as SLAs are being met, why would consumers be interested in migration events? Why is being informed about such migration events useful for establishing trust in cloud systems? The answer to these questions stems from the variety of cloud security issues. More specifically, the security issues related to the hypervisor. In Chapter 3 we have already established that the virtualisation platform suffers from a variety of security issues. In the IaaS model, attacking the hypervisor would mean compromising all the running VMs (managed by the hypervisor). This is mainly because hypervisors have more privileged access to hardware resources than typical applications [149]. If the hypervisor is compromised, then the migration module, which is responsible for migration, can also be exploited to migrate the VM to an unknown place. The hypervisor is able to suspend a VM at any point during execution, make a snapshot of its CPU states, memory and disk, and resume the snapshot later without without the guest VM being aware of this [158].

If the previous scenario occurs, the migration event may not even be detected by the cloud provider. We therefore emphasise the importance of detecting migration by

the consumer directly or via a trusted third party service. Another reason for detecting such an event from the consumer side is that the cloud provider may not reveal to the consumer that a migration has taken place [46]. Where a cloud federation is involved, the VM may be migrated to some other compatible cloud provider without informing the consumer. According to [24], if a VM is migrated to a remote location crossing national borders, internal data can become subject to new legislation. In this case, the VM could fall out of the control of both the owner and the cloud provider.

While there are many research papers which cover cloud security issues, including hypervisor issues, such as [11, 13, 75, 48], a limited number have focused on how live migration of VMs can be used to cause attacks. Initial research on exploiting live migration analysed man-in-the-middle attacks by manipulating VM memory during a live migration procedure with the help of a malicious modified router [102]. Fiebig et al. [46] provide a possible scenario for VM theft that starts with compromising the hypervisor migration module. The attacker may then use IP tunnelling to migrate the VM to another host on a different subnet (owned by the attacker). Xia et al. [158] highlight the rollback attack, in which a compromised hypervisor runs a VM from an old snapshot without the user's knowledge, thereby exploiting the fact that it is hard to distinguish between normal suspend/resume and migration operations. We argue that it is important to keep the user informed of all migration events that happen to the virtual machine.

5.4 Related Work

In the live migration of VMs, the entire state of the VM, including e.g. memory pages, are transferred to the destination host and the VM can resume its execution from its state prior to migration [14]. Many researchers have explored the live VM migration process, such as in [38, 145, 66, 27], whereas limited coverage exists related to techniques for detecting VM migration. We have already covered the process of live migration of VMs in Chapter 2. Fiebig et al. [46] generalised the stages of live migration of virtual machines into three phases: *memory copy, CPU copy, and switch.*

1. *Memory copy phase*: the VM memory state is copied in order to start the migration process.

2. *CPU-copy phase*: the CPU is stopped and the registers are copied from the source server to the destination server.

3. *Switch phase*: the VM is fully removed from the source server and started in the new destination server.

It is worth mentioning that not all public cloud providers support direct VM live migration. For example, to the best of our knowledge, Amazon Web Services (AWS) still does not support live migration. However, live migration is supported by the Google cloud platform. It is also important to note that, in our work, what matters to us is the fact that the VM could be migrated, regardless of the means. Thus, the migration event that we would like to detect does not need to have occurred live. The live migration process, which happens in IaaS, could be the means by which a VM has been migrated, exploiting the fact that consumers may not notice the occurrence of the event.

Latency and time-lag measurement techniques appear to be the most widely used. Konig and Steinmetz [83] identified the importance of considering communication network monitoring and found that the ICMP-ping Round Trip Time (RTT) during the live migration is a promising metric for detecting migration. In their research, they send ICMP packets with a size of 64 bytes at an interval of 0.1 seconds from and to the VM being migrated. They examined the migration process both locally and remotely using different CPU loads, and in both cases report peaks on the round-trip. Several research papers used the approach in [83] with different realisation. Gottron et al. [59], for example, used latency measurement to visualise the migration process for VM owners. They stream a video on the VM which is to be migrated to make sure it can be viewed continuously without interruption.

We argue that this technique of monitoring ICMP latency for a short monitoring interval is inappropriate when considering a critical VM within a cluster in an IaaS public cloud. The reasons for this are: (1) we are interested in detecting migration that has already occurred; (2) we do not know when the migration is going to occur, hence it is not feasible to measure the latency continuously, as this is likely to result in notable performance degradation. Thus, when we consider the combined approach, as will be highlighted in Section 5.10, we cannot rely on monitoring ICMP latency for very short intervals.

Fiebig et al. [46] used a hybrid approach of delay measurement with ICMP ping and time-lag measurement, using a heuristic based on the characteristic features of the ICMP ping round-trip times. These characteristics are: increased average round-trips, very high outliers, and unanswered pings in the second and third phases of migration – namely: CPU copy and switch. In their experiment (which made use of a Network Time Protocol (NTP) server to sync clocks) they found that time-lags are over 100 ms. They argue that a low monitoring interval is possible because it takes a while for the time value to converge after synchronising with the NTP server. Biedermann et al. [24] also used a hybrid approach (i.e. using multiple measurements) to detect migration, but they were interested in delaying the migration process to allow time to execute the security policies. They developed a Live Migration Defense Framework which uses machine learning techniques to identify the current location of a VM (on the Internet) based on generated network latency fingerprints to fixed remote landmarks. Such fingerprints can be, according to them, associated with unique locations. The VM then starts self-latency mechanisms (slowing down) in order to execute security policies and verify location. Slowing down of the ongoing live migration was performed by intensively altering the memory pages of the VM, which causes re-transmission of packets. It then checks for a specific kernel event to detect the end of the live migration events, and compares the latency fingerprints to detect if the VM is within the same data centre or not – raising an alarm if it is not. As the work involved delaying migration, preference was for detection at an early phase.

5.5 Migration Detection Techniques

Techniques used to detect VM migration by a consumer often require additional input from a hypervisor. Fiebig et al. proposed taxonomy of VM Live Migration detection [46] . The taxonomy consists of four detection techniques: (i) hypervisor detection, (ii) hardware detection, (iii) time-lag detection, and (iv) delay measurement. We extended this taxonomy with additional detection techniques which we believe they could improve the accuracy of detection. The following subsections list the detection techniques:

Hypervisor detection

Because the hypervisor manages the lifecycle of all VMs hosted on a system, when VMs are migrated to a different hypervisor, such an event can be monitored and reported.

According to [46], different types of hypervisors have different fingerprints. This can be used to detect if a hypervisor is changed or updated, i.e. another version of the hypervisor. Ferrie et al. [45] identifies how widely-used hypervisor systems, such as VMWare and qemu-kvm (amongst others) can be detected. This work highlights an attack that is activated when malware detects that it is running on a VM and not a real machine. Raffetseder et al.[112] have also covered the detection process of *qemu* and listed several checks that can be used in the detection.

In practice, due to the lack of compatibility, VMs do not benefit from the live migration to a host with a different type of hypervisor unless both the VMs and the hypervisor are OVF compliant. OVF[2] is a platform-independent, open packaging and distribution format for virtual machines. This means that if the VM is packaged using the OVF format, there is a possibility that it can be migrated to another type of hypervisor which is OVF compliant. In the case where OVF compliance was not achieved, detecting a different *version* of the hypervisor may indicate VM migrations.

Hardware detection

As the VM runs on a physical server which has specific hardware characteristics, after migration there could be detectable traces related to the new hardware hosting the VM. For example, a change in the CPU execution speed or link speed [46] may indicate a change in the hardware. One way of detecting such changes in the hardware is by using appropriate hardware benchmarks. In [138], Sonnek and Chandra distinguish between two concepts: *physical footprints*, and *virtual footprints*. The physical footprint is the amount of physical resources consumed, e.g. CPU time, storage (memory and disk), network bandwidth and power, while virtual footprints are location-independent VM characterisations. These show how knowledge of the virtual footprints can be used to reshape the physical footprint of a VM such that both the cloud provider and the consumer are satisfied. An example of reshaping is to migrate the VM to another server based on the virtual footprint of the VM, or to migrate the VM closer to its file system.

Time-lag detection

According to the migration steps covered in Section 2.3.5, the VM stops for some time before it starts again in the new host. Although the duration is very short, it is noticeable – thereby providing a way to detect the migration. After migration, the

[2]https://www.dmtf.org/standards/ovf

time continues from the exact time as before the migration but then converges to the proper time [46]. In [28], Broomhead et al. highlighted the problem of time-lag during and after live VM migration. The slow converging of time allows the detection of migration. In this case, the migration will have occurred already.

Delay (latencies) measurement detection

Measuring the delay in receiving ICMP ping replies has been demonstrated to be a suitable way to detect migration by [83]. They discovered that the round-trip time of pings shows a higher average during the migration process, as discussed earlier in Section 5.4.

Another metric which is related to measuring the latency is the number of network hops. The hop count refers to the number of intermediate devices (like routers) through which data must pass between the source and the destination. Theoretically, data between nodes in the same cloud data center pass through fewer hops. In [43], they suggested setting a threshold value such as the median of the number of hops in order to decide whether a group of nodes are local –in the same datacenter. In that sense, if the number of hops between any two nodes is less than or equal to the threshold value, nodes are said to be local in the same datacentre. In practice, some cloud providers may prevent displaying the hops or the hops count, which may limit its applicability in detecting the migration of VMs from one region to another. This can be achieved using firewalls. As a proof of concept, we used the 'traceroute' tool to check the number of hops between VMs hosted in same zone in the Google Cloud Platform. The number of hops was only one. Testing the tool between a VM hosted in the US and another one in Asia within the Google Cloud Platform has also shown details of only one hop. This means that we cannot rely on the number of hops metric in this scenario unless there were methods to bypass the firewall and find the exact number of hops between the VMs.

HTTP throughput

Network throughput is the rate of successful message delivery over a communication channel. Bradford et al.[27] discovered that the migration process causes a drop in HTTP throughput, as part of the network is used for bulk transfer. This drop in the HTTP throughput can be used to detect migration in environments with limited network properties, e.g. available bandwidth.

IP change detection

According to [27], when migration takes place between servers on different networks, the migrated VM has to obtain a new IP address, and thus existing network connections break. This IP address change can be used to detect migration of VMs across different sub-nets. However, Biedermann et al. [24] argue that generating network fingerprints which are based on DNS databases (geolocation) or IP addresses is not meaningful, as network surroundings can be easily manipulated and network paths can be rerouted after a live migration using a VPN, for example . Although attacks can occur on this mechanism, we argue that it can be useful in cases where a cloud provider is exploiting migration to achieve cost saving without criminal intent.

Monitoring kernel events

After the migration process is performed successfully , the guest OS has to be modified to support the sharing of drivers [24]. The modification causes particular kernel events. In VMs hosted by the Xen hypervisor, the *"suspending xenstore"* event can be detected, which indicates that the migration was successful. The *xenstore* is a database that is used to share information between the hypervisor and the VMs, mostly concerning status and configuration. This detection technique can be used to detect migration *after* it occurs. However, this technique may be environment specific, e.g. using a particular hypervisor such as Xen.

Remote Landmark Fingerprinting

This method was highlighted by Biedermann et al. in [24]. It is based on measuring latency to several remote fixed targets on the internet (landmarks), thereby providing a unique fingerprint for each location of the VM. They can estimate the location of the VM with reference to previous locations to infer if the VM was moved to a distant location, i.e. not stated in the SLA. Usually, machine learning algorithms are used to deduce any underlying pattern in the migration process and identify potential locations. In Section 5.9, we employ this technique to detect migration but with a different realisation that uses general internet landmarks.

Hypervisor-assisted detection

Our goal is to detect migration from inside the VM itself or provide means to do so – the basis for techniques covered previously. We argue that if there is a *channel*

between the hypervisor and the VM, then information related to migration can be easily communicated to the VM, and hence to the VM owner. There are tools available which could facilitate this. Libvirt [25], which is a toolkit used to interact with the virtualisation capabilities of recent versions of Linux and other operating systems, can be used to manage the life-cycle of a virtual machine. This includes managing the migration of the VM from one server to another. If the migration process is successful, an event can be generated to confirm this. If there is a service in the VM that is listening for such event originating from the libvirt API or similar broker, then migration events can be analysed and audited. Of course, one can argue that this is not cloud provider independent. However, the realisation is via the VM itself (based on hypervisor provided information). We argue that this can provide a useful and more accurate alternative where available, and can be useful in a *federated* cloud context.

5.6 Comparison of the Migration Detection Techniques

To the best of our knowledge, no research has provided thorough coverage or a comparison between the different migration detection techniques. Table 5.1 compares the various discussed detection techniques qualitatively using a variety of criteria including:

- **Simplicity of implementation:** How challenging is the implementation of such technique? This can be subjective, but our purpose is only to give an estimation assuming that no existing tools are widely available to perform the detection.
- **Migration phase:** for which the detection technique is suitable for, i.e. (1: Memory Copy, 2: CPU-Copy, 3: Switch) - according to the discussion in [46] and Section 5.4.
- **Storage overhead:** Detection usually requires the storage of metrics in the VM. This is usually proportional to how frequently the metrics are checked. Some detection techniques require more frequent checks, which may result in storage overhead.
- **Processing overhead:** Processing of the metric data to detect migration may result in performance overhead, especially if computations are required prior to arriving at the decision.

- **Hypervisor-specific?:** This criterion is concerned with whether detection depends on a certain hypervisor or it is hypervisor-independent. The dependency on a particular hypervisor may limit the usage of some techniques.
- **Suitable for local or regional or both:** This refers to whether the detection technique is suitable for detecting the migration of VMs within the same data centre, or if the migration was to a regional site, or both.

Table 5.1 shows that the implementation of both *hypervisor detection* and *hardware detection* techniques can be challenging as it requires knowledge of low level machine language as well as many specific technical details. In [45], for example, detection of the emulators required the use of assembly code. There are tools which could reveal the hypervisor type, such as *virt-what* (we made use of it in Section 5.10), but there is no guarantee that the information has not been modified by the hypervisor. Some detection techniques have average implementation complexity. For example, the latency measurement technique requires comparing the ICMP ping requests and observing the latencies. This requires the use of statistics or machine learning algorithms, or checking of the deviations in latencies, which means that some of the latencies or fingerprint profiles will be stored in the VM. Both techniques of detecting IP address change and monitoring kernel events have a simple implementation burden, simple storage overhead, and a simple processing overhead. In the IP address change detection, the current external IP address can be compared to the previously stored one which can be obtained automatically from the machine itself. Similarly, in detection using monitoring specific kernel events, the process only requires checking for the existence of a specific kernel event, e.g. the "suspending xenstore" event.

Table 5.1 Comparisons between the migration detection techniques

Detection Technique	**Phase**	**Simplicity**	**Storage overhead**	**Processing overhead**	**Hypervisor specific**	**local?/regional?**
Hypervisor detection	3	challenging	minimal	minimal	yes	both
Hardware detection	3	challenging	minimal	minimal	no	both
Time-lag detection	2	average	Minimal	minimal	no	regional
Latency measurement	2	average	high	high	no	regional
HTTP Throughput dropping	3	average	average	average	no	regional
IP change Detection	3	simple	Simple	Simple	no	regional
Monitoring kernel events	3	simple	simple	simple	yes	both
Remote landmark fingerprint	3	average	high	high	no	regional
Hypervisor-assisted	2,3	average	simple	average	no	both

5.7 Spatio-temporal based taxonomy of the migration detection phases

After reviewing the literature on migration detection, we have noticed that the migration event may be sensed at various stages: (1) before migration happens; (2) during the migration event itself; and (3) after the migration event has commenced. Not all detection techniques may be appropriate for all these stages. In addition, given that migration could happen internally within the data centre, some of the techniques may be more suitable for detecting migration which happens regionally in a different location.

In this section we introduce a spatio-temporal taxonomy of the detection techniques based on detection phases. This taxonomy takes into consideration the point in time migration can be detected and also the relative location (within the data centre or outside). We aim to map the spatio-temporal phases to the appropriate detection techniques described earlier in Section 5.5. The motivation for creating this taxonomy stems from our aim of detect the migration of a virtual machine in an efficient way. We do not intend to stop the migration or delay it, so some of the techniques may not be needed in all situations. We just need to report potential migration events so we can report them to the owner to make decisions.

Phases of the suggested spatio-temporal based taxonomy of the migration detection include:

1. **Migration about to happen:**
2. **Migration is happening:**
   (a) Migration is happening at an early stage
   (b) Migration is happening at a late stage
3. **Migration has happened:**
   (a) Migration has happened in the same data centre
   (b) Migration has happened to a different data centre, e.g. regional.

The taxonomy can be seen in Figure 5.3. It can be considered an extension to the work of [46]. This work developed a taxonomy of live migration detection which included techniques for internal and external detection. Our coverage of the

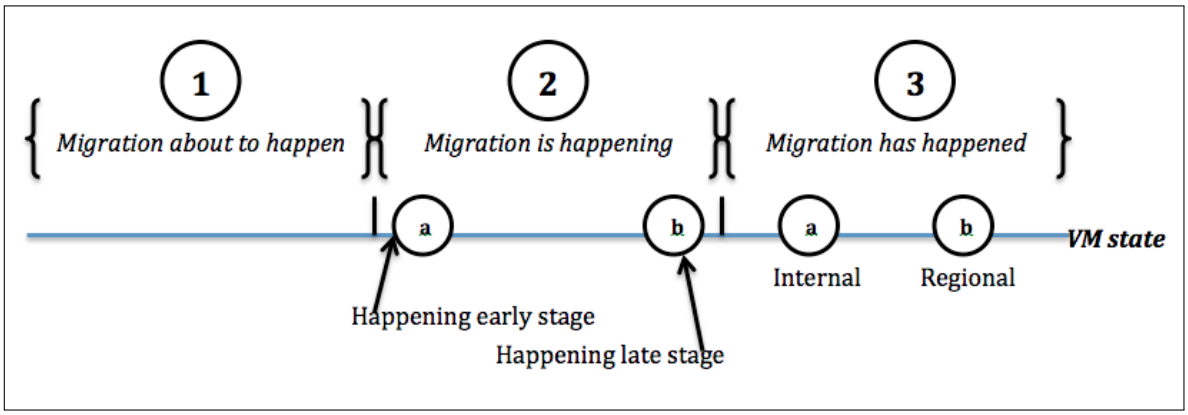

Fig. 5.3 A spatio-temporal based taxonomy of the migration detection techniques

techniques is more comprehensive, as seen in Section 5.5. Table 5.2 shows the mapping of the techniques to the different taxonomy phases. The table shows how the latency measurement technique is suitable for detecting migration at earlier stages due to the instant deviations in ICMP pings. Hence, this technique is useful for delaying the migration early or launching security measures, as done by [24]. For instance, when they delayed the migration by intensively writing to the VM memory. The table also shows that most of the detection techniques are suitable for the last phase in spatio-temporal based taxonomy, i.e. when the migration has already happened. In this case, it is already too late to prevent the migration. However, detection is still useful in order to audit or arrive at an informed decision regarding the trust between the cloud provider and the consumer. In the CPU-Copy phase (refer to Section 2.3.5), the CPU is stopped, so the VM clock will not be accurate, which indicates a time-lag. This can be detected either while the migration is happening or more clearly after it has already occurred -before the time converges again using NTP (Section 5.4). HTTP throughput drop occurs just before the VM resumes on the destination server, which means that the migration is either taking place or has already occurred. It may not be a good metric to check if migration prevention is needed, but it is a good positive indication of the occurrence of a migration event, provided there are tools for it. As can be seen from the table, both IP change detection and remote landmark fingerprint detection techniques can be used in the last phase of the spatio-temporal based taxonomy, but to detect migration to a distant server. Generally, when a VM is moved in the same data centre, its IP address stays the same [24]. However, it has to obtain a new IP address if it is moved to a distant data centre in another sub-net. Similarly, remote landmark fingerprint detection will not be effective unless the deviations in latencies indicate the migration of the VM to a distant data centre. If no notable deviations are observed,

this may indicate that the VM has not been migrated outside the data centre, or it has been migrated, but to another host within the same data centre. However, it will be difficult to verify that using this technique. Detecting kernel events is suitable for the last phase also, but as mentioned before in Section 5.6, it can be hypervisor-specific.

Table 5.2 Time-line based taxonomy for the migration detection techniques

Detection Technique	**spatio-temporal phase**
Hypervisor detection	3
Hardware detection	3
Time-lag detection	2,3
Latency measurement	1,2
HTTP Throughput dropping	2,3
IP change Detection	3(b)
Monitoring kernel events	3
Remote landmark fingerprint	3(b)
Hypervisor-assisted detection	3

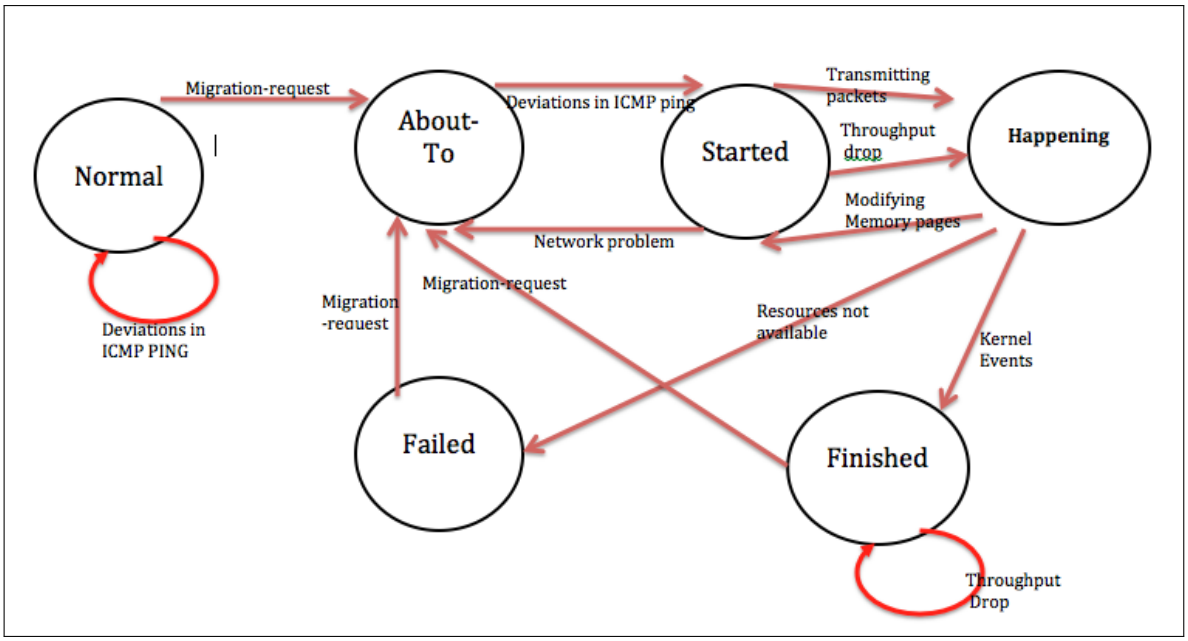

Fig. 5.4 A finite state-machine diagram for the spatio-temporal based taxonomy

Figure 5.4 shows an illustration of the spatio-temporal based taxonomy using finite state machines. The figure consists of six states and various transitions which represent operations that can trigger the states. The *normal* state is when the VM is running normally in the physical machine of the IaaS public cloud system and no migration

is taking place. When there is a request to migrate the VM, deviations in ICMP ping can be observed. At this moment, it can either start or fail if resources are not available in the destination. Network problems could cause the migration to restart. If the migration request is successful, packets will be transmitted to the destination and throughput drop may be observed. As it is taking place, migration can revert to the *start* state if memory pages are corrupted. When the migration is successful, depending on the hypervisor platform, kernel events may be triggered indicating that migration is complete.

5.8 Hybrid/Combined Migration Detection Approach

Our methodology of reporting migration events is based on the *VMInformant* architecture [5], as covered in Chapter 4. In this work, a taxonomy of security-related VM events, broken down into six classes of events, is also presented – with migration detection being a key event in the VM lifecycle. We argue that the use of one technique may not be sufficient to detect the migration event, and a hybrid approach which combines several detection techniques (metrics) in one framework is likely to be more accurate. We make use of the following techniques where possible: (i) time lag detection; (ii) latency/remote landmark fingerprinting; (iii) HTTP throughput drop; (iv) IP change; (v) monitoring of kernel events; (vi) hypervisor change detection; (vii) hardware change detection. A hybrid approach that provides a weighted combination of these techniques enables reduction in potential false negatives, as correlation between such event types can provide greater accuracy.

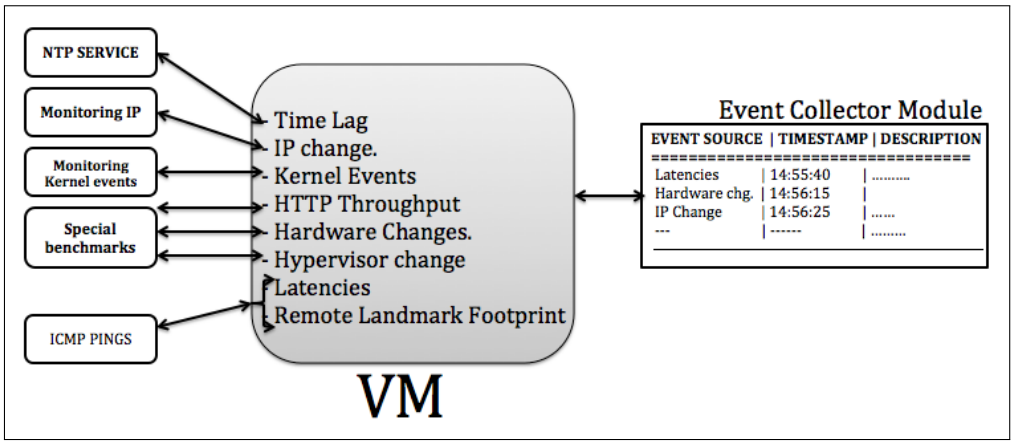

Fig. 5.5 Detecting migration

Figure 5.5 shows the suggested combined framework, which can use a number of open source monitoring tools. For instance, to detect time-lag, sync between a time server using NTP can be employed. To detect hypervisor, hardware or throughput changes, suitable benchmarks can be run periodically. ICMP pings to the local host, or a predefined internet landmarks can be used to monitor latency.

As a proof of concept, in Section 5.9, we use a remote landmark fingerprint technique based on ICMP pings (Section 5.9.5). As soon as the events are generated, they are sent to the Event Collector Module, which organises them in a suitable format. The source of the event and a time-stamp will be written to disk. Any additional description will also be appended. The generated file can later be used for analysis. However, despite the benefits of using a hybrid approach to detect migration of VMs, the storage needed to record such events may increase considerably (as additional detection frequency is used). These techniques can be combined into a decision function $d(m)$ to control when the alarm is raised, as follows:

$$\begin{aligned} \mathbf{d(m)} = \frac{1}{\sum_{\forall i} a_i}(a_1(&hardware\ detection)+ \\ &a_2(IP\ address\ change) + ...+ \\ a_n(&HTTP\ throughput\ drop)), (0 < a_i < 1) \end{aligned} \tag{5.1}$$

Each a_i represents the importance of a detection technique (migration metric) from a user's perspective – often based on the accuracy with which this metric can be measured. We assume each detection technique (e.g. IP address change) results in a binary outcome (i.e. whether a migration is detected or not). The result $d(m)$ is a weighted combination of various migration techniques and describes the likelihood of a migration having taken place.

It is important to mention that our research does not intend to make recommendation as to how the migration detection parameters should be set. The *importance* of the metrics values are decided by the cloud consumer. However, it is assumed that between the end user and the provider sits an intermediate party who understands security and the implications of choosing the parameters. Hence, this party can determine the values of the parameters based on past experience or based on some data mining tools such as: feature selection, component analysis, etc.

In Section 5.10 we illustrate how this hybrid approach can be used and the detection algorithm associated with it.

5.9 Approach 1: Virtual Remote Landmark Fingerprint

5.9.1 Overview

In this approach, we choose the remote landmark fingerprint technique, used in [24]. An internet landmark is any reliable web server which could be used as a reference. There are three reasons for using this technique: (i) we are not interested in the start of the migration process, (ii) it will provide a better probability that the VM was moved to a distant location rather than within the same data centre; (iii) we want to support our argument that using only one detection technique may not be sufficient. Our approach: (i) we considered general but reliable landmarks (in [24], they used AWS root servers); (ii) we considered several variables, e.g. *varying number of ping intervals* to prevent IP blocking from a server, *parameter thresholding, e.g. min/max on latency*, etc.; (iii) provided a more detailed design and implementation details of the technique.

5.9.2 Methodology

Our methodology of detecting or estimating the occurrence of the migration event or critical VMs based on virtual landmark fingerprint can be summarised as follows:

1. We identify the critical VM of interest hosted in the public IaaS cloud system.
2. 2. We select reliable internet landmark locations and record the IP addresses. We could use the full DNS name, but doing so can result in different IPs every time. Therefore, we favour IP addresses.
3. We use ICMP ping to measure the latency from the VM to the selected landmarks a number of times with varying intensity. This can be performed at different times of the day. We record the latencies and calculate the averages.
4. Based on the outcome of the previous latency calculation, we form a fingerprint for the VM–we call it *VM normal profile*, which will consist of the average latencies to the selected landmarks.

5. Periodically, according to the configuration chosen by the owner and a specific threshold factor, latency will be re-calculated and checked against the VM normal profile.

6. If the obtained latency calculations do not satisfy the VM normal profile values and the minimum/maximum allowed threshold range, an alarm is sounded and the migration is said to have likely occurred.

5.9.3 Requirements for choosing internet landmarks

Since the approach depends on the selection of a suitable internet landmark, it is important to specify certain selection criteria. We want to use these landmarks to infer the relative change in the location of the VMs, therefore we argue that the landmarks should be:

- **Reliable:** If the landmark is not reliable, the latencies will not be consistent, and this will result in inaccuracies calculating the initial VM Virtual Landmark fingerprint (VVLFP) or the subsequent calculation of it in order to compare results.

- **Available:** ICMP Ping will be used to calculate the VVLFP, so if the landmark is not available or reachable, comparison cannot be made.

- **Close to the VM:** to infer the relative change of location, the chosen landmarks are preferably located close to the data centre which hosts the VM. Latencies will normally be small in this case, hence comparison will be easier. If the landmarks are distant and the VVLFP was obtained and then VVLFP is calculated again, then it may not show a difference, despite the fact that migration could have occurred-(false positives).

- **Multiple:** Theoretically, the more landmarks chosen the more accurate the decision on whether the VM has been migrated or not. We will base our discussion and experiments on three landmarks. More can be chosen if needed.

5.9.4 Migration Monitoring Framework: based on virtual landmark fingerprint

Our general migration monitoring framework consists of the following components:

1. **ICMP Ping Launcher:** The ICMP ping requests have to be issued frequently – a parameter that needs to be adjusted by the consumer. Issuing too many pings could result in the destination server blocking the requests. Another aspect that determines the frequency of ping requests is the packet size to be sent. As this technique is based on fixed landmark locations, the component makes use of the specified landmark (internet) locations before launching the ICMP pings.

2. **Latencies Collector/Trainer:** In order to start the training process, latencies, which are responses to the ping, have to be recorded and used to create the unique landmark fingerprint.

3. **Latencies Correlator:** Machine learning techniques can be used in this component to find deviations in latency values compared to the fingerprint. This component accesses the unique VM fingerprint and compares it with real time latencies. If the latencies are not within a specific range, then migration is said to have occurred with a particular probability.

4. **Migration Notifier:** This component is responsible for notifying the Event Collector module, which in turn feeds the captured events to the *Event Aggregator* component in *VMInformant* [5]– as covered in Chapter 4.

Figure 5.6 shows the suggested framework with all the components. As can be seen, the framework was made with the architecture of VMInformant in mind, as discussed in Chapter 4. The framework is intended to feed VMInformant with VM migration-related information. The integration process will be discussed further in Chapter 6. Figure 5.7 shows a high level view of the VM and the landmarks which form its unique fingerprint (normal profile). If the VM is moved to another (remote) data centre, the captured latencies are expected to change, and will differ from the initial fingerprint.

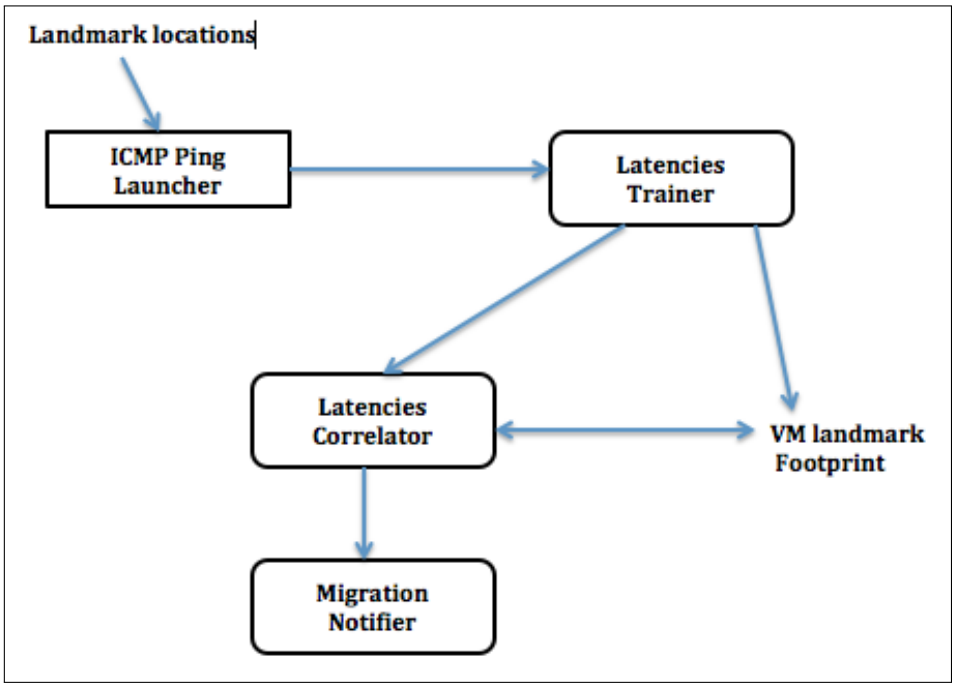

Fig. 5.6 Migration detection framework

5.9.5 Prototype design & implementation of the migration detection module

In this section we present our suggested prototype design and implementation for detecting the migration of VMs . It has a simple and intuitive graphical interface to allow the VM owner/administrator to easily manage the detection process. There are a few text boxes for inputting the server names or IPs (**landmarks** (LM)), as shown in Figure 5.8. For each LM there are *Avg Latency 1*, *Threshold-min* and *Threshold-max* boxes. The *Avg Latency 1* contains the average of all the latencies from the ICMP *ping* attempts. The number of ping attempts is determined by the value of the ping *Counter text box.* For example, if the number is 100 then the ping to the servers will be attempted 100 times before arriving at an average value to fill in the Avg. Latency 1 box. The *Threshold-min* and *Threshold-max* boxes will contain the latency range that is acceptable and not considered an anomaly. The Threshold-min and Threshold-max values are generated by subtracting and adding the value of *Avg latency1* $\times$ *Threshold factor* respectively. For example, if the Threshold factor is 0.1 this means that 10% of the *Avg. Latency* is acceptable. Any results beyond this value are an anomaly. The *calculate Latencies* button will just fill in the *Avg. Latency 1* boxes and the *threshold values* will be calculated automatically.

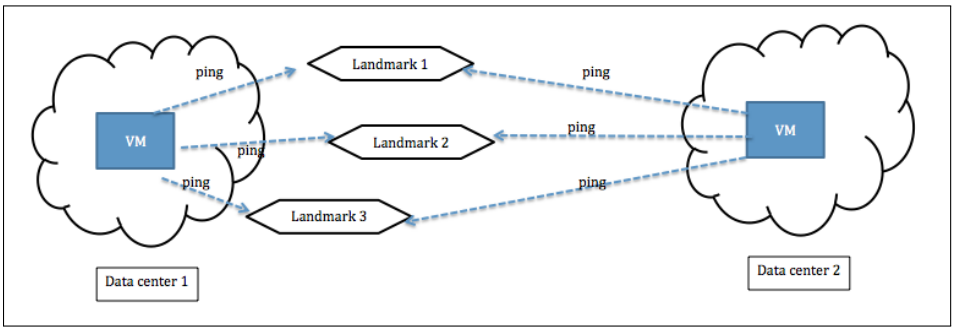

Fig. 5.7 VM landmark fingerprint based on ICMP Latencies

Re-calculation of latencies can be configured to be undertaken *automatically* after a specified interval or *manually*. In each case, the *number of required pings* can be set. This is different from the number of pings set previously. If after recalculating latencies it is determined that at least one of the Avg. Latencies2 of the hosts are *not* within the allowed Threshold range, then an alarm is raised. The time to complete the ping operations and the relevant calculations will be displayed in the corresponding boxes.

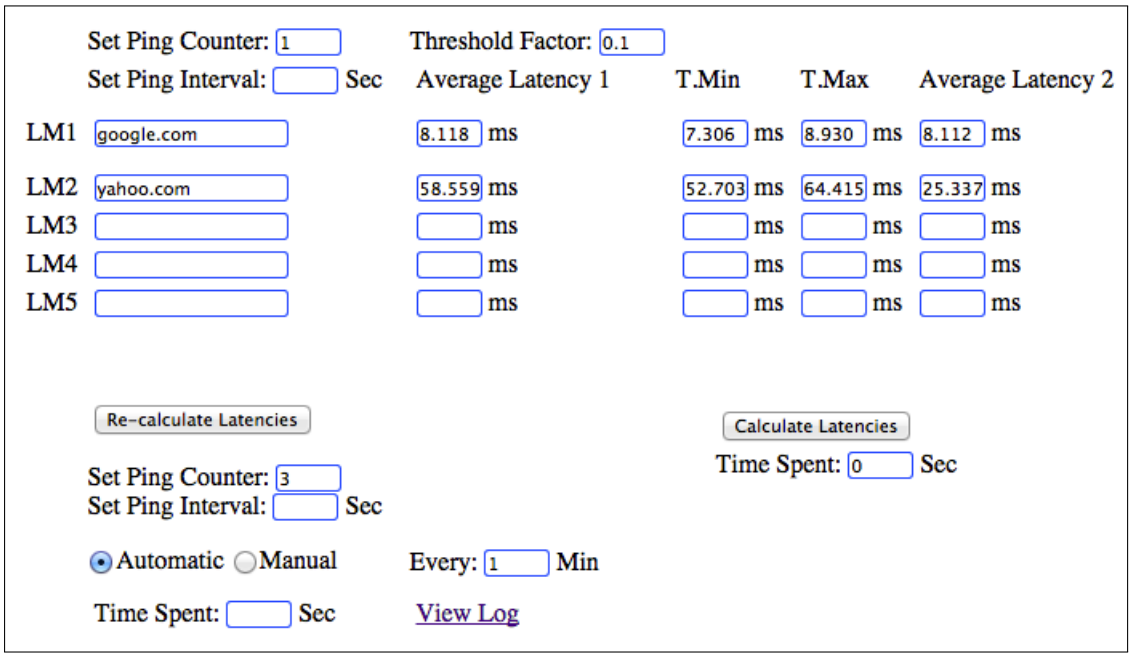

Fig. 5.8 A sample illustration of the implemented prototype

Figure 5.8 shows the actual design of the implemented prototype which is supposed to run in the monitored VM. It was written using PHP as a web application which contains an easy to use interactive interface. The web application allows the consumer/owner to specify the IP addresses/names of the landmarks (initially up to

five landmarks). After setting the: ping counter, ping interval and the threshold factor values, the application will start calculating ICMP latencies (By pinging IP addresses of the landmarks from the VM). ICMP pings along with other operations are executed in the background, and all of this is performed behind the interface using PHP (mainly via the `shell_exec` command). Results are stored in a log file and current latencies values are stored temporarily in a text file in order to be compared later when latencies are re-calculated. Recalculation of the latencies can be carried out either manually or automatically. In either cases, the latencies are going to be compared with the previous ones, and if deviations are encountered an alert message will be displayed clearly and the event will be logged in the log file with a timestamp. The prototype was a proof of concept to allow the performing of experiments. We added a feature that determines the time the application spends calculating the latencies. It was configured to run automatically after system restart in case of system shutdown or reboot. In general, the prototype has three phases:

1. **The training phase:** Latency profiles are being calculated.
2. **The settlement phase:** Latency threshold values are being determined based on the Threshold factor. In this phase, the VM profile has already been generated and decided.
3. **The detection phase:** This is where re-calculation of the latencies will occur in order to check whether the values are within the allowed range.

The detection phase is assumed to take place after the migration has occurred.

5.9.6 Experimental design

Based on a prototype to detect the VM migration using the VM Virtual Landmark Fingerprint (VVLFP), several experiments will be performed. These experiments include:

- *Using distant and nearby landmarks*: The terms "distant" and "nearby" are relative – and in the context of a communication network cannot be characterized purely on geographical distance. Our choice of these terms reflects the average latency observed between a VM and one or more potential landmarks.
- *Using a small and large number of ping attempts to find avg. latencies*: i.e. to check the efficiency of the training and also the processing time.

- *Using different threshold factor values:* to see how this affects accuracy. We assume that the greater the value for the chosen threshold, the greater chance that migration detection could be missed.

5.9.7 Results & evaluation

To show a proof of concept of the migration detection process using the virtual landmark fingerprint technique, Amazon Web Services (AWS) was used. Two Ubuntu instances were rented in order to carry out the initial experiments. One VM instance was hosted in the USA (Oregon region). The other was hosted in Asia (Singapore). Since the migration detection prototype application that we developed was web-based, an Apache web server was deployed in both VMs in order to run the web application.

Scenario 1: Nearby landmarks

For evaluation purposes, we chose three well known internet landmarks (L1, L2, L3) which are located in the US (to act as relatively nearby landmarks, as follows:

- L1: Google (IP Address: 173.194.116.48, California)
- L2: Yahoo (IP Address: 206.190.36.45, Washington)
- L3: GoDaddy (IP Address: 208.109.4.218, Arizona)

More landmarks can be used, but we limited the number to three in the evaluation. The scenario is that the VM in the US (Oregon) represents the *original* VM, and the one in Singapore the *migrated* VM. We run the migration detection application in the VM hosted in US region after setting the landmarks (e.g. using IPs), the Ping Counter (P.C), and the Ping Interval. The Ping Counter was set to 10, 30, 50, 70 in each of the the experiments. Each experiment with a specific ping counter was repeated three times. This means that for a specific landmark the latencies were calculated approximately 240 times in total, with the results being averaged. To prevent IP blocking, the Ping Interval was set to 3 seconds. Tables 5.3, 5.4 and 5.5 show the results. Each table shows the result of obtaining ICMP latencies from the original VM (U.S Oregon region) to the landmark shown. The table also shows the allowed range values that result from using both low and high threshold factor values (K). This means that if the new calculated latency falls between the allowed range, there will be no alert for any migration event. The threshold value should be chosen by the

cloud consumer based on their estimation regarding the error of the detection. For example, if it is known (from experience) that an ICMP latency measurement to a specific landmark is fluctuating between 30 ms to 50 ms but it averages to about 40 ms, then K can be set to 0.25. As a proof of concept, in all the experiments, we set K=0.1 to mark the minimum allowed threshold range and K=1 to mark the maximum allowed threshold.

Table 5.3 ICMP Latencies results from the VM to landmark: Google

Landmark: IP Address: 173.194.116.48, Google, California					
	P.C =10	P.C =30	P.C =50	P.C =70	avg-all
1st run (sec)	159.281	159.308	160.4	159.095	
2nd run (sec)	159.263	158.951	160.326	159.698	**159.58**
3rd run (sec)	159.214	158.93	160.343	160.155	
avg-3 runs	159.252	159.063	160.356	159.649	
t-min (k=0.1)	143.327	143.156	144.320	143.684	
t-max (k=0.1)	175.177	174.969	176.391	175.614	
t-min (k=1)	0	0	0	0	
t-max (k=1)	318.505	318.126	320.712	319.298	

Table 5.4 ICMP Latencies results from the VM to landmark: Yahoo

Landmark: IP Address: 206.190.36.45, Yahoo, Washinghton					
	P.C =10	P.C =30	P.C =50	P.C =70	avg-all
1st run (sec)	11.474	10.607	11.85	11.453	
2nd run (sec)	10.782	11.697	10.373	11.357	**11.260**
3rd run (sec)	10.667	11.438	12.165	11.261	
avg-3 runs	10.974	11.247	11.462	11.357	
t-min (k=0.1)	9.8769	10.122	10.316	10.221	
t-max (k=0.1)	12.071	12.372	12.608	12.492	
t-min (k=1)	0	0	0	0	
t-max (k=1)	21.948	22.494	22.925	22.714	

After sending ICMP packets constantly based on different combinations of ping count values, we were able to notice that the latency is consistent most of the times. Hence, we consider the average ICMP latency from the original VM to each chosen landmark. Tables 5.3, 5.4 and 5.5, show the results. The group of average ICMP latencies from the original VM to the landmarks forms the virtual landmark fingerprint or the "VM profile". We use the values in the profile to detect or signal the migration event after comparing them with the values obtained from calculating the ICMP latencies again; but from the destination (migrated VM). Table 5.6 shows the virtual

Table 5.5 ICMP Latencies results from the VM to landmark: GoDaddy

Landmark:,IP Address: 208.109.4.218, GoDaddy, Arizona					
	P.C =10	P.C =30	P.C =50	P.C =70	avg-all
1st run (sec)	44.096	44.181	44.267	44.062	
2nd run (sec)	44.421	44.151	44.139	43.804	**44.171**
3rd run (sec)	44.151	44.199	44.48	44.109	
avg-3 runs	44.22	44.177	44.29	43.991	
t-min (k=0.1)	39.80	39.7593	39.86	39.592	
t-max (k=0.1)	48.64	48.5947	48.72	48.390	
t-min (k=1)	0	0	0	0	
t-max (k=1)	88.44	88.354	88.59	87.983	

fingerprint based on specific landmarks. The VM owner can use this fingerprint as a profile for the original VM (to check periodically). The table also shows what the latency became for the same landmarks when pinging from the VM after migrating to Asia region. As expected, there was a considerable increase in the latencies values. According to our methodology, this notable change in the latencies may indicate that a possible migration of VMs has occurred. Figure 5.9 illustrates the scenario and the obtained latencies from both the source and the destination sites.

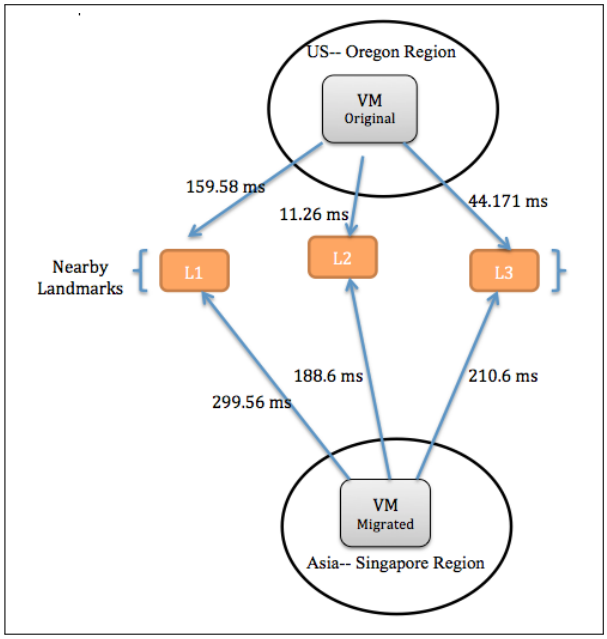

Fig. 5.9 Scenario1: Illustrating ICMP latencies between the VM in the US region to the selected nearby landmarks

Table 5.6 The Virtual Landmark fingerprint (VM profile) based on scenario 1 and the latencies values after migration to the Singapore region

Profile of the VM hosted in US region using the Nearby landmarks				
LANDMARK	**IP-ADDRESS**	**LATENCY (US-Oregon Region)**	**LATENCY (Asia-Singapore Region)**	**Percentage of Change in Latencies**
Google	173.194.116.48	159.6 ms	299.65 ms	87.7 %
Yahoo	206.190.36.45	11.26 ms	188.6 ms	1575%
GoDaddy	208.109.4.218	44.17 ms	210.6 ms	376.8%

Scenario 2: Distant landmarks

In order to see the effect on accuracy when choosing relatively distant landmarks instead of relatively close ones, we chose three more landmarks (L4, L5, L6) located in Europe, as follows:

- L1: Colt.co.uk (IP Address: 62.116.130.8 , Germany)
- L2: Rackspace (IP Address: 212.64.133.165 , London)
- L3: Facebook (IP Address: 31.13.90.2 , Dublin)

Table 5.7 summarises the results obtained after sending ICMP ping requests to these landmarks from the original VM hosted in the US (Oregon), and the results after the migration process is assumed to have commenced. There was also a percentage of change in the latencies obtained after migration. This may also help to detect the migration process. However, if the consumer chooses a high threshold factor for these distant landmarks, there is a chance that the migration event detection will be missed (false negative). Table 5.8 shows the status of migration detection in cases where the threshold factor was low (0.1) or high (1.0). From the table, there is a high chance that the migration will be missed when using distant landmarks and a high threshold factor. This may indicate that if very close internet landmarks were chosen, migration would have been detected more accurately. Thus, in approach 5.2, we chose co-located VMs in the public IaaS cloud to act as landmarks. Besides, in order to avoid the dependency on the latency metric alone which can sometimes lead to inaccuracies, we combine several other metrics such as: detecting hypervisor type/version change, external IP change, and hardware change.

We further chose two landmarks (Google & Yahoo) to measure the variability of the latency data from the Oregon VM. ICMP ping to these landmarks from the

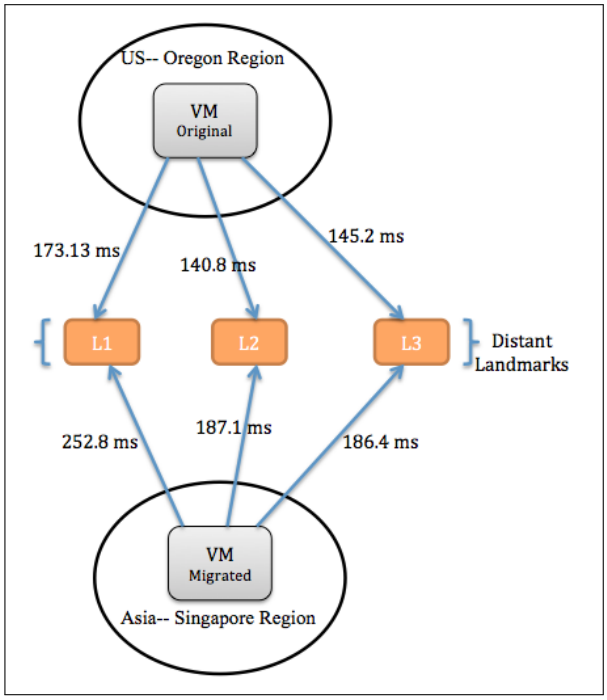

Fig. 5.10 Scenario2: Illustrating ICMP latencies between the VM in the US region to the selected distant landmarks

Table 5.7 The Virtual Landmark fingerprint (VM profile) based on scenario 2 and the latencies values after migration to the Singapore region

Profile of the VM hosted in US region using the distant landmarks				
LANDMARK	**IP-ADDRESS**	**LATENCY (US-Oregon Region)**	**LATENCY (Asia-Singapore Region)**	**Percentage of Change in Latencies**
Colt	62.116.130.8	173.13 ms	252.8 ms	46 %
Rackspace	212.64.133.165	140.8 ms	187.1 ms	32.8 %
Facebook	31.13.90.2	145.2 ms	186.4	28.37%

Table 5.8 Checking whether migration can be detected or missed, based on the selected threshold factor

TRUE: detected, FALSE: missed			
Detection with Low Threshold Range (K=0.1)			
	L1	L2	L3
Nearby Landmarks	TRUE	TRUE	TRUE
Distant Landmarks	TRUE	TRUE	TRUE
Detection with HIGH Threshold Range (K=1)			
	L1	L2	L3
Nearby landmarks	FALSE	TRUE	TRUE
Distant landmarks	FALSE	FALSE	FALSE

original VM was executed every 30 seconds for a duration of approximately 18 hours. The *Google* landmark yielded a variance of 0.284 and a standard deviation of 0.533. However, the *Yahoo* landmark yielded a variance of 14.36 and a standard deviation of 3.78, indicating much greater variability. This also means that we could have made a better choice by selecting an alternative landmark which provided better latency results, such as the Google landmark.

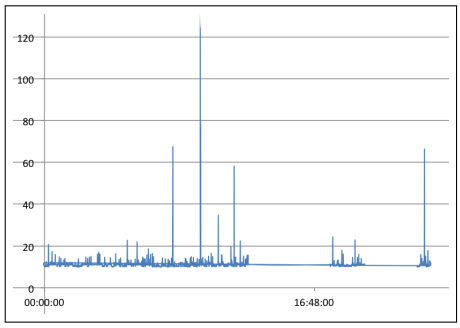

Fig. 5.11 Yahoo landmark's data variability

5.9.8 Approach 1: discussion & conclusion

The landmark-based virtual fingerprint technique was used to detect a migration event based on the choice and interaction with a number of selected "landmarks". The experiments demonstrate that multiple ICMP ping requests to chosen landmarks yielded consistent latency results, and could be included in a VM fingerprint/profile (we made use of both nearby and distant landmarks). We argue that this technique cannot *guarantee* the occurrence of a migration event, but provides a useful estimate

that a migration is likely to have taken place. Latencies are affected by many factors including network restrictions, background load, hardware, etc. We therefore believe that combining multiple detection techniques could provide a more reliable outcome, as discussed in Section 5.8. In practice, the cloud provider may restrict ICMP pings from the VM or to the VM to prevent ping flood attacks. Hence, other techniques can be used instead. In a typical cloud scenario where the consumer wants to apply our suggested hybrid migration detection approach, proper tools can be deployed in the VM in a unified framework. Each detection technique is weighted based on its importance/accuracy and combined in decision function to control when the alarm is raised. Using a hybrid approach can increase awareness of the occurrence of a migration event. It could also incur some overhead in terms of performance and storage. However, it is up to the consumer to decide the number of techniques used and the frequency of execution. Section 5.10 discusses using the hybrid approach.

5.10 Approach 2: Combined Approach

5.10.1 Overview

In Section 5.9 we highlighted nine techniques that could be made use of to detect or infer migration or provide a probabilistic estimate that VM migration has occurred. We evaluated a technique that is based on measuring the latency to specific "virtual" network landmarks, and we found that we cannot rely on one technique alone. We therefore proposed a decision function that combines several (measurable) metrics to detect migration. In this section we extend this decision function (by considering ICMP ping interactions between the *critical* VM and some *light-weight* monitoring VMs) and demonstrate how it can be made use of in practice.

5.10.2 Problem specification

We focus on the Infrastructure-as-a-Service (IaaS) cloud delivery model where a consumer deploys a cluster of VMs (virtual cluster) on a public cloud, hosting a variety of critical services such as: web-server, database server, mail server, etc. It is important to emphasise that VM clusters are of particular interest, as we are able to also investigate interaction between the VMs. Virtual clusters are supported by most cloud providers where the VMs are logically connected by a virtual network across several physical networks. This allows the VMs to communicate with each other freely through the virtual NIC and configure the network automatically [71].

As mentioned previously, we refer to these virtual servers that hold importance to the customer as *Critical* VMs; any VM that is sensitive to the customer due to the tasks it performs or the data it processes or generates.

Migration of such VMs to an unknown location, may lead to the loss of critical information; due to the different laws in different regions which govern data access and protection. For example, the Patriot Act in the U.S.A gives the U.S. government the right to access any hosted data. Data encryption has limited benefit in such a scenario, as decryption would be needed at some point to process the information (unless a fully homomorphic encryption mechanism is in place [55]). Cloud providers usually have different zones and regions to host the data of consumers. If the zones are within the same region, migration within the zone may not be an issue. However, if the VM and its data was migrated to another region other than the one agreed about,

issues may arise. It is not just about the privacy of the data. As mentioned before, migration of VMs without notifying the consumer, may breach SLA terms; if the SLA states the region at which data must reside. Hence, a mechanism to detect or signal the migration of VMs is needed. If the SLA terms is violated, provided an evidence that migration event has occurred, consumers could have stronger grounds for their claim. The introduction of services to migrate VM workloads across different cloud providers has also emphasised on the need to detect the migration. For example, it is now possible to use Cloud Endure[3] services to migrate VMs across different cloud providers without service downtime. According to their website, they are able to migrate VMs lively from their original working environment to different public clouds such as: Amazon Web Services (AWS) or Google Cloud Platform (GCP). Although the availability of such services could be useful for cloud consumers, it may increase the risk of migrating workloads to another region without notifying the consumer (especially in a federated cloud context, where multiple cloud providers may collaborate with each other to provide services to the consumer).

In Section 5.9 we evaluated the *virtual landmark fingerprinting* migration detection technique, where we measured the ICMP latency from a given VM to several selected reliable known servers, e.g. Google or Yahoo, in order to create a VM profile/fingerprint. We established that latency to these landmarks changes considerably if the VM is migrated to another region, which may give a good estimate that VM migration has occurred. However, due to various factors related to the variability in latency likely to be observed within a publicly managed network, we argued that relying on the ICMP latency metric alone may not be appropriate and may result in an unreliable conclusion. For this, we suggested the aggregation of several metrics in a decision function instead of just one, to provide a probabilistic assessment of a potential migration event. In this work, we again use the *latency metric*, but in this case to measure the latency between the critical VM and selected light-weight VMs in the same cluster. Light-weight VMs are small and low cost VMs whose sole purpose is to monitor or track the critical VM.

We believe that this has several benefits over the reliance on general internet landmarks –some of these benefits include: (1) avoiding IP blocking due to excessive ICMP ping operation, as VMs in the same cluster can communicate freely; (2) deducing what happens to VMs from other VMs, i.e. not having to rely on a single point of alerting; (3) latency between VMs in the virtual cluster in a specific region is usually very small,

[3]https://www.cloudendure.com/

which makes comparing it with other latencies in distant regions easier.

Along with the latency, we consider three other metrics: *change in Public IP*, *change in hardware*, and *change in the hypervisor or its version*. It is important to note that consumers could have clusters of VMs across multiple cloud providers. This work focuses on VMs being migrated within data centres owned by a single provider. Consideration of VM migration across multiple providers is outside the scope of this work, primarily due to the difference in measured metrics (and their possible semantics) that could be adopted by different providers. However, due to the availability of solutions which facilitate the migration of VMs across multiple cloud providers with minimal service shutdown (as mentioned before), it appears that there is a need to consider this possibility. Figure 5.12 illustrates how VMs may be clustered across multiple cloud providers whose architecture spans several regions. The VMs included within a cluster could be physically hosted across several physical servers if the cloud customer requests. The figure also shows the critical VM and its interactions with various monitoring VMs that co-exist in the cluster.

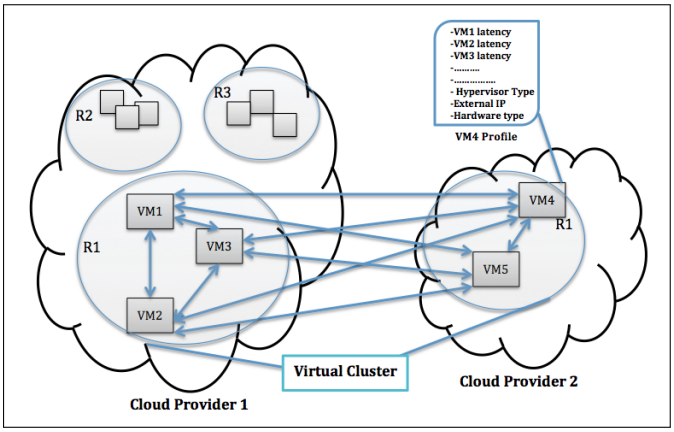

Fig. 5.12 High level cluster of VMs in the Cloud

From the previous points we hypothesise that monitoring certain metrics from within the VM itself and the monitoring VMs, and combining this using a weighted decision function, could lead to better detection of migration events. Hence, if a change in one of the metrics occurs, a flag will be set to identify potential migration. The decision function is the key proposal in our migration detection system and a key outcome for the consumer. For example, a change in the public IP address may indicate a migration event but from the perspective of the consumer this may not be as significant as the change in the hypervisor type. Section 5.10.3 covers the metrics

we make use of and the decision function in greater detail.

In the context of public cloud systems, migration of the VMs in public IaaS clouds could happen: (1) within the same zone- from one host to another; (2) from one zone to another in the same region; (3) from a region to another zone in a distant region. To contextualise this, we highlight some general points related to how some cloud providers manage their infrastructure. For example: (i) regardless of the location of the VMs, most cloud providers tend to keep the private IP address of the VMs the same, but the external/public IP addresses could change each time the zone or region changes; (ii) cloud providers could have *different* hardware components in every zone and region – some cloud providers, e.g. Google, explicitly mention processor related information in each region/zone. (iii) it is possible that cloud providers have different types of hypervisors running on physical hosts. Thus migrated VMs could be under the control of different hypervisors and this could happen across zones or regions.

The discussion above also brings some requirements for the migration detection system within a cluster of VMs:

- **Automated:** The consumer can instruct the system to start the monitoring/detection process by inputting IPs of the VMs in the cluster and set-up configurations.

- **Performance-aware:** This can allow a VM owner to decide the level of granularity of the monitoring process so as to not affect its performance. This is a key requirement – i.e. internal-VM monitoring should not significantly impact the application that is hosted by the VM.

- **Modular:** This allows the addition of more metrics or indicators to the decision function and to set their importance from the perspective of the consumer.

- **Informative:** The VM owner may only be interested in successful migration detection events or the status of a metric at a particular point in time, however the consumer may not be interested in all the captured data used to arrive at the decision.

- **Economic:** The monitoring/detection of migration events within a cluster of VMs should not consume resources from the VMs unnecessarily (this also relates to the performance-aware requirement identified above).

- **User-driven:** techniques used to deduce the migration decision should be based on data that can be acquired by a consumer who owns the VM, and not require (as much as possible) input from the cloud provider.

We consider all these requirements as part of the proposed work. It is imperative, however, to set out the scope and some assumptions of this research before we proceed. Migrating VMs *within the zone* is unlikely to create any privacy issues or lead to breaches in the terms of the SLA (so this is considered out of scope). We are going to assume that migrating from one zone to another in the same region also violates the terms of the SLA. Public IP addresses usually change automatically when migration occurs, unless the consumer owns a static public IP which will be assigned to the VMs. In this work, we assume that the consumer does not use static external IP addresses for the VMs. In IaaS public clouds, if the VM is cloned or snapshotted by unauthorized parties, this could also raise security and privacy concerns. However, this is beyond the scope of this research (unless the clone or snapshot event can be detected by the VM).

5.10.3 Detecting the migration of critical VMs

We focus on whether migration of a *critical* VMs can be detected in public cloud systems. To achieve this, we suggest the use of inexpensive light-weight VMs (which do not consume a lot of computational resources) to monitor and track the critical VMs. Such light-weight VMs can be owned by the same cloud provider, and may be launched and hosted across different data centres owned by the provider. The intra-*cluster* VM interaction is used to detect a migration event. The research methodology is comprised of four stages: (1) generating initial VM profiles (referred to as "profiling" in subsequent sections); (2) periodic checks for measurable metrics used in migration detection; (3) determining how the results of these metrics should be weighted; (4) reporting the findings to the VM owner. Each of these aspects is described below in further detail.

Generating and updating VM profiles

Information about the current state of all VMs is gathered, based on the measured metrics, and we refer to this as the *normal profile*. This profile becomes a basis for comparison to decide if migration has occurred. The normal profile includes several variables such as:

1. *Latency to all other VMs:* The average ICMP latencies between VMs are determined, varying the number of ICMP ping messages between VMs (based on storage and performance overhead limitations). More ICMP requests can generate high overheads – and may also be limited by the provider hosting the VM.

2. *Hypervisor Type:* as VMs can be managed by a variety of hypervisors, a change in location may lead to a change in hypervisor type.

3. *Processor Information:* VMs are placed on physical hosts which can differ in their capability, model, features, etc. It is therefore essential to detect any change to the hardware hosting the VMs. We record processor information which can be acquired from the VM itself, such as vendor-id, CPU family, model, model name, CPU MHz rating, cache size, and CPU cores. We assume that processor information is accurate and has not been manipulated by the hypervisor.

4. *External & Public IP:* In most cloud providers, the private IP address remains the same for ease of configuration, but the public IP changes when the VM is migrated.

Periodic check

A periodic check of the metrics for each VM covered in 5.10.3 is performed. Metrics are expected to stay the same or within the accepted range if nothing changes for the VM. If there is a change in the metrics, a new profile with the time stamp will be recorded. The frequency at which a periodic check is performed is determined by the VM owner – a high frequency check of the metrics may affect the performance of the application hosted within the VM. A periodic profile is generated after every check and uses the same recording format as the normal profile. Comparison with the normal profile can be used to determine whether a potential migration event has taken place.

Decision mechanism

Profiles of the VMs at different time stamps are checked against the previous normal profiles in order to determine the likelihood of migration. The decision is based on four flags (all initialised to zero):

- **Latency flag (LAflag):** This flag uses minimum and maximum latency thresholds to determine when an obtained averaged latency can be classified as an anomaly. If the average latency in the normal profile falls below or above the minimum latency in the normal profile, then the latency is said to have *probably* changed and the flag will be set to 1. The newly calculated average will be the result of multiple ICMP pings within specified intervals. ICMP ping data do not have to be stored in the VM but they can be stored temporarily before being sent to the VM owner for further analysis (Section 5.10.9). In the evaluation, we assume that ICMP ping data is stored in the VM just to measure the storage overhead.
- **Changed-Hypervisor flag (CHflag):** If the yielded hypervisor type or the hypervisor version differ compared to the one in normal profile, then the flag is set to 1.
- **Changed Processor Information flag (CPflag):** the processor information contains several important variables related to the hardware of the physical host. Not all properties associated with this variable may change after migration. We set the flag to 1 if there is any detected change. However, a percentage change (i.e. some properties -of all measured- have changed) can also be considered to allow some flexibility.
- **Changed External IP flag (CEflag):** There are a variety of ways to obtain the public or external IP address of the VM from within the VM itself. If the IP is changed, the flag will be set to 1.

The decision function $d(m)$ (based on the four metrics above) is used to control when the alarm is raised, and can be expressed as follows:

$$d(m) = \frac{1}{\sum_{\forall i} a_i}(a_1(LAflag) + a_2(CHflag) + a_3(CPflag) + a_4(CEflag)), (0 <= a_i <= 1) \tag{5.2}$$

Each a_i represents the importance of a detection technique from a consumer's perspective – often based on the accuracy with which this metric can be measured. We assume each flag (e.g. CEflag) results in a binary outcome (i.e. whether it has changed or not). The result $d(m)$ is a weighted combination of various migration detection techniques /metrics and describes the likelihood of a migration having taken place.

Each a_i will be decided on a scale from 0 to 1 with 1 being the most important from the consumer's perspective and 0 having the least importance or impact. It is expected after recording the normal profile that the function will yield a zero result if there was no change in the newly generated profiles. As mentioned before in section 5.8, we do not make recommendations as to how the importance of the migration metrics could be set. In the experiments in section 5.10.6, the values of a_i were just examples to execute the experimental scenario and not recommendations.

Reporting

Based on the outcome of the decision function, the VM owner needs to be notified. The profiles generated during the monitoring process may need to be sent to the owner for further analysis. The newly generated profiles can be sent as soon as they are ready, or they can be sent in batch (to avoid the network overhead of opening multiple network connections). In contrast, sending the profile as soon as it is ready will help identify a potential migration event earlier, but this is up to the owner to decide. The migration detection system is part of our VMInformant system. In Chapter 6 more details about the types of supported user reporting can be found.

5.10.4 Algorithm design

Algorithm (1) provides a high-level look into the logic behind generating the normal profiles and the decision making processes to be embedded in the VMs in the cluster. The algorithm we use spans three phases: initialisation, generating the normal/periodic profile, and migration detection. In the initialization phase, all variables will be set, including: IP addresses of the VMs in the cluster, ping interval, ping count, threshold factor, periodic check interval, metrics importance (defined by the consumer). The *periodic check interval* is to decide the period after which periodic profiles will be generated in the VM again. Options related to the normal and periodic profiles are managed and are based on three methods:

1. Method of storing: Whether to store all the generated periodic profiles or store only the recent one (with timestamp). This is useful to reduce the storage overhead.

2. Method of reporting: Whether to send the profiles all at once (in bulk), or send them individually as soon as they are generated.

3. Method of profiling: Whether to perform an intensive latency check or a light-weight check. For example, in the intensive check the ping counter can be set to 500. Depending on the ping interval, generating the profile can take considerable time, but the accuracy of calculating the average latency will be better. Choosing a light-weight method for profiling can speed up the process of generating the profile at the cost of accuracy.

The process of generating the profiles will involve calculating the average ICMP latencies from the VM to the other VMs in the cluster. Also, it involves setting the values of the other metrics. In our implementation we record the current external/public IP address of the VM, hypervisor type, and processor information (to check hardware changes). The normal profile will be created only once. A flag will be set indicating that the normal profile has already been generated. If the flag is set to true, periodic profiles will be generated instead, based on the selected methods of profiling, storing and reporting. In the migration detection phase, a decision has to be taken as to whether the newly generated periodic profile differs from the original normal profile. If it does, appropriate metrics flags will be set and a decision outcome will be sent to the consumer. The decision is based on the values obtained in the flags and also on the importance of metrics from the perspective of the consumer. The approach used within the decision function is provided in Algorithm 2. The data collected in the VMs can either be discarded after the decision and profiling is re-initiated, or may be kept for further analysis.

5.10.5 Some background about real public IaaS cloud systems

To the best of our knowledge, Google is the only IaaS cloud provider which has implemented the live migration feature in their platform in order to achieve transparent maintenance. This feature is enabled by default, and consumers may not be significantly affected. In their website, it is mentioned that during the live migration guest OS performance will be impacted to some degree in terms of performance, but the VM

Algorithm 1 Profiling logic embedded in the VMs

```
1: procedure GetSetParameters( )                    ▷ Initializing some parameters globally
2:     get VM IP address for ICMP,
3:     get ping interval, ping count, ThresholdFactor,timeToSend
4:     get MetricsImportance[], periodic check interval
5:     get method of storing, method of reporting
6:     define MinThresholdLatency[],MaxThresholdLatency[],Latency[]
7:     define CLflag,CHflag,CPflag,CEflag
8:     NormalProfileGenerated = false ;              ▷ Boolean stored globally
9:     LatenciesCount = NoVMs − 1
10: end procedure
11: procedure GenerateTheVMprofile( )               ▷ Collecting the metrics
12:     ICMP Ping related VMs;                      ▷ Based on collection method
13:     Calculate average latency to each VM;       ▷ latency[]
14:     Get hypervisor type;
15:     Get processor info;
16:     Get external IPs;
17:     Generate the profile                        ▷ (label: normal or periodic + timestamp)
18: end procedure                                   ▷ Based on method of profiling:
19: if NormalProfileGenerated == False then         ▷ Creating profiles for first time
20:     GenerateTheVMprofile()                      ▷ (label: normal/periodic + timestamp)
21:     for i=1 to LatenciesCount do
22:         MinThresholdLatency(i) ← latency(i) − latency(i) ∗ ThresholdFactor;
23:         MaxThresholdLatency(i) ← latency(i) + latency(i) ∗ ThresholdFactor;
24:         ++i;
25:     end for
26:     NormalProfileGenerated ← True;
27: end if
28: while True do                                   ▷ Run continuously
29:     wait(PeriodicCheckInterval)
30:     GenerateTheVMprofile()
31:     if storingmethod == onlyIfChanged then
32:         DecisionFunction()                      ▷ Decision Function comes next
33:         if DecisionFunction() > 0 then
34:             send alarm and new profile to owner;
35:         end if
36:     else
37:         store new generated profiles;
38:         if TimeToSend then
39:             send all the profiles in bulk;
40:         end if
41:     end if
42: end while
```

Algorithm 2 Decision function logic embedded in the VMs

```
1: function DecisionFunction()
2:     for i=1 to LatenciesCount do                          ▷ Deciding the CLflag
3:         if (latency(i) < MinThresholdLatency(i) or latency(i) > MaxThresholdLatency(i)) then
4:             CLflag ← 1
5:         end if
6:         ++i;
7:     end for
                                                             ▷ Deciding the CHflag
8:     if periodicProfile.hypervisorType ≠normalProfile.hypervisorType then
9:         CHflag ← 1
10:    end if
                                                             ▷ Deciding the CPflag
11:    CPflag=getPercentageOfChange(periodicProfile.processorInfo
       ,normalProfile.processorInfo);
                                                             ▷ Deciding the CEflag
12:    if periodicProfile.ExternalIP ≠normalProfile.ExternalIP then
13:        CEflag ← 1
14:    end if                    ▷ Apply decision function by using weighted metrics
15:    for i=1 to flagsCount do                  ▷ flag(i) denote the list of calculated flags
16:        DecisionIndex = DecisionIndex + flag(i) ∗ MetricsImportance(i);
17:        ++i;
18:    end for
19:    DecisionIndex = DecisionIndex/flagsCount;
                          ▷ Optional: send alert when latency and other metrics change
20:    if (CLflag == 1 and at least one other flag == 1) then
21:        DecisionCategory ← 'strong'
22:    end if
23:    return DecisionIndex, DecisionCategory
24: end function
```

instance remains online throughout the migration process [57].

If the transparent maintenance feature is not enabled by the consumer, maintenance will be scheduled where consumers will be informed beforehand so they can switch off their machines. The next time it starts, it will start in another machine. This is currently the case for AWS consumers. Google also allows the migration of VMs from one zone/region to another. This, however, may incur some minor downtime as VMs are copied across the network and started again. The Google Compute Engine offers several types of data disks that can be used as primary storage for virtual machine instances: Persistent disk storage, and Local SSD storage. Persistent disks can be attached and detached to any instance, while Local SSD disks can be physically attached to a server that is running the instance. Hence, data stored in persistent disk storage remains intact regardless of the state of the instance to which it is attached, while in the local SSD the data does not persist beyond the lifetime of the instance. To allow the migration of the VM along with its data, persistent disk storage which is attached to the instances is considered.

The Google Compute Platform has four regions and thirteen zones within these regions. To manage and control the execution of virtual machines in this infrastructure, Google uses the KVM hypervisor. Each region has zones which may use different hardware. For example, the Western Europe region has three zones, and each zone has different processor types –i.e. different CPU micro architectures from Intel. Details of the processor types are published online.

This also means that the VM may have been migrated to another distant region which has the *same* processor information. That is why in this research we use a hybrid approach that not only considers clues about the change of processor information but also other important metrics.

In the Google Compute Platform, moving or migrating the VMs from one zone/region to another is performed using the *gcloud*[4] tool, which provides the main command-line interface for Cloud Platform products and services. It is part of the general Google Cloud SDK which includes a set of tools that you can use to manage resources and applications hosted on Google Cloud Platform.

[4]https://cloud.google.com/sdk/gcloud/

5.10.6 Implementation & scenarios

In this section we provide an overview of how the experiments were carried out to evaluate the decision function. We also describe the implementation of algorithms 5.10.3 and 5.10.4.

Experimental setup & design

As a proof of concept of applying the migration detection of VMs in public IaaS cloud, we used the Google Cloud Platform[5]. We picked one zone in Europe as default (europe-west1-b), and created the critical VM and lightweight monitoring VMs initially in this region (two monitoring VMs). The platform allows triggering of the migration from one zone to another in the same region, or from one region to another – so we will use this to check if the metrics change after the migration takes place by comparing the profiles of the VM. The check is based on the following scenarios:

- From one zone to another in the same region: migration to europe-west1-c
- From one region to a distant region: migration to: us-central1.b, us-central1.a and asia.east1-a.

In order to make use of the decision function, we assume that the importance of all the metrics is decided by the VM owner/ cloud consumer. For this work, as a proof of concept, we assumed that CL Importance was set to **0.6**, CE Importance set to **0.8**, CP Importance set to **0.4** and CH Importance set to **0.6**. The parameter values are just examples. As mentioned before, this research does not intend to make recommendations as to how the migration parameters should be set.

Implementation

In each VM, we used a monitoring script for generating the normal and the periodic profiles. This script collects the required metrics using the parameters set by the consumer, e.g. Ping count. It also applies the decision function in order to generate the decision index which can be used to raise alerts. It runs in the background and can start automatically in the case of shut-down or restart, and continue to communicate with other VMs; provided that it is not behind a firewall. To retrieve the hypervisor type information from the VM, a tool called *virt-what*[6] was used.

[5]Google Cloud Plaform https://cloud.google.com
[6]Virt-what : https://people.redhat.com/ rjones/virt-what/

5.10.7 Results

Normal VM Profiles

Table 5.9 shows the generated normal profile of the critical VM after collecting the metrics. The normal profiles of the monitoring VMs were also generated. From the normal profiles, it was noted that all the calculated average latencies were very similar and very low because they were all located in one region and one zone. All of the VMs have the same processor info as in Figure 5.13. These normal profiles are considered virtual fingerprints of the VMs and can later be used to compare against changed profiles.

Table 5.9 Critical VM normal Profile

VM1 Latency	1.236
VM2 Latency	1.235
Hypervisor Type	kvm
Public IP Address	104.155.86.33
Processor info	see Figure 5.13

```
vendor_id       : GenuineIntel
cpu family      : 6
model           : 45
model name      : Intel(R) Xeon(R) CPU @ 2.60GHz
cpu MHz         : 2600.000
cache size      : 20480 KB
cpu cores       : 1
address sizes   : 46 bits physical, 48 bits virtual
```

Fig. 5.13 processor information collected from VMs running in europe-west1-b

Migrating the Critical VM to another zone in the same region

Table 5.10 shows the periodic profile of the critical VM after initiating the migration to another zone in the same region. Latency was still within the range, but the public IP and process information changed. From the profile it is obvious that the hardware in this zone may differ from the hardware of the default zone. Table 5.11 shows the results of applying the decision function based on the changes in the measured metrics. The resulted decision index shows that a possible migration has occurred, but since latency was not significantly affected, the migration may have been within the same region. From the perspective of some owners, it may be acceptable for critical VMs to be migrated within the same region as long as it does not violate the SLA, but for others it may not. That is why it is up to the owners of the critical VMs to carefully decide the importance of metrics.

Migrating the Critical VM to distant regions

Table 5.10 Critical VM Periodic Profile after migration to europe-west-c

VM1 Latency	1.011
VM3 Latency	1.23
Hypervisor Type	kvm
Public IP Address	104.155.103.207
Processor Info	see Figure 5.14

```
europe-west1-c
vendor_id       : GenuineIntel
cpu family      : 6
model           : 62
model name      : Intel(R) Xeon(R) CPU @ 2.50GHz
cpu MHz         : 2500.000
cache size      : 30720 KB
cpu cores       : 1
address sizes   : 46 bits physical, 48 bits virtual
```

Fig. 5.14 Processor information from a VM running in europe-west1-c

Table 5.12 shows the results of applying the decision function in the critical VM when migrating within the Google Cloud Platform. As can be seen from the table, some metrics always change, such as latency and the public IP address. Processor information sometimes changes, but hypervisor details remain the same – as Google mainly uses KVM. The decision function shows a higher decision index, which may indicate that a potential migration to *another* regional site has occurred. However, this index is based on a variable set by the consumer to denote the importance of the metrics. In that sense, it is up to the consumer how the decision index can be used.

Table 5.13 highlights all possible combinations of the cases where the metrics can change according to the scenario above. It shows the inference that can be made based on changes in the migration metrics. It also identifies the outcome of the decision function based on the importance of the metrics (set by the consumer) – we use (CL)= 0.6; (CE)= 0.8; (CP)= 0.4; (CH)= 0.6, as in the aforementioned experimental scenario. In the table, if a metric is not changed it is preceded by the (!) symbol, e.g. (!CL) indicates that the latency was not affected.

5.10.8 Evaluation

We provide an evaluation of the proposed decision function in terms of performance and storage overheads, since we do not want critical VM operations to be affected. The normal profile is generated for the critical VM and the monitoring VMs. To evaluate the performance, we use the Povray benchmark application within the critical VM. As this is a computationally intensive benchmark, insertion of monitoring events will have an impact on the rendering time. We render a Povray file (chess2.pov), but use different resolutions each time; first without migration detection, and then with the

Table 5.11 Applying the decision function on VM 2 according to the new periodic profile after the migration to europe-west1-c

	FLAGS	**Importance (0 to 1)**	**Total**
Latency Change	0	0.6	0
IP Change	1	0.8	0.8
Processor Info Change?	1	0.4	0.4
Hypervisor change	0	0.6	0
(Decision Index)			**0.3**

Table 5.12 Decision function result of the critical VM when migrating to distant regions

	Latency Change	**IP Change**	**Proc-Info Change**	**Hypervisor Change**	**Decision Index**
us-central1-b	yes	yes	yes	no	0.45
us-cenral-a	yes	yes	no	no	0.35
asia-east1-a	yes	yes	yes	no	0.45

detection in place for a different number of monitoring VMs. Table 5.14 shows the result of running the Povray file benchmark. From the experiments we find that the detection mechanism does not significantly affect the performance of the rendering process. The periodic check performance is also not significantly affected by an increase in the number of monitoring VMs. It is mostly affected by the ping count and the ping interval. Total periodic-check time (in seconds) in a day can be approximated as:

$$d(m) = pingCount * pingInterval * (24/periodicCheckInterval(hr)) \tag{5.3}$$

Storage is proportional to the number of VMs and it is a factor of the ping interval [i] and the ping count [c] – we use i=1 and c=500. Also, it is affected by the interval after which the periodic check is repeated. This means that if the periodic check interval=30 mins (0.5 hr), according to our experiment using three VMs, storage will consume about 4320 KB for 24 hours. If the number of VMs is increased to seven, storage will be approx. 13300 KB. Given that the analysis data will be reported to the owner at some point, and the recording of all latencies may not actually be needed in practice, storage may not be a significant issue. In the worst case, where the number of VMs=100, storage will be approximately 213 MB. The flexibility in the system also allows reduction of the ping count or increase of the ping interval. We conclude that our migration detection system does not significantly impact the routine processes running in the VMs, and little storage is consumed in normal cases. If we assume the storage for ping count [c]= 1 is 0.09 kb (according to our calculations), the total storage in 24 hours will be:

Table 5.13 Possible combination of the changed metrics and the inference that can be made

Importance of metrics: (CL)= 0.6; (CE)= 0.8; (CP)= 0.4; (CH)= 0.6;					
Cases				*Inference*	*df*
CL	CE	CP	CH	Possible migration to another region	1
!CL	CE	CP	CH	Possible migration within data centre or to another zone and IP address changed	0.45
CL	!CE	CP	CH	Possible Migration to another region but IP address was kept	0.4
!CL	!CE	CP	CH	Possible migration within data centre or to another zone in the same region	0.25
CL	CE	!CP	CH	Possible migration to another region but same CPU specs	0.5
!CL	CE	!CP	CH	Possible migration within data centre or to another zone in the same region while keeping same IP	0.35
CL	!CE	!CP	CH	Possible migration to another region while keeping IP address and same CPU specs	0.3
!CL	!CE	!CP	CH	Possible migration within data centre (because IP was not changed)	0.15
CL	CE	CP	!CH	Possible migration to another region	0.45
!CL	CE	CP	!CH	Possible migration within data centre or to another zone same region	0.3
CL	!CE	CP	!CH	Possible migration to another region but IP was not changed neither has the hypervisor type	0.25
!CL	!CE	CP	!CH	Possible migration to another zone (since each zone has same CPU spec)	0.1
CL	CE	!CP	!CH	Possible migration to another region	0.35
!CL	CE	!CP	!CH	Possible migration within the same data centre or just IP was changed	0.2
CL	!CE	!CP	!CH	Possible migration to another region while none of the other metrics have changed.	0.15
!CL	!CE	!CP	!CH	No migration from data centre at all or (maybe inside and more metrics are needed)	0

Table 5.14 Using povray to evaluate the performance and storage overhead

Povray File/ Options	**Execution without Detection**	**Execution with Detection c=500,i=1**	**Exec Time Diff**	**p. overhead %**	**Storage (KB)**
chess2.pov W640 H360 (3 VMs)	187 s	188 s	1	0.53%	90
chess2.pov W640 H640 (3 VMs)	400 s	402 s	2	0.50%	90
chess2.pov W640 H640 (7 VMs)	400 s	404 s	4	1.00%	270

$$d(m) = 0.09 * pingCount * (24/periodicCheckInterval(hr) * (NoOfVMs - 1) \tag{5.4}$$

5.10.9 Approach 2: discussion & conclusion

We have established that the migration detection of critical VMs is important for a number of reasons. We argued that in order to support the trust between the cloud provider and the consumer, a user-driven approach is needed whereby the consumer will be informed of VM migrations that occur in a cluster of VMs. The use of a cluster around a critical VM enables interaction between VMs to help locate a VM of interest. Other than the critical VM, the other VMs in the cluster do not need to have high specifications (as they are primarily used to act as pivots to locate the critical VM).

We highlighted a hybrid migration detection system which aggregates several metrics in a weighted function based on the importance of each metric from the perspective of the VM owner. The output of the function is a decision index that VM owners can use to determine what to do when a migration event is detected. Thus, the system *informs* the owner/administrator about potential migration operations but does not take action when that happens. It is part of a greater system called "VMInformant", which informs the consumer of various security-related events that occur to the VMwithout requiring input from the cloud provider (Chapter 4). The system is flexible in that it allows the VM owner to decide the methods of storing, profiling and reporting in order to reduce the performance and storage overhead. Our system can give an estimate that a likely migration of the critical VM has occurred, however, a number of research issues remain:

- What if the VM has been migrated but was behind a firewall or de-scheduled by the cloud provider? In that case, that particular VM will not be contactable. However, we argue that since the VM is being monitored by other VMs, the consumer will know about this and can be alerted.

- If the latency check resulted in a change for all monitored VMs, how do we know whether the critical VM itself has been migrated or the migration happened to one or all the other monitoring VMs in the cluster? We argue that if the external IP of the critical VM stayed the same while the latencies changed beyond the threshold, it is likely that one or more monitoring VMs have migrated.

- In our approach, we assume that we can use tools which give some information about the hardware in order to detect hardware changes. Since the cloud provider has full control, can we guarantee that the provided information is necessarily correct?

- Since there is a cost associated with the use of light-weight monitoring VMs, how many additional VMs are enough? This is a parameter that currently can be determined based on historical data by a VM owner. We argue that at least two light-weight VMs could give good monitoring support.

To overcome some of these issues (especially the latter), we argue that the decision function of our proposed system can be either computed within the critical VM itself or from another VM which acts as a supervisory virtual machine. The role of this supervisor machine could be (1) collecting/managing profiles (normal/periodic) from all critical and monitoring VMs; (2) determining if abnormalities in the captured metrics are encountered; (3) configuration of the learning/periodic check/reporting process. This supervisory VM can be the basis for a service that can be used by other cloud consumers. Figure 5.15 shows the proposed supervisory VM which collects data from the machine for analysis. Chapter 6 highlights the role of the supervisory VM in detail.

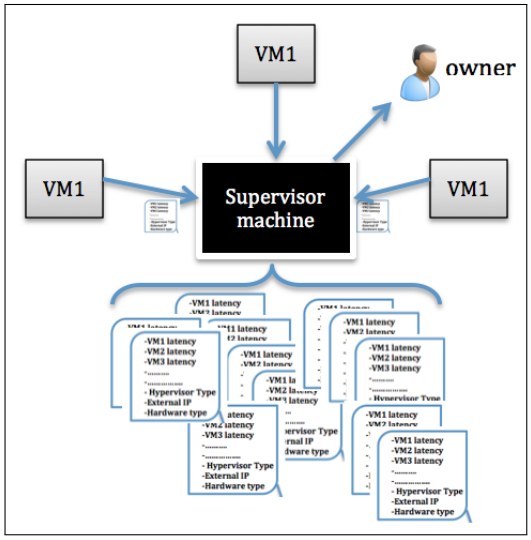

Fig. 5.15 Suggested supervisory VM

5.11 Discussion: Detecting Migration of Containers

Given the two methods of deploying containers in the public IaaS cloud (discussed briefly in in Section 4.14), a question can asked as to whether it is possible to live migrate a container from one host to another. Also, the feasibility of using VM migration detection approaches to detect the migration of containers is discussed in this chapter. In general, there is limited literature on live container migration. This is mainly due to the possibility of creating new containers in new machines faster; instead

of wasting the time to migrate running ones. CRIU[7] is a project that implements a checkpoint/restore functionality for Linux. P.haul[8] is the project on top of CRIU that implements live- migration. According to the manual, the migration requires shared storage for the source and destination hosts. Unlike VMs, which could have persistent virtual disks attached to them (see Section 2.3.6), containers do not seem to support this. This means that it may not be possible to migrate containers and data from one region to another unless there is shared storage between both regions. In the case where containers are deployed within VMs, when VMs migrate, containers will migrate with them. Thus, live migration mechanisms used to migrate VMs will apply.

In this chapter we identified several techniques which can help detect migration of VMs as well as hypervisor change, hardware change, IP change, and latency change. When it comes to containers, since no hypervisor is involved (unless containers run in VMs), detecting the change in hypervisor type may not be applicable. Containers may have a separate IP address, but some, particularly application containers, do not. Hence, the public IP metric might sometimes be applicable, but not always. Moreover, it was not clear how cloud providers deal with the external IP assigned with containers (whether they change it after migration occurs or not). The ICMP latency check maybe a useful metric, but as discussed in this thesis, we cannot rely solely on it; hence multiple metrics should be taken into consideration.

Also in this chapter we highlighted an approach to detecting the migration of critical VMs using co-located light-weight VMs. The main purpose of these VMs is to track the critical VMs by sending ICMP pings and analysing the latency of replies. The light-weight VMs are small and cheap, yet, they have to run a full OS whose capabilities will not be fully utilised. In [142], it was argued that operating systems often consume more memory and disk space than the actual application they host. Thus, it makes sense to devise a more economical method of carrying out tracking of critical VMs than deploying a VM with full OS which is not utilised. The concept of containers sounds like a good alternative to light-weight VMs; especially given the fact that each container can have its own network stack, e.g. IP address. This means that it can be reachable from the internet much like how we reach typical VMs using their external IP addresses. Hence, we argue that the consumer could make use of containers

[7]CRIU: https://criu.org/Main_Page
[8]P.haul: https://criu.org/P.Haul

instead of VMs for tracking and generating profiles. This may have some benefits:

- It is unlikely that containers will get migrated frequently: live migration of VMs usually happens to load balance the resources by moving VMs from a busy server to another one which is not utilised. With containers, given that they can be small in size, there may be less tendency to migrate them. This means that if containers are used for tracking, they will remain where they are for a longer time.

- Given their small size and low cost, it may be possible to deploy more tracking containers than VMs. This can result in a more accurate outcome after analysing ICMP latencies from critical VMs to the tracking containers.

Since tracking critical VMs requires the creation of periodic profiles to be analysed later, questions remain on how to store the profiles in the container and how to send them from the container to the inspector station in an efficient way. Also, the necessary container size to be able to perform this needs to be determined. In other words, can we build a container which does exactly what we want it to do while retaining its small size, so that it does not cost much in terms of resources, e.g. storage, processing power, etc? We argue that this area needs further exploration, and due to time constraints, this was not possible, as well as being beyond the scope of the main work.

5.12 Chapter Conclusion

In public IaaS cloud systems, virtual machines can be migrated to another (jurisdictional) region/availability zone other than the region where the VMs were initially deployed. This is performed by utilizing live migration techniques, where VMs can be migrated by the cloud provider, without having to shut down services. Cloud providers may perform the live migration of VMs to: (1) load balance requests across multiple physical data centres; (2) reduce the cost of energy in data centres by moving to regions where energy is cheaper; (3) performing maintenance of data centres. This has often been identified as being of benefit for both the consumer and the cloud provider, in that the consumer does not see any interruption in service when the migration takes place, and for the cloud provider to benefit from reduced operational costs. Cloud providers

generally do not engage consumers in such migration decision. In this chapter, we argued that hiding the migration event from consumers could pose security and privacy concerns, which may include: possibility of VM theft and Denial of Service, Data Privacy and violation of SLA terms. Thus, we explored various techniques to detect the migration of VMs in public cloud systems from the perspective of the consumer. An example of the detection techniques is observing the ICMP latency between VMs. We have indicated that the number of hops between VMs -via the traceroute tool- can be useful to decide if the VM is in the same datacenter or not. However, due to the possibility of hiding the hops using firewalls, this metric might not be applicable.

We noticed that some of the techniques are suitable for detecting the migration at various stages: just before it happens, while it is happening and after it has happened already. Some of them are more suitable for detecting the migration inside the datacentre, while some are suitable for detecting it when it happens to another region. Thus, we proposed a spatio-temporal taxonomy for the migration detection techniques to help achieving detection with minimized performance and storage overheads. Some explored techniques, for example "Time-lag detection", require that the recording of metrics is performed more frequently inside the VM. This is why considering such a spatio-temporal taxonomy of the detection techniques may help to ensure that the impact of detection on storage and performance is minimized. Then, we evaluated one of the VM migration detection techniques (the virtual landmark fingerprint) by utilizing VMs hosted in Amazon Web Services (AWS). The technique relies on the use of ICMP to measure the latency between the VM and a set of selected general internet landmarks. The evaluation showed that this technique may give a good estimate that migration of VM has happened. However, it may result in inaccuracies due to various network issues. Therefore, we proposed another hybrid approach which combines the use of multiple migration detection techniques (referred to as: *migration metrics*), to estimate the likelihood of the migration event. The approach considers the importance of the metrics from the perspective of the cloud consumer and uses a weighted decision function to generate the probability of the migration event. By using the combined approach, we argued that it may not be enough to rely only on a single metric in the detection process as this might lead to inaccurate outcomes. An experimental evaluation of our approach was carried out on VMs hosted on the Google Cloud Platform (GCP) where we considered four migration metrics: ICMP latency, External/Public IP, hypervisor type and processor information. Few light-weight VMs were used track the critical VMs. They maintain ICMP connections between the

monitored VM and between the other light-weight monitoring VMs; all VMs within the cluster will be linked together by ICMP interactions. Investigating the interactions between the VMs and observing the state of the *migration metrics* after the migration, indicated that our approach is suitable. The consumer can have a greater assurance that their VMs have not been migrated to another region. Using our approach, which relied on four metrics as a proof of concept, detecting the migration of VMs to another zone within the same region may be possible; if the CPU architecture of the servers in that zone is different. In cases where zones in the same region use the same CPU architecture, additional metrics will be required. However, migration of VMs within zone of the same region does not usually raise security concerns.

In practice, most cloud providers use a specific type of hypervisor which is used in all data centers. In such case, where hypervisors might not change, detecting a different version of them may indicate a migrated VM. VM live migration across hosts with different hypervisors require an OVF compliant hypervisors and VMs; using the open standard for packaging and distributing VMs. We covered some limitations with the combined approach and argued that having the profiles of the VMs sent to a central supervisory machine to be analysed may give accurate results. This is to be addressed in Chapter 6, when we discuss the architecture of the supervisory system.

The chapter also discussed briefly the possibility of detecting the migration of containers; as they are similar to VMs with some major differences. In practice, providers also utilize VMs to run containers; thereby benefiting from the isolation provided by the hypervisor. We concluded that the area of detecting the migration of containers needs more exploration; especially that tools to support the migration are still not supported in many container management systems.

Chapter 6

Supervisory & Analysis System

6.1 Chapter Overview

This chapter proposes a general events management and analysis architecture of a system that aggregates events captured from multiple VMs, belonging to the *same* consumer. The aggregation and analysis of the events of multiple VMs is performed: (1) to enable the consumer to learn about the overall security status of monitored VMs in real-time; (2) to find patterns and relationships between security-related VM events which occur across multiple VMs; (3) to ensure that VMs are not migrated to a different region. The architecture consists of four main components: the policy centre, the event centre, the report centre and the alert centre. All the components are integrated to together to provide the consumer with analysis, alerting and reporting facilities. We proposed suggestions for finding patterns in the observed security-related events across multiple VMs. The patterns identify whether events are related to each other, e.g. triggered by the same user across several VMs. The architecture was partially implemented using some selected open source tools such as: Filebeat, Logstash and Kibana. To illustrate the usefulness of the architecture, several experimental scenarios involving VMs hosted on the Google Cloud Platform (GCP) are highlighted. The chapter provides an evaluation of the proposed architecture against four aspects: performance, security, storage and usefulness.

6.2 Introduction

The VMInformant architecture highlighted in Section 4.7 allows the monitoring and recording of security-related events in a single VM. The events are stored in a log file

before they are sent to the consumer for analysis. We have established that analysing the security events of the VM gives the consumer a greater visibility on the malicious activities which occur inside it. However, if there is a large number of monitored VMs, which may generate a large number of events, a suitable mechanism to manage these events is needed. Besides, it may not be enough to consider the events of a specific VM in isolation of the events which occur in other monitored VMs. Finding patterns in the events which occur across multiple VMs may enhance the visibility and help the consumer make informed decisions. For example, several security-related events may be triggered by *VM-user1* in multiple VMs; one of which he is not authorised to use. This may imply that another privileged user is involved, who gave access to VM-user1. Alerting the consumer to such incidents as soon as they occur, may resolve security and privacy issues and improve the security of hosted applications. Therefore, there is a need for a security event management system, which manages and analyses the events generated by multiple VMs. This system should help the consumer learn about the overall security status of the monitored VMs, including checking whether any of the VMs have been *migrated*.

Thus, we hypothesise that *managing security-related events which occur across multiple VMs owned by the same consumer, supports trustworthy cloud computing.* The event management and analysis are performed from a consumer's perspective. This means that cloud providers do not contribute VM event data to the consumer nor do they analyse the events on behalf of the consumer. In general, this chapter attempts to answer the following two questions: (1) by monitoring security-related events which occur across multiple VMs, is it possible to find patterns which give the consumer an improved visibility on the overall security of VMs?; (2) From the perspective of the event management system, how can we identify which of the VMs have migrated/moved away from the ones interacting with it?

To answer the aforementioned questions, we propose an architecture of a system to manage and analyse events received from VMs in public IaaS cloud systems. We call the system: *Inspector Station*. The architecture consists of four main components: the policy centre, the event centre, the report centre and the alert centre. In general, the *policy centre* is responsible for configuring the parameters related to how events are stored or reported, and how the consumer will be alerted. The *event centre* receives events and analyses them based on input from the policy centre. While the *report centre* generates useful reports for the consumer, the *alert centre* is responsible for alerting

the consumer/administrator about the events– according to the rules predefined in the policy centre. In addition to analysing the events to find patterns, the event centre applies a general algorithm to detect the migration of any VM within a virtual cluster/group of connected VMs. The concept of the virtual cluster has been discussed in Chapter 5, Section 5.10.2.

Figure 6.1 shows a high-level view of the integrated systems. As can be seen from the figure, migration-related VM profiles are dispatched by the *Life-cycle Event Engine* to the *Event Mapping Engine* in *VMInformant*, where they can be included in the provenance file. The *Life-cycle Event Engine* manages VM life-cycle events (Section 4.5.1,p 68), locally in the monitored VM, e.g. data related to migration events. Later, the provenance log file will be sent to the inspector station for analysis.

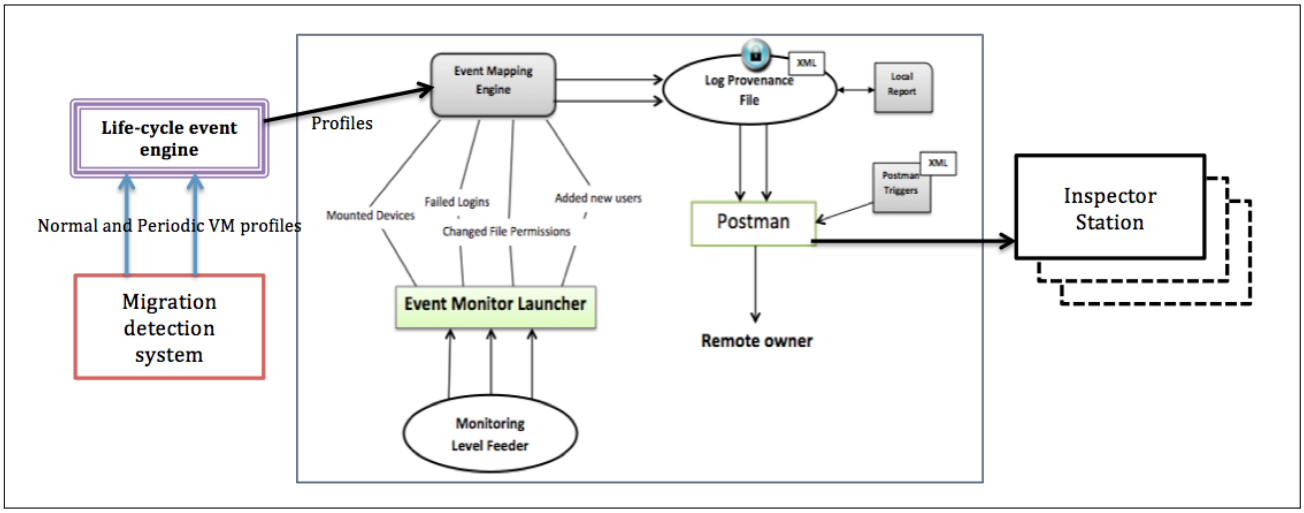

Fig. 6.1 A high-level view of VMInformant, migration detection system and inspector station

In this chapter, we describe the suggested architecture of the inspector station. We discuss a framework to find patterns within the pool of aggregated events from multiple VMs. Then, we discuss how migration of VMs can be detected from the perspective of the inspector station. As a proof of concept, we then highlight how the architecture can be implemented by combining the VMInformant tool and some open source tools such as: *Filebeat*[1], *Logstash*[2] and *Kibana*[3]. We then provide an experimental evaluation

[1]https://www.elastic.co/products/beats/filebeat
[2]https://www.elastic.co/products/logstash
[3]https://www.elastic.co/products/kibana

of the use of the architecture by following a scenario-based approach. Finally, the chapter provides an evaluation of the proposed architecture of the Inspector Station system against four aspects: performance, security, storage and usefulness.

This chapter provides the following contributions:

- The architecture of an event management system to manage and analyse events of multiple VMs. We believe that this contributes to the area of event management from a consumer's perspective, where specific questions about the overall security of monitored VMs can be answered. This can help the consumer attain better visibility on what is taking place in all the VMs they *own*, which allows a more informed decision to be made.
- A general algorithm to detect VM migration from the proposed inspector station.
- An illustration of how selected open source tools could be used to aggregate and analyse events.

6.3 System Architecture of the Inspector Station

The system architecture of the inspector station consists of four main components: *The Event Centre*, *The Policy Centre*, *The Alert Centre* and *The Report Centre*–Figure 6.2. The following sections highlight the role of each of these components.

6.3.1 The Policy centre

This component serves as the backbone of the architecture, and all other components refer to it to carry out their operations. In general, the role of this component is to allow the administrator/owner of the monitored VMs to configure several parameters concerned with the storage of events, alerting, and reporting. These tasks are summarised as follows::

Configuring event types and groups

Security-related events observed in the VMs are stored in the provenance log file in a certain text format based on the type of event. Each event belongs to a specific event group as outlined in our taxonomy of the security-related events highlighted in Section 4.5. The policy centre will map each event to an event group according to the taxonomy. This helps categorise events when reporting is needed.

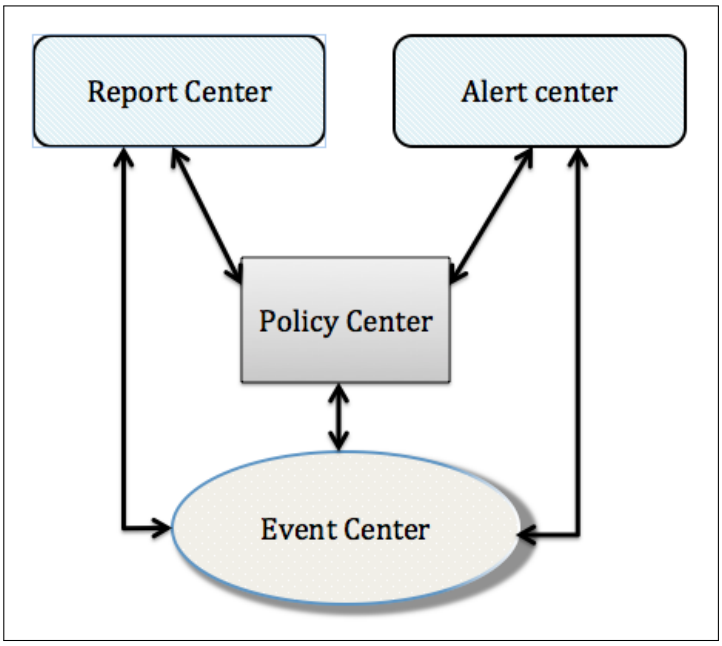

Fig. 6.2 Architecture of the Inspector Station

Configuring the granularity of monitoring in the critical VMs

The VMInformant architecture is flexible in that it allows determination of the level of detail at which monitoring is needed, e.g. monitoring specific file-centric events rather than monitoring all of them. It contains local configuration files to allow this in addition to the GUI. The policy centre in the inspector station can update the configurations remotely.

Configuring the legitimate services/processes

Assuming that not all the critical VMs perform the same tasks, there may be processes which are authorised to run in these VMs, while there may be others which are not. The policy centre maintains a list of the legitimate processes to run in each VM, such that if any illegitimate process is observed, the consumer will be alerted by the alert centre. Similarly, this is also useful in cases where a legitimate process has disappeared from one of the VMs.

Configuring the importance of metrics

This is useful in detecting migration events; where the migration decision function requires determining the importance of the metrics to calculate the decision index

(discussed in Chapter 5, Section: Section 5.8). The importance of metrics is a consumer defined variable which can be defined in the policy centre (a value between 0 and 1).

Configuring the priority and the severity of the events

Multiple streams of malicious events could be observed from a particular VM. The consumer may be looking at events of particular interest to them. The policy centre will configure the priorities of the events from the consumer's perspective. In addition, it configures the action(s) to be performed regarding alerts and notifications. The priority of the events could be given on scale from 1-low to 5-high. For example, in detecting an event which has the highest priority "5", the action to be taken is to bring the event forward so it can be noticed, alerted, and an SMS notification delivered to the administrator.

Some events may be more severe than others due to the impact they may have. For example, from a consumer's perspective, attaching a device to the VM may be considered more severe than viewing a single file– due to the potential damage this may cause. Thus, we introduce a *severity index* which can be associated with every event. This is entirely up to the consumer/administrator to define. However, they can make use of historical data and reports from information security organisations to determine the severity levels. For example, Symantec[4] bases its severity assessement of computer threats on three metrics: Wild, Damage and Distribution [144]. According to their assessment, if the event resulted in deleted or modified files, the *Damage* metric will have a "high" severity level. We argue that it would be useful for the consumer to classify security-related events based on similar approaches. For example, if the security-related event is observed in 10 VMs or more, then the severity can be considered "high". The severity index of the events could be given on scale from (1-low to 5-high).

Configuring the retention period for acquired VM events

The number of events which are received and processed by the inspector station could increase , and the consumer may choose to delete some of these events after a certain period of time. Some of the events may be retained for longer periods than others, but this is up to the consumer to decide. The retention decision can be for a specific event type or a specific event group. A periodic profile, which arrives at the event centre

[4]https://www.symantec.com

waiting to be analysed, may not need to be retained for a long time since it is only needed to detect the migration event. The retention decision can be made based on statistics which involve the number of monitored VMs and the amount of analysed events. Severity of the events and their priority may also be taken into account when determining the retention policy.

6.3.2 The Event centre

The event centre performs most of the analysis tasks. It retrieves the configurations from the policy centre in order to prepare the events and make them ready for the other components namely: the Alert centre and the report centre. The event centre receives two types of data structures: (1) collection of events as recorded by *VMInformant*; (2) collection of VM profiles (to serve the migration detection process). The distinction is important here because by the time profiles are received at the event centre, the migration event or the likelihood of it is still not concluded. The event centre has several duties:

1. **Receiving the events and VM profiles:** A security-related event may be received as a record which includes many attributes, as discussed in Chapter 5 (records can be in XML). The record contains attributes such as: *event-id, time-stamp, user-id, user-group, etc.* In the case of VM profiles, they are also received as records which include different attributes, such as: *profile type (normal/periodic), VM type (critical, monitoring(light-weight)), time-stamp, latencies, metric 1 value, metric 2 value, etc.*– as discussed in Chapter 5.

2. **Organising the events and VM profiles:** These are classified by VMs according to the event groups. Given the event-id, the event centre will fetch the relevant event group from the policy centre. In the case of VM profiles, the event centre will also organise the obtained profiles and prepare them for analysis.

3. **Analysing VM profiles to detect VM migration:** We argue that detecting migration from the perspective of the inspector station is different than detecting it from each VM individually. The detection has to consider the metrics of the critical VM and monitoring VMs all at once. Consider the two scenarios highlighted in Figures 6.3 and 6.4. The first figure shows the normal state of all the VMs (vm3 is monitored by vm1, vm2 and vm4). The second figure shows the

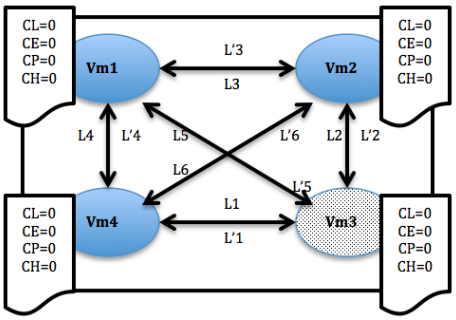

Fig. 6.3 normal state of the VM and the metrics (normal profile)

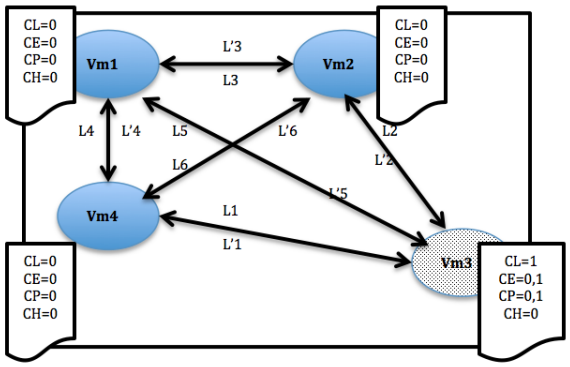

Fig. 6.4 state of VMs and metrics when migration happens

periodic state of the VMs and the migration metrics in each VM– after migration takes place. The question is can we develop a general decision algorithm which can be applied in the inspector station to detect the migration of any VM (e.g. VM_k)?

Thus, by analysing the scenario and the state of the metrics, we derived the following condition to decide if a VM_k has migrated (note: the variable *'allVMs'* refers to all the other VMs which interact with VM_k–light-weight monitoring VMs:

$$\text{If CL } ((VM_k, \text{allVMs}) \text{ \& CL } (\text{allVMs}, VM_k) \text{ \& !CL}(\text{allVMs},\text{allVMs})) \text{ \&}$$
$$(\text{!CE}(\text{allVMs}) \mid \text{!CP}(\text{allVMs}) \mid \text{!CH}(\text{allVMs}))$$

The condition means that in order to decide on the migration of VM_k from the inspector station: (1) there has to be a change in the latency metric (CL) between VM_k and all the other monitoring VMs; (2) a change in the latency between VM_k and all the other VMs; (3) no change in the latency among all the other VMs; (4) at least one of the other migration metrics in the other VMs has to remain unchanged. Generally, from the experiments, the average latency measured from VM_x to VM_y is similar to the latency from VM_y to VM_x. However, because the inspector station considers profiles sent by VMs individually, we chose to include the first two parts of the condition; as outlined above. The condition can be used to detect the migration of any given VM. Alternatively, the inspector station could just consider applying the decision function, as outlined in Chapter 5. In that case, the event centre will fetch the *importance* values of the metrics set by the administrator from the policy centre. The decision index can then be

calculated by comparing the periodic profiles with normal ones. If the result of the decision function is greater than zero, it may indicate the relevant VM has been migrated. The policy associated with the obtained decision index has to be checked from the policy centre to decide whether to alert or not.

4. **Finding patterns in the observed events across multiple VMs:**

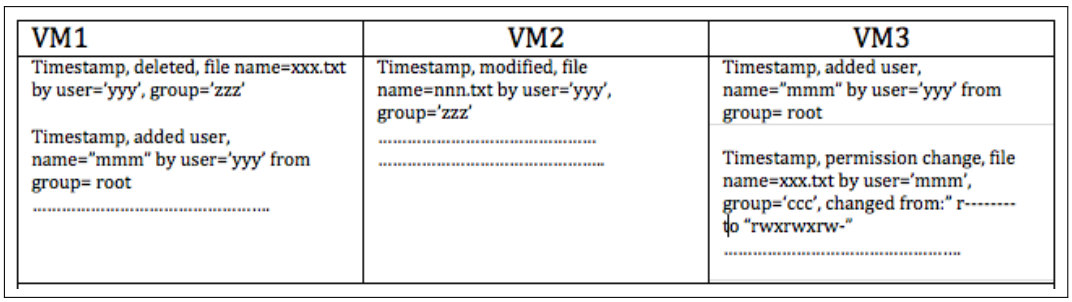

Fig. 6.5 Events as observed from multiple VMs

Figure 6.5 illustrates a possible situation where observed events in multiple VMs may be related to each other. For example, in VM1, user: 'yyy' added the user name: 'mmm'. The same user also modified a file in VM2. The added user in VM1 was also added in VM3. Meanwhile, permission bits of one of the files in the same VM (VM3) were changed by the added user. From the scenario, user 'yyy' is involved in events which were observed in multiple VMs. Finding patterns and correlating events can be useful for the owner of the VM to learn about the security status of VMs and determine the sequence of events as performed by users. We suggest that the questions to be answered concerning events across multiple VMs may include (but are not limited to):

- Given all events in all VMs, how many are triggered by the same user?
- Have any files with the same names been affected in more than one VM and what were these VMs?
- How many files have been modified in all VMs?
- What are the files in all VMs which were affected by the change in permission?
- What was the permission changed into? Are these files related, e.g. do they relate to a known project? Have they been modified by the same user? After the change in permission, have they been accessed/modified by another unknown user?

- What are the VMs which included "port opened" events? And what were these ports? Similarly, were there any ports closed in the VMs preventing production tools from working properly?
- Has a device been attached to any VM? During which user session was this performed?
- Were there any security related events just after a port was opened?
- Were there any security related events just after a device was attached to the VM? Did this happen to only one VM or to multiple ones?
- How many of the events were executed by root users, what were they, in which VMs, and within what time frame?
- Were there any VMs with minimal security exposure, i.e. few observed events?
- Were there any VMs with significant security exposure, i.e. significantly altered documents?
- Is there any relation between the timing of the events in multiple VMs, i.e. observed at a specific time frame where no activity is supposed to take place?
- If a time-stamp exists of multiple events in different VMs, whose initiation was by the same user, are very similar, does that mean it was an automated attack or a manual attack? Can we infer something from correlating the timings of the events?
- Are the events in multiple VMs related to each other, e.g. same user, same time frame, same affected files ? Are the affected files related (e.g. Having the file name, created by the same user, etc.)?
- Was any non-listed file executed in multiple VMs, and what other events were triggered by the execution of this particular file, was the creation/deletion of any other files triggered?
- Did any legitimate processes suddenly disappear in the VMs? Are process-related events in multiple VMs related to each other, e.g. by the same user, in the same timeline, etc.?
- Has any user been created in multiple VMs? Is he authorised to access/use these particular VMs?

- Have any events been triggered in VMs outside working hours? By which users? Have these users been responsible for security-related events in the past and in which VMs?

The aforementioned questions are just guidelines to develop an analysis strategy which spans multiple monitored VMs. The conceptual diagram in Figure 6.6 indicates the relationships between the events, VMs, users, time-stamp, and all the others are handled by the components of the architecture. The digram was derived from the understanding of the relationships between all the entities handled by the the architecture. For example, it shows that an event can be traced to a particular user at a specific time; users are authorised in specific VMs, a process is involved in an event where it could have appeared/disappeared in one or multiple VMs, etc. The diagram may be used as a basis for developing an event management system to find patterns in the events which occur across multiple VMs. In addition, it can be used to generate useful statistics. For example, it may be used to generate statistics about the number of files modified by a certain user-id in set of VMs, within a specific time-frame.

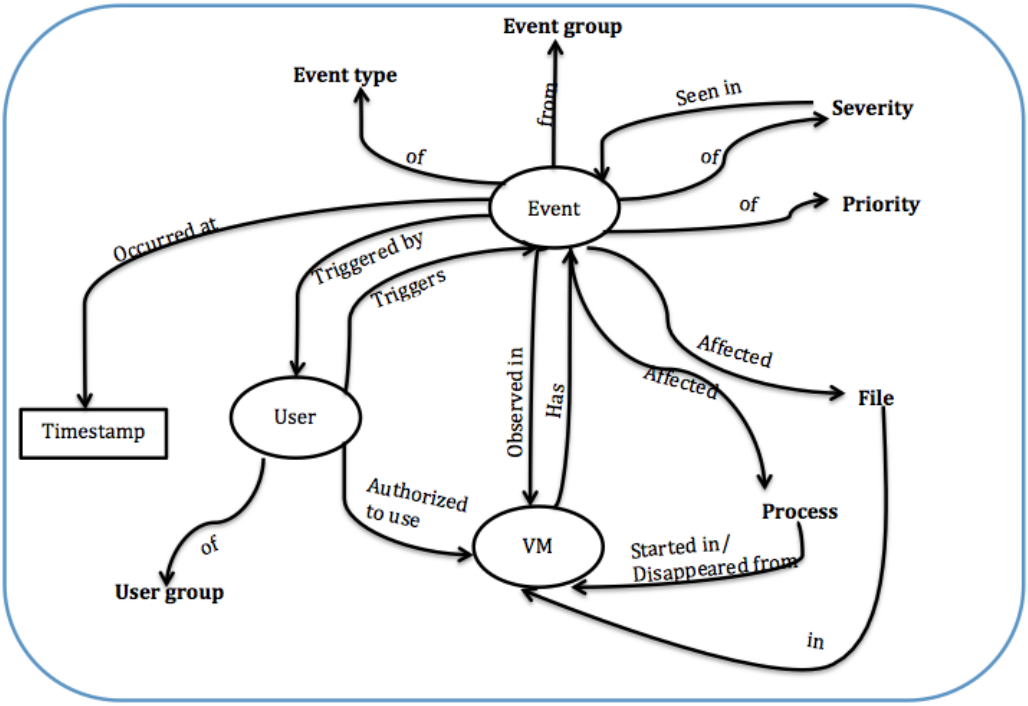

Fig. 6.6 A conceptual diagram to help finding patterns in the events of multiple VMs

The event centre receives events from VMInformant can be using: (1) ssh clients– where logs are copied securely to the inspector station; (2) using specialised agent

daemons such as: *splunk* or *collectd*– see Section 6.7. There is one important point to note here; the events may be received (1) in bulk as a group of events sent at a specific time on the day previously configured by the consumer; (2) real time: as they occur. In either case, the event centre organises the events and applies the roles according to the policy centre configurations. There may be times when VMs do not communicate with the event centre for a long period. This could indicate problems, that the VM is off, or that the agent was off or it was compromised. To maintain connectivity with monitored VMs and check continuity, we suggest periodically sending a *dummy* event from the monitored VMs to the event centre in the inspection station. The purpose of the dummy event is to check that monitoring is on. The event can be discarded later.

6.3.3 The Alert centre

The alert centre is responsible for alerting the administrator/consumer of events. It makes use of the roles embedded in the policy centre to decide when and how to alert. Assuming there is a live dashboard that displays events after they are analysed, the alert centre will decide which specific observed event or specific event group should stay on top given that the consumer still has not acted upon them. The decision is performed after checking the priorities of the events and the associated alert action. In the case where the priority of a specific event or event group is 5, for example, the alert centre integrates with an SMS server to send a message to the administrator to allow immediate action.

6.3.4 The Report centre

The report centre presents the analysis obtained from the event centre to provide the consumer with useful information about events. It enables the consumer to query the system for a specific event or specific VM, etc. The report centre may be configured to generate reports according to the needs of the consumer, e.g. based on the questions highlighted in Section 6.3.2. Thus, patterns related to the events across multiple VMs can be reported to the consumer. Statistics about the security status of the VMs can also be generated by the report centre.

6.4 Trust Profile

We argue that, with time, by observing the incoming events and ranking them according to severity, the consumer/owner may be able to establish what we call a *trust* profile.

The *trust* profile is the consumer's perception of how the cloud provider can be trusted with hosting VMs. This can be subjective, but it can give an estimate for the consumer. If VMs of the consumer are hosted with several cloud providers, the trust profile can be generated in order to determine which of them can be relied on. Since all the events which are captured are security-related, we assume that the greater the number of observed events in VMs hosted by a specific cloud provider, the lower grade this provider will receive, and this will be clear from the trust profile. This is only applicable is the events were found to be triggered by employees of the cloud provider. Several factors may contribute to forming the trust profile: number of affected VMs per each provider, number of events per VMs, and severity of the events based on the severity index obtained from the policy centre. If the trust profile of the cloud provider is consistently negative, the consumer may choose to switch providers.

6.5 VM Health Profile

By checking the severity index of the observed events and their volume, a VM *health* profile can be built. This profile shows how healthy the VM is in terms of the number of events and their severity from the perspective of the consumer. The following function may help determine one of the VM health metrics of a particular VM:

$$\begin{aligned}\mathbf{d(m)} = \sum_{\forall i} & \mathit{Severity\ Index\ of\ Event}_1 * \mathit{Number\ of\ Events\ of\ Type\ Event}_1 \\ & + ... + \mathit{Severity\ Index\ of\ Event}_i * \mathit{Number\ of\ Events\ of\ Type\ Event}_i)\end{aligned} \tag{6.1}$$

The higher the VM health metric, the lower the health state of a particular VM. Only events observed in a particular VM are considered. A healthy VM can be traced back to its cloud provider, and this can support the trust profile of the provider.

6.6 Experiments

Due to time constraints, the development of a fully *dedicated* system for analysing the events according to the inspector station architecture was not possible. Instead, we combined our tools (VMInformant, migration detection system) with off-the-shelf open source tools. In this section we highlight the set of experiments carried out to validate and evaluate concepts discussed in the chapter. The experiments are carried out on a public cloud platform (Google Cloud Platform).

6.6.1 Aim

As discussed before, the inspector station architecture allows the gathering and analysis of security-related events from various VMs. It also provides altering and reporting facilities. The main purpose is to assist the owner of the VMs in making trust judgements and decisions based on the monitored events. Thus, in the experiments we apply scenarios which will trigger events in VMs, and show how the owner can make use of this to make an informed decision.

6.6.2 Setup

We chose the Google Cloud platform to perform the experiments for several reasons: (1) It is a public cloud– which matches our main research focus;(2) they provide a free two-month subscription to use their platform and they provide enough quota for that;(3) it provides an easy to use interface for accessing and managing VMs using *ssh* via the browser, and also to transfer files to the VM when needed; (4) it also allows the transport of instances (migrating them) to another zone or region by using the *glcoud* tool. This tool is one of the tools provided in the *Google Cloud SDK*[5] which allows remote management of VMs using the command line.

We first created a template for the monitored VMs called: *informant-client*. This is equipped with:

- **Ubuntu 12.04:** Any Linux-based OS can be chosen.
- **The Auditd deomon**[6]: The auditd sub-system is an access monitoring and accounting for Linux developed and maintained by RedHat. The VMInformant makes use of auditd to retrieve certain events and analyse them in order to produce the provenance log file (see Chapter 4).
- **VMInformant:** This is our prototype implementation, as described in Chapter 4. VMInformant was implemented in Python and it has a GUI to make it easier for non-experienced VM owners to configure the monitoring. Hence, all monitored VMs are required to have a GUI desktop environment.
- **Migration detection system:** This is our prototype implementation for recording migration metrics from each of the monitored VMs. It is configured to

[5]Google Cloud SDK https://cloud.google.com/sdk/
[6]https://linux.die.net/man/8/auditd

commence automatically when the system starts, once enabled by the owner. We modified the system slightly to allow recording the metrics only in order to be sent to the inspector station– without applying the decision function from the monitored VM.

- **GNOME:** This is the desktop system for Unix and Linux systems that provides an easy to use desktop.
- **VNC server:** to access the GUI of the monitored VMs when needed. This was configured to automatically commence when the system starts.

The template can be used to create as many monitored VMs as needed. As a proof of concept, we use three *monitored* VMs (informant-client1, informant-client2, informant-client3). We also deployed a another VM (*inspector station VM*) to capture the events from the monitored VMs. The inspector station has similar software as the template of the monitored VMs, except that it does not have the VMInformant tool. Additional software tools will later be added to the monitored VMs and to the inspector station to allow shipping of the events and to parse/analyse the events in the inspector station. In addition to creating the inspector station VM and the monitored VMs, we created three VMs to act as *light-weight monitoring* VMs. All the monitored VMs maintain ICMP interactions with the monitoring VMs. This is useful for detecting the migration using the approach discussed in Section 6.3.2.

In our local machine which controls the experiments, the Google Cloud SDK was already setup. We use the *gcloud* command line tool for most of the operations. Although the web console provides an easy way to manage the VMs, the gcloud tool offers more powerful control. Besides, some of the operations, such as migrating instances, have to be performed using the gcloud tool. It will also be used to effectively copy files to the monitored VMs.

6.6.3 Design

The experiments are scenario-based, meaning that we test several scenarios that will involve malicious behaviour. Each scenario will result in triggering events which will be captured by the inspector station and traced back to the monitored instance from which the event originated. The scenarios include:

1. ***Scenario 1 (Manipulation of files scenario):*** In this scenario, certain folders and files are monitored for manipulation. Touching these files will trigger

events which will be sent to the inspector station. The owner will be able to determine: 1) the affected VM; 2) which user initiated the process; and 3) what time it was performed along with other attributes.

2. ***Scenario 2 (Malicious script scenario):*** In this scenario, a malicious script will be executed in the potential VM to perform specific operations which aim at damaging the VM or disclosing confidential information. We consider the following use cases:

   - Generating a large number of files in one VM for the purpose (possibly) of bringing the machine down.
   - Copying whole folder contents to another location for the purpose of stealing confidential data.

   Injecting scripts into the VM and executing them will trigger some *process-related* and *file-centric* events which will be identified by the inspector station.

3. ***Scenario 3 (Migrated instance scenario):*** Based on the migration detection concepts discussed in Chapter 5, the normal profiles of all the monitored VMs will be generated and sent to the inspector station. The monitored VM (in our experiment this is the *informant-client* VM) is monitored by three light-weight monitoring VMs. All the VMs (the monitored and the monitoring VMs) are interacting with each other using ICMP ping. In this scenario, the potential VM is migrated to another zone or region. After the migration, periodic profiles are generated. The inspector station compares the new periodic profiles with the existing normal profiles and checks the *migration condition* outlined in Section 6.3.2.

6.6.4 Methodology

In order to simulate the scenarios on the monitored VMs, we prepared some scripts to control the experiments from our local machine. The affected VM will be chosen randomly from a pool of the monitored VMs– which we created previously. This is to mimic real life situations whereby any VM can be a target without the owner's knowledge. Also, since the owner will not know the timing of the malicious events, we chose a random waiting time during which a script will do nothing before starting the

scenario process. Based on the scenario, relevant executable script code will be sent to the arbitrarily chosen VM along with commands to execute. Figure 6.7 illustrates the experimental methodology.

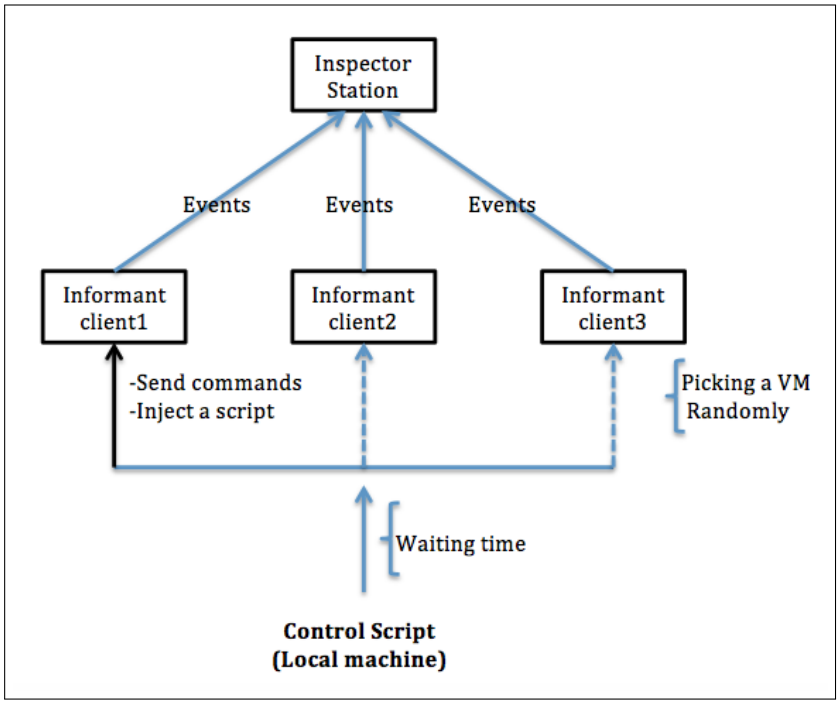

Fig. 6.7 Controlling the experiments

6.7 Implementation

There are two important points to implementation. First, we need to set up the tools to collect the monitoring logs from the monitored machine and send them to the inspector station. Second, we need to start implementing the scenarios as discussed previously.

6.7.1 Setting up the monitoring environment

We have already mentioned that VMInformant and the auditd system are installed in the monitored VMs. Based on rules manually added to auditd, the latter will start to store the events to */var/logs/audit/audit.log*. Using VMInformant, rules will be added automatically from the GUI according to the consumer's preferences. This allows the creation of multiple threads which monitor changes at multiple locations on the disk. Because the log file entries obtained from auditd can contain a large volume

of information which needs further analysis, VMInformant performs an analysis on them in order to provide an informative record and present it to the owner in the provenance log file. The log file of VMInformant is stored in a location determined by the consumer. However, the question of how to send the log file(s) to the inspector station remains.

For this, we have several options. We could use *splunk*[7]. Splunk is used to analyse a large set of system logs. It allows a distributed analysis of logs gathered from remote hosts using the concept of *forwarders* and *indexers*. Forwarders are *splunk* agents deployed inside remote machines and configured to send selected logs for monitoring. Indexers are the machines which receive logs and perform an analysis or search on them.*Splunk forwarders* can be installed in the monitored VMs and configured to send logs to the *splunk indexer* (inspector station). Since splunk only offers limited indexing volume per day for free, we decided to find an alternative. Another option was to use *collectd*[8]. *Collectd* is an open source tool for collecting monitoring states which is highly extensible and supports all common applications, logs and output formats [152]. Some cloud providers, such as Rightscale[9], use it as part of their own monitoring solutions. Collectd is mainly concerned with sending performance metrics of the monitored VMs to a central location for analysis. The metrics can later be visualised using graphical tools such as: Graphite[10]. Collectd clients can be configured in the monitored VMs to send metrics to the collectd remote server (in the inspector station). It uses a large suite of plugins to send the metrics to other applications for analysis. Although collectd mainly focuses on collecting performance-related metrics from the monitored instances, some of the plugins enable collectd to send logs to data processing applications such as *Redis*[11].

In this research we are not focusing on performance metrics. Besides, making use of the log files in the inspector station after being collected from the monitored VMs by *collectd* was not straightforward. Thus, we decided to use a specialised tool to ship the logs from the monitored VMs named *Filebeat*[12]. Filebeat is a light-weight shipper for logs which can easily be configured to send logs to a central remote site. It uses the concept of prospectors, where each prospector has paths to look for. Each path

[7] http://www.splunk.com
[8] https://collectd.org/
[9] Rightscale http://www.rightscale.com/
[10] https://graphiteapp.org/
[11] http://www.redis.io
[12] https://www.elastic.co/products/beats/filebeat

will contain the location of the log files to be shipped. Therefore, we installed Filebeat in all the monitored VMs (informant client machines) and configured it to send logs (VMInformant log files and other logs) to the inspector station machine by specifying the IP address.

In the inspector station, logs sent by Filebeat have to be received and processed effectively and securely. This is one of the tasks of the event centre in the inspector station architecture. In order to implement some the features of the event centre by using off-the-shelf tools where possible, we used *Logstash*[13], an open source processing pipeline that ingests data from a multitude of sources simultaneously and transforms it before sending it to another application for further processing. It is generally used to prepare the data to be sent to the *Elasticsearch*[14] engine, which can receive structured and non-structured data and perform complex searches on it. Another option that Logstash offers is to prepare the data to be sent to Kibana[15], an open source visualisation tool that integrates Elasticsearch capabilities to allow visualization of the data in real time. It provides features which are similar to the *splunk* dashboard, but has more tools and allows dashboard customisation.

Therefore, we equipped the inspector station with: Logstash, Elasticsearch and Kibana and configured them to receive events data from the monitored VMs. Because we created a template for the informant-client VM (monitored VM), as soon as we launch the monitored VM, the inspector station will be able to capture its events automatically. These events will be displayed by the Kibana visualisation tool. Figure 6.8 illustrates how events of the various monitored VMs are received by the inspector station.

It is worth mentioning that *Logstash* allows the use of SSL certificates to secure the communications between instances with Filebeat and the server hosting the Logstash. This is to allow the instances to send events to the authorised server only, and to instruct the server to receive data only from authorised instances. We created the certificate using *openssl*[16]. The public certificate was then copied to all the monitored VMs. Figure 6.9 shows how the tools in the monitored VMs and the inspector station are integrated. It also shows how the owner or the administrator of the monitoring

[13]https://www.elastic.co/products/logstash
[14]https://www.elastic.co/products/elasticsearch
[15]https://www.elastic.co/products/kibana
[16]https://www.openssl.org/

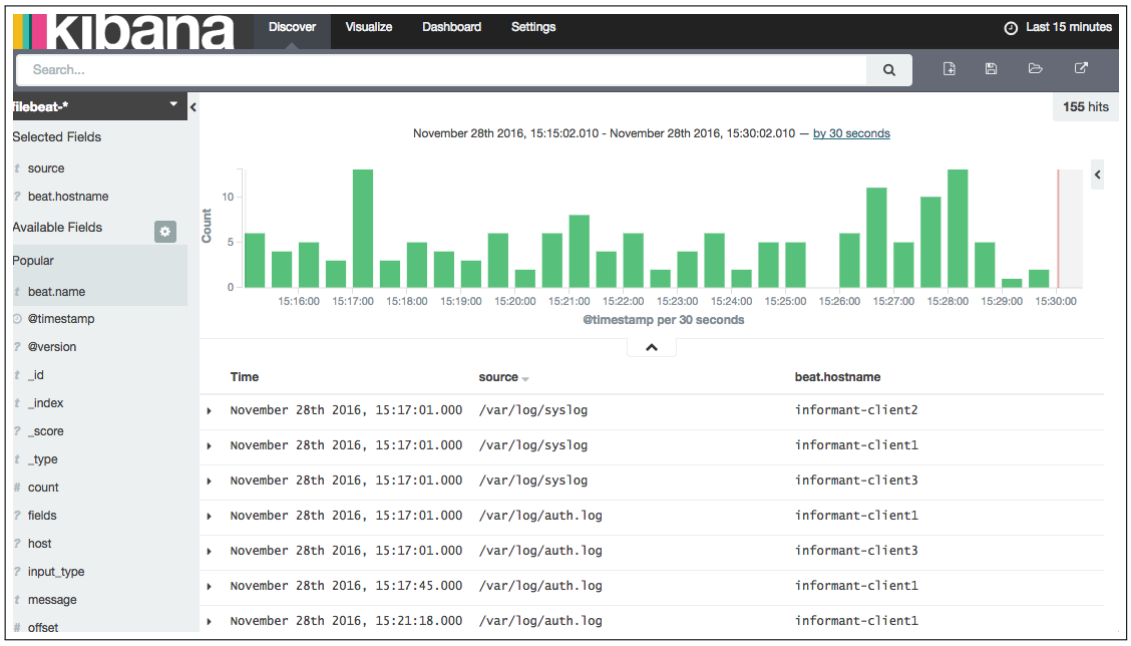

Fig. 6.8 Kibana web visualisation tool receiving inputs from various sources

process can use any web browser to check the events from various monitored client machines. In the experiments, we send the entire log produced by auditd to the inspector station. The prototype implementation of VMInformant currently lacks certain features related to the detection of malicious processes. Because of this, in order to check some processes we analyse the *auditd.log* directly to obtain process-related event information. In addition, we make use of another tool called: *snoopy*[17], which is an open source command logger that records all commands typed into the machine by all the users and stores it in /var/log/auth.log. This will be useful in some of the scenarios covered in subsequent sections.

On top of the tools in the inspector station machine, we implemented a script to perform analysis tasks on the events that categorises the events into event groups. It currently works with VMInformant events only. VMInformant events were modified to include the event unique ID according to the taxonomy discussed in Chapter 4. This is carried out before sending the event to the provenance log, and before the log is shipped by *Filebeat*. The analyser checks the ID of the event and it can then produce statistics and apply some of the concepts discussed earlier in this chapter, including determining the severity of the events. Since *Logstash* and *Kibana* can assist in performing similar operations effectively, we relied mainly on the output generated

[17]https://github.com/a2o/snoopy

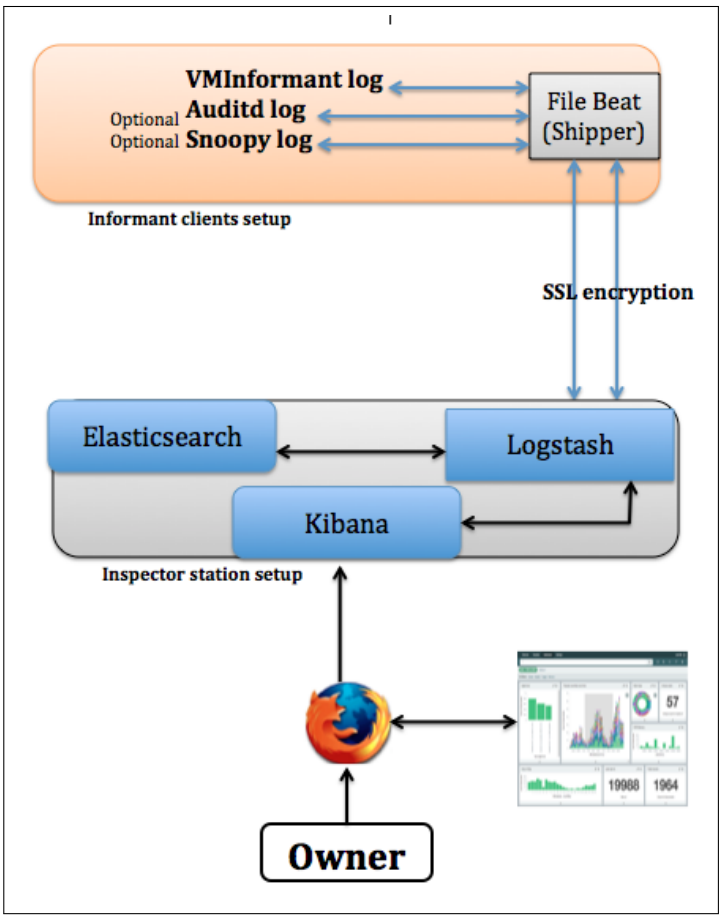

Fig. 6.9 How all the tools are integrated together

in *Kibana*.

6.7.2 Implementation of the experimental scenarios

In general, most of the experimental scenarios that we implemented use the following bash script to determine which VM to apply the scenario to (randomly), and also to determine the waiting time (if any) before the scenario process is launched:

Listing 6.1 Deciding the random waiting time and the VM that will be affected in the experiment

```
#!/bin/bash
waitingtime=$((1 + RANDOM % 1000))
randomid=$((1 + RANDOM % 3))
case $randomid in
  1)
    chosen=informant-client1
    ;;
  2)
    chosen=informant-client2
    ;;
  3)
    chosen=informant-client3
    ;;
 esac

echo $chosen
echo $waitingtime
sleep $waitingtime
```

6.7.3 Implementing scenario 1 (manipulation of files scenario)

In this scenario we configure VMInformant to monitor a certain folder in /home directory in all the monitored VMs. The folder name is *credit-cards*. It contains confidential archived data that we assume that no employee should access or manipulate. Our controller script selects the affected VM randomly and then injects the manipulation script to the chosen VM. The manipulation script is a sequence of operations to view,

alter, and delete files. The controller script looks like:

Listing 6.2 Injecting a script to the chosen VM and executing it

```
gcloud compute copy-files /users/taimur/Desktop/manipulation
    $chosen:/home/taimur/ --zone us-east1-d
sleep 5
gcloud compute ssh $chosen --zone us-east1-d --command="cd /home/taimur/ &&
    ./manipulation"
echo "Job is done"
```

Commands to execute the manipulation script are sent in the *–command* attribute of the *glcoud* ssh tool.

6.7.4 Results: Scenario 1

Through the inspector station, the owner can observe malicious activities on the *informant- client1* machine (which was chosen by the algorithm). Based on that, an action can be chosen and the matter can be investigated further. Figure 6.10 shows how the events observed on informant-client1 are represented in the Kibana web interface.

Fig. 6.10 Manipulation events observed on kibana

6.7.5 Implementing Scenario 2 (malicious script scenario)

In this scenario we inject two malicious scripts into the chosen VM. Both scripts perform different operations. The first script generates a large number of files of variable sizes in the selected VM. The other script copies all the contents of a selected folder to a different location – possibly to send the content later via email. The scripts can be found in Appendix A.

In this scenario we are interested in the detection of a malicious process. The Linux auditd system has the ability to monitor all system calls or to focus on specific types of system calls. As a proof of concept, we configured *auditd* to monitor all executed processes. This is performed by adding a rule to the auditd to monitor the *execve* system call. Thus, the owner will get a list of all the executed processes in the VM. Based on filters, malicious processes can be identified and events can be checked. Process-related files can trigger file-centric events as well.

6.7.6 Results: Scenario 2

As expected, the first script resulted in heavy I/O processes in the chosen VM. This was also reflected in the inspector station, as a huge number of events were received. Since the script must have been executed using the Linux shell, filtering the output of snoopy or auditd to display the "bash" command revealed the event and the event details, including the user who started the process. The owner/administrator can utilise these observations to make informed trust decisions. As mentioned before, the policy centre contains information about the processes authorised to run in the monitored VMs. This will enable detection of unknown processes.

6.7.7 Implementing Scenario 3 (migrated instance scenario)

As in the previous scenarios, a monitored VM is selected randomly (in order to be migrated). All the monitored VMs interact with three light-weight monitoring VMs (explained in Section 6.6.2). This means that the monitored VM and the three VMs operate in a virtual cluster. All the VMs are hosted in the Google Cloud Platform in region: *us-east*, zone: *us-east1-d*. First, *normal profiles* of the VMs were generated and stored in the inspector station.The migration process of one of the monitored VMs to *us-east1-c*, which is a zone in the same region, is automated. This takes place after a random period of time. Then, periodic profiles are generated and checked against the normal profiles. The same procedure will be repeated, but the migration is to be

Table 6.1 Results of the migrating one of the VMs randomly

Procedure	**ICMP change**	**External IP change**	**Hypervisor Change**	**Processor Information Change**
migrating the chosen instance to us-east1-c	no	no	no	no
migrating the chosen instance to asia-northeast1-c	yes	yes	no	yes

automated to another region: asia-northeast1-c. The following sample command will migrate the VM from one zone to another, or to another region:

```
gcloud compute instances move $chosen --zone us-east1-d --destination-zone
    us-east1-c
```

The sample script used to generate the normal/periodic profile inside the VMs is available in Appendix: A. The script was configured to restart if the machine reboots. A number of tools were used to assist in generating the normal/periodic profiles, including: *virt-what* (to get the hypervisor type) and *lscpu* (to get processor-related information).

6.7.8 Results: Scenario 3

Table 6.1 shows the result of invoking the migration process and comparing periodic profiles with the normal profiles of the VMs. Periodic profiles were sent to the inspector station for analysis. We noticed that when moving the chosen instance from *us-east1-d* to *us-east1-c*, none of the metrics changed. Even the processor information did not change. Going back to the Google platform documentation, we found that all the zones in the Eastern US region have the same processor types. This means that, although a migration occurred, detection was not possible using the metrics we collected. However, migrating within the same region does not usually pose problems, and in this thesis we are concerned mainly with migrations that occur to different regions.

In the second case, migrating to *asia-northeast1-d*, there was a significant ICMP (average) latency change between the monitored VM and the collection of light-weight monitoring VMs, while there was no change in the latency of the monitoring VMs. Also, no changes in the other migration metrics were observed in the monitoring VMs. The external IP address metric (CE) and the processor information metric (CP) were

altered due to the change in region and the processor type respectively. The hypervisor type remained the same. All this indicates a higher probability that a migration to another region has occurred. In the inspector station, an analyser script written in Python was used to compare the periodic profiles coming from different VMs by applying the algorithm explained in Section 6.3.2. Once changes are identified in the profiles, an alarm will be fired. Thresholds to fire the alarm can be stored in the policy centre of the inspector station. When the potential migration event of a particular VM is detected, the owner of the VM can make decisions regarding trusting the cloud provider. Also, if this violates the SLA terms, the owner could claim compensation.

6.8 Evaluation

We present an evaluation of the proposed system. The evaluation spans four aspects: performance, security, storage and usefulness.

6.8.1 Performance

The inspector station is the consumer's investment towards improving the security of critical VMs running sensitive services in the IaaS public cloud. The applications deployed in an inspector station VM, are all related to monitoring and analysis tasks. Thus, this will not affect the critical applications running the monitored VM themselves. The number of processed events depends on the number of VMs and the frequency frequency at which events are received. Since the expected incoming events are all security-related (already filtered) based on the suggested taxonomy, we argue that the likelihood of receiving a unusually high number of events at once is fairly low. Further, Processing may be required to: (1) find patterns in the events which occur in multiple VMs; (2) detect life-cycle events, e.g. migration; in which case this will require comparing VM profiles by performing analysis to them; (3) generate reports based on the consumer's queries. All these tasks depend on the number of events to be processed in the inspector station. As mentioned before, a full dedicated event management system which reflects the architecture of the *Inspector station* was not implemented. Rather, we made use of some open source tools to aggregate/analyse events centrally in the Inspector Station. Hence, a comprehensive evaluation of the proposed architecture is set as one of the future goals. However, we implemented a script, using python, which runs in the inspector station to generate useful statistics. It accepts, as input, datasets in a certain format containing events which belong to

Table 6.2 Execution time of the script according to the different event datasets

Data Sets	Number of VMs	Number of Events	Execution time (seconds)
Data set 1	15	1155	0.01
Data set 2	30	1889	0.02
Data set 3	55	4312	0.04
Data set 4	200	14937	0.13
Data set 5	500	38407	0.45
Data set 6	1000	75438	0.85

multiple VMs. From the dataset, the script generates statistics that include (but are not limited to): number of events, number of events per VM, number of events of a certain event group according to taxonomy, severity of the events, VMs which have the highest number of severe events, etc. We wanted to check the effect of the increased number of processed events on our *analyser* script. Therefore, as a proof of concept, we prepared 6 datasets containing random number of events belonging to a different number of VMs (15, 30, 55, 200, 500, 1000). Severity of the events has been configured separately and the analyser script is able to fetch the severity for every event. The script was run on a VM (Standard 2.3 GHz Intel Xeon, 3.75 GB RAM) hosted in the Google Cloud Platform (GCP). Execution time of the script, based on the 6 events datasets, is shown in Table 6.2. From the table, the execution time is proportional to the number of events. Analysing more than 75000 events took less than a second. However, this performance evaluation is for this specific script, which performs limited tasks and does not involve correlating the security-related events to find patterns. If the script is to find patterns and correlate events across multiple VMs, then the performance of script may be affected differently.

6.8.2 Security

Security of systems is commonly evaluated based on the CIA Triad (**C**onfidentiality, **I**ntegrity and **A**vailability). As discussed throughout the chapter, events data is collected from multiple VMs and stored in the Inspector Station. This means we have to ensure a secure communication link *between* the VMs and the inspector station (in transit). Also, we have to secure the events *in* the inspector station (at rest). In our use of the open source tools, Logstash was used to receive and process the events before sending to Kibana (see Section 6.7). Logstash allows using SSL certificates to secure the communications between the *sending* VMs and the *receiving* inspector station. This way, data in transit/motion will be secure. When the event data are at rest (In

Table 6.3 The set of suggested controls/features and how they map to the CIA triad

	The concepts in CIA addressed by the control		
Feature/ Control	***C***	***I***	***A***
Securing data in the Inspector Station	✓		
Securing the communication between monitored VMs and the Inspector Station	✓	✓	
Providing fault tolerance (e.g. Peer to Peer Inspector Stations)			✓
Ensuring event data is not tampered with (e.g. checking hash value)	✓	✓	
Specifying the authorised users to use the monitored VMs and the inspector station VM	✓		

the Inspector Station), encryption can be used (when the data is not processed). To check that events have not been tampered with, hash values can be checked; thereby preserving the integrity of the data. To ensure that the system is available when faults happen, a Peer to Peer architecture can be utilised which allows the system to run across multiple Inspector Station machines. Table 6.3 lists some of the suggested features/controls and shows how they are mapped to the CIA Triad.

6.8.3 Storage

Events are sent to the event centre in the inspector station and stored in the appropriate format. Therefore, depending on the volume of stored events, the impact of storage may be determined. As mentioned before, the Policy Centre in the Inspector Station architecture contains parameter values to govern the retention period of events, so the impact of storage may also be limited. In the case where a peer to peer (P2P) architecture is considered, storage of the events could be divided among several VMs (Inspector Stations). In general, if we assume that an event record needs 0.57 KB (worst case), then a machine with 10 GB of storage will be able to store more than 18 million events. Since the events that we are targeting are focused security-related events, the storage is not an issue when we consider events only. The impact on storage is not caused only by the number of events. Storing reports and queries generated by the consumer may consume storage, unless they were stored outside the inspector station. Configuration files on the policy centre constitute part of the storage but the impact is likely to be minimal; as their volume is not likely to increase.

6.8.4 Usefulness

The architecture supports the trust between the consumer and the provider and between the consumer and his employees. By aggregating and analysing events from multiple VMs belonging to the same consumer, we argue that the consumer will be better informed about what is happening to their VMs hosting critical applications. Knowledge of the overall security status of the monitored VMs help consumers draw conclusions about the health status of the VMs, and enable them to react to incidents when they happen. Finding patterns in security-related events across multiple VMs may reveal information which can save time and help consumers implement countermeasures/controls to safeguard the data. The architecture can help consumers check the adherence to SLA terms and preserve consumer's right to claim in the case of incidents, e.g. migrating VMs to a different region. The reporting facilities that the inspector station provides can help draw conclusions in a simple and informative way, and without the jargon of unnecessary data that may require experienced technical staff to understand.

6.9 Limitations

Since the inspector station will mostly be running in a VM in the IaaS public cloud, it will naturally be susceptible to all the issues that we are trying to overcome in this work: including being tampered with, data disclosure, data destruction, DoS attack, etc. While this cannot be prevented absolutely, a number of solutions may be available, such as: (1) encrypting the contents; (2) monitoring specific file locations; (3) and using multiple inter-related inspector station machines to provide fault tolerance.

Some of the attributes that govern the decision making are based on choices made by the consumer, such as: importance of metrics, priority of the event, and severity of the event. The question is, how accurately can the consumer determine the value of these attributes? The decision is greatly dependent on the attributes. In addition, they are essential for determining the VM health and forming a trust profile. We argue that the values of these attributes must be chosen carefully.

6.10 Chapter Conclusion

Security-related events observed in VMs owned by one consumer may be related, e.g. triggered by the same user or affect same files. In this chapter, we argued that analysing the security-related events which occur across multiple VMs and finding patterns between them could give the consumer greater confidence on the overall security of their hosted VMs. We conclude that there are many tools which offer the aggregation of events from multiple VMs. Yet, there are a lack of tools which can find patterns and correlate secuirty-related events from multiple VMs. We proposed an architecture of a system to aggregate, manage and analyse events received from multiple VMs in public IaaS cloud systems (Inspector Station)–highlighted in Section 6.3. The architecture consists of four main components: Policy centre, Event centre, Report centre and Alert centre. All the components are integrated to together to provide the consumer with analysis, alerting and reporting facilities. We proposed some suggestions for finding patterns in the observed events across multiple VMs. We discussed how finding patterns in the events could enhance the visibility and help the consumer make informed decisions.

Based on the importance and impact of the events, from the perspective of the consumer, we suggested the metrics: *priority* and *severity*. We indicated how they can be used to trigger alerts and form what we called: *VM Health Profile* and *Trust Profile*. We believe that the concepts of the Trust Profile can support the trust between the consumer and the provider; provided that the observed events have been identified to be triggered by the provider.

The architecture was partially implemented and evaluated using our prototype implementation of *VMInformant* and selected open source tools such as: *Filebeat* (to ship the events), *Logstash* (to process the events) and *Kebana* (to report and visualize the events). We have indicated the usefulness of the architecture by illustrating three scenarios: manipulation of files, malicious script and migration of instances. This was experimentally illustrated by utilising VMs hosted on Google Cloud Platform (GCP). Events of multiple VMs were aggregated in the Inspector Machine in real time, which gives the consumer visibility on what is happening to their hosted VMs. Even though a dedicated system to find patterns in the events from multiple VMs could not be fully implemented, we believe that the suggestions we provided to find patterns and correlate the events, could be the basis for developing an analysis tool, which is able to

give the consumer more useful information.

Finally, we evaluated the proposed architecture based on three aspects: performance, security, storage and usefulness. We also highlighted few potential limitations associated with it.

Chapter 7

Conclusions & Future Work

7.1 Chapter Overview

This chapter summarizes the thesis and recites the research contribution. The chapter goes on to highlight some future work related to the thesis.

7.2 Thesis Summary and Contributions

While public IaaS cloud systems have proved useful and beneficial for many organisations, many of these organisations are still reluctant to host their data in the cloud. This is mainly due to the various security/privacy concerns associated with the cloud computing model. The shared tenancy feature of cloud systems meant that VMs of different consumers might reside in the same physical machine, separated logically by the hypervisor. Attacking the hypervisor would mean attacking all the VMs under it. By studying the literature of cloud security, we have come to realize that compromising the virtualisation platform is possible due to many security issues, which were outlined and discussed in detail in Chapter 3. Cloud security challenges combined with the full control the cloud provider has on the public cloud infrastructure, both affected the trust between the consumer and the provider. The consumer is not sure how his data is accessed or processed, and where it is exactly located. According to our initial study of the literature, we found that the trust could be supported by monitoring and recording security-related events which occur in the VM, and reporting them to the consumers. Hence, we hypothesized that:

"If security/privacy events happening inside virtual machines (VMs), hosted in the IaaS cloud infrastructure, are monitored and recorded, a consumer can have greater

confidence in what is happening to their VMs. This enhances the level of trust in a cloud provider, and improves the level of security of a consumer application"

To examine this research hypothesis and address the aim/objectives highlighted in Section 1.3, a number of approaches were used. The main aim of this thesis has been to monitor security-related events for the purpose of supporting trustworthy cloud computing, i.e. so the consumer can have a greater confidence in what is happening to their VMs which are hosted in the cloud. Thus, to achieve this aim, we first tried to identify the VM security-events of interest to us. This was mainly to provide a focused security monitoring; especially given that VMs have limited resources and monitoring should not affect normal operations of these VMs as much. Therefore, we proposed a taxonomy of security-related VM events which included six groups of events: *file-centric, life-cycle, network-related, process-related, user-access, attached-devices*. This taxonomy was highlighted in Section 4.5 and it marks our first contribution. To detect and monitor VM security events, we proposed the design and architecture of a system to monitor, analyse and report the events to the consumer–called: *VMInformant*, highlighted in Section 4.7. The architecture allows launching monitors for the events according to the need of the consumer, by making use of the proposed taxonomy. The events are processed locally before they are recorded in a log file and sent to the consumer. A prototype implementation of the *VMInformant* architecture was provided whereby some selected VM security events could be monitored and recorded in an informative way. By monitoring, the consumer is able to learn about some of the operations which occur on his hosted data. Hence, the level of trust can be enhanced. To learn about the performance and storage overhead resulting out of monitoring VM security events, an experimental evaluation was carried out where a number of I/O and CPU intensive benchmarks were used, e.g. *Povray* and *Bzip2*–covered in Section 5.10.8. The results indicated the overhead of monitoring can be tolerable, and can be reduced by an improved code or a better storage medium. By coming up with taxonomy of security-related VM events and the architecture of VMInformant with its prototype implementation/evaluation, we were able to address two objectives. The first one is related to the types of security-related VM events which can be monitored from a consumer's perspective, while the second one is concerned with achieving a focused monitoring (technically) from the perspective of the consumer.

We surveyed a number of techniques which can assist in detecting the migration of VMs in public cloud systems. We noticed that some of the techniques are suitable for detecting the migration at various stages: just before it happens, while it is happening

and after it has happened already. Some of them are more suitable for detecting the migration when it happens within the data center while others are suitable for detecting it when it happens outside the data center. Thus, we proposed a spatio-temporal taxonomy for the migration detection techniques to help achieving detection with minimized performance and storage overheads– Section 5.7. This is because some techniques require that the recording of metrics is performed more frequently inside the VM. Then we evaluated one of the VM migration detection techniques (the virtual landmark fingerprint) by utilizing VMs hosted in Amazon Web Services (AWS). The technique relies on the use of ICMP to measure the latency between the VM and a set of selected general internet landmarks. The evaluation showed that this technique may give a good estimate that migration of VM has happened. However, it may result in inaccuracies due to various network issues. Therefore, we proposed another hybrid approach which combines the use of multiple migration detection techniques (referred to as: *migration metrics*), to estimate the likelihood of the migration event. The approach considers the importance of the metrics from the perspective of the cloud consumer and uses a weighted decision function to generate the probability of the migration event. To the best of our knowledge, our combined approach is the first, which considers aggregating migration metrics in a decision function. By using the combined approach, we argued that it may not be enough to rely only on a single metric in the detection process as this might lead to inaccurate outcomes.

An experimental evaluation of our approach was carried out on VMs hosted on the Google Cloud Platform (GCP) where we considered four migration metrics: ICMP latency, External/Public IP, hypervisor type and processor information. Few light-weight VMs were used track the VMs. They maintain ICMP connections between the monitored VM and between the other light-weight monitoring VMs. Investigating the interactions between the VMs and observing the state of the *migration metrics* after the migration, indicated that our approach is suitable– This is highlighted in Section 5.10. The consumer can have a greater assurance that their VMs have not been migrated. We argued that detecting VM migration can help with checking the adherence to SLA and to protect the data from being compromised in regions with differing data access regulations.

We noticed that some of the events observed in VMs owned by the consumer may be related, e.g. triggered by the same user. Analysing the security-related events which occur across multiple VMs and finding patterns between them could give the consumer greater confidence on the overall security of their hosted VMs. Therefore, we

proposed an architecture of a system to aggregate, manage and analyse events received from multiple VMs in public IaaS cloud systems. We called the system: *Inspector Station*–highlighted in Section 6.3. The architecture consists of four main components: the policy center, the event center, the report center and the alert center. All the components are integrated to together to provide the consumer with analysis, alerting and reporting facilities. We proposed some suggestions for finding patterns in the observed events across multiple VMs. The architecture was partially implemented and evaluated using our prototype implementation of VMInformant and some open source tools such as: *Filebeat* (to ship the events), *Logstash* (to process the events) and *Kebana* (to report and visualize the events). Due to the time constraint, a dedicated system to find patterns in the events from multiple VMs could not be fully implemented. However, we believe that the suggestions we provided to find patterns and correlate the events, could be the basis for developing an analysis tool, which is able to give the consumer more useful information.

By following the research methodology described above, we examined the hypothesis to the point where we can say with some confidence that it holds true. Monitoring security-related VM events can give the consumer an improved visibility on what happens to their VMs hosted in public IaaS cloud systems. Consequently, this could support the trust between the provider and the consumer.
The contributions in this thesis can be-reiterated as follows:

1. Detecting security-related VM events:

    - A taxonomy of security-related VM events.
    - Design, architecture, prototype implementation of a system to detect and monitor security-related VM events (*VMInformant*).
    - Design and architecture of a system to aggregate, manage, analyse and correlate security-related events observed from multiple VMs (*Inspector Station*).
    - An illustration how some of the state-of-the-art open source tools can be used to aggregate events from multiple VMs in real time and represent them.

2. Detecting VM migrations:

    - A spatio-temporal taxonomy of migration detection techniques.

- A prototype implementation of a tool to detect migration of VMs in IaaS using an approach called: Virtual Remote Landmark Fingerprint.
- An algorithm to detect VM migrations in public IaaS cloud systems using a combination of *migration metrics*.

3. Experimental evaluation:

   - A detailed evaluation of monitoring and detection of security-related VM events, in terms of performance, storage and usefulness.
   - An evaluation of migration detection techniques by experimenting with various scenarios in common public IaaS cloud systems, e.g. Amazon Web Services (AWS) and Google Cloud Platform.

4. A literature survey of cloud security challenges, monitoring in the cloud and VM migration detection.

7.3 Future Work

- **Identifying and detecting VM-specific malicious processes:** Since VMs run full operating systems, malware can exploit vulnerabilities found in them to run malicious processes. In VMs running critical applications, this could cause damage. It will be useful for the consumer to detect the occurrence of such an event as soon as it happens in order to react. One possible way of doing this is by inspecting the memory and disk to analyse system calls. Some of the malicious processes may use unusual system calls to perform operations. If some specific system calls were encountered, a proper monitoring system should inform the consumer about it. For that, a taxonomy of potential harmful system calls in the medium of VMs have to be developed. The detection of malicious processes should not affect the running applications of the critical VMs nor should it consume storage.

- **Exploring how security-related events can be monitored in containers:** As discussed earlier in Sections 4.14 and 5.11, this area needs further research.

- **Implementing an autonomic monitoring system which monitors according to the needs:** Due to the limited resources of VMs, excessive monitoring may affect the performance of the running applications as well as consuming

storage. In this thesis, we have evaluated the monitoring process so that we monitor useful events and send them to the consumer, while at the same time we do not affect the performance or storage much. A question that can be asked is: is it possible to control the level of monitoring (granularity) in an autonomic way, so the monitoring system can monitor more or less according to some variable input? Also, what would these variables be? We argue that building this intelligent logic into the monitoring system could save resources and reduce the need for humans to act or decide or manage things, especially if the architecture of the Inspector station(s) was considered– (Chapter 6).

References

[1] Imad M. Abbadi. A framework for establishing trust in cloud provenance. 12 (2):111–128, April 2013. ISSN 1615-5262. doi: 10.1007/s10207-012-0179-0. URL http://dx.doi.org/10.1007/s10207-012-0179-0.

[2] Imad M Abbadi, John Lyle, et al. Challenges for provenance in cloud computing. In *3rd USENIX Workshop on the Theory and Practice of Provenance (TaPP'11). USENIX Association*, 2011.

[3] Taimur Al-Said and Omer Rana. Implementing migration-aware virtual machines. In *2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing (CSCloud)*, pages 54–61, Nov 2015. doi: 10.1109/CSCloud.2015.92.

[4] Taimur Al Said and Omer F. Rana. Analysing virtual machine security in cloud systems. In *Intelligent Cloud Computing: First International Conference, ICC 2014, Muscat, Oman, February 24-26, 2014, Revised Selected Papers*, pages 137–151. Springer International Publishing, 2014. ISBN 978-3-319-19848-4. doi: 10.1007/978-3-319-19848-4_9. URL http://dx.doi.org/10.1007/978-3-319-19848-4_9.

[5] Taimur Al-Said, Omer Rana, and Peter Burnap. VMInformant: an instrumented virtual machine to support trusthworthy cloud computing. *International Journal of High Performance Computing and Networking (IJHPCN)*, 41(8):13–15, 2015. doi: 10.1504/IJHPCN.2015.071257.

[6] Suaad Alarifi and Stephen Wolthusen. Dynamic parameter reconnaissance for stealthy dos attack within cloud systems. In *IFIP International Conference on Communications and Multimedia Security*, pages 73–85. Springer, 2014.

[7] Suaad S Alarifi. *Attacks on Cloud Environments and their Mitigation*. PhD thesis, Royal Holloway, University of London, 2015. URL https://pure.royalholloway.ac.uk/portal/files/25492867/Suaad_Alarifi_PhD_Thesis.pdf. Supervised by Dr. Stephen Wolthusen.

[8] Suaad S Alarifi and Stephen D Wolthusen. Detecting anomalies in iaas environments through virtual machine host system call analysis. In *Internet Technology And Secured Transactions, 2012 International Conferece For*, pages 211–218. IEEE, 2012.

[9] Pavel Alpeyev, Joseph Galante, and Mariko Yasu. Amazon. com server said to have been used in sony attack. *Bloomberg, May*, 14,

2011. URL http://www.bloomberg.com/news/articles/2011-05-13/sony-network-said-to-have-been-invaded-by-hackers-using-amazon-com-server. Accessed: 2013-04-23.

[10] MR Anala, Jyoti Shetty, and G Shobha. A framework for secure live migration of virtual machines. In *Advances in Computing, Communications and Informatics (ICACCI), 2013 International Conference on*, pages 243–248. IEEE, 2013.

[11] R Anand, S Sarswathi, and R Regan. Security issues in virtualization environment. In *Radar, Communication and Computing (ICRCC), 2012 International Conference on*, pages 254–256. IEEE, 2012.

[12] Vinod Anupam and Alain J Mayer. Security of web browser scripting languages: Vulnerabilities, attacks, and remedies. In *USENIX Security*, 1998.

[13] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010. ISSN 0001-0782. doi: 10.1145/1721654.1721672. URL http://doi.acm.org/10.1145/1721654.1721672.

[14] Django Armstrong, Daniel Espling, Johan Tordsson, Karim Djemame, and Erik Elmroth. Runtime virtual machine recontextualization for clouds. In *Euro-Par 2012: Parallel Processing Workshops*, pages 567–576. Springer, 2013. doi: 10.1007/978-3-642-36949-0_66. URL http://dx.doi.org/10.1007/978-3-642-36949-0_66.

[15] AWS. Netflix case study. https://aws.amazon.com/solutions/case-studies/netflix/. Accessed: 2016-04-24.

[16] Fabrizio Baiardi and Daniele Sgandurra. Building trustworthy intrusion detection through vm introspection. In *Information Assurance and Security, 2007. IAS 2007. Third International Symposium on*, pages 209–214. IEEE, 2007.

[17] Marco Balduzzi, Jonas Zaddach, Davide Balzarotti, Engin Kirda, and Sergio Loureiro. A security analysis of amazon's elastic compute cloud service. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, pages 1427–1434, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-0857-1. doi: 10.1145/2245276.2232005. URL http://doi.acm.org/10.1145/2245276.2232005.

[18] Mircea Bardac, Răzvan Deaconescu, and Adina Magda Florea. Scaling peer-to-peer testing using linux containers. In *9th RoEduNet IEEE International Conference*, pages 287–292. IEEE, 2010.

[19] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164–177, October 2003. ISSN 0163-5980. doi: 10.1145/1165389.945462. URL http://doi.acm.org/10.1145/1165389.945462.

[20] Masooda N Bashir, Jay P Kesan, Carol M Hayes, and Robert Zielinski. Privacy in the cloud: going beyond the contractarian paradigm. In *Proceedings of the 2011 workshop on governance of technology, information, and policies*, pages 21–27. ACM, 2011.

[21] Adam Bates, Benjamin Mood, Joe Pletcher, Hannah Pruse, Masoud Valafar, and Kevin Butler. Detecting co-residency with active traffic analysis techniques. In *Proceedings of the 2012 ACM Workshop on Cloud computing security workshop*, pages 1–12. ACM, 2012.

[22] Chris Benninger, Stephen W Neville, Yagiz Onat Yazir, Chris Matthews, and Yvonne Coady. Maitland: Lighter-weight vm introspection to support cyber-security in the cloud. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 471–478. IEEE, 2012.

[23] Sushil Bhardwaj, Leena Jain, and Sandeep Jain. Cloud computing: A study of infrastructure as a service (iaas). *International Journal of engineering and information Technology*, 2(1):60–63, 2010.

[24] Sebastian Biedermann, Martin Zittel, and Stefan Katzenbeisser. Improving security of virtual machines during live migrations. In *Privacy, Security and Trust (PST), 2013 Eleventh Annual International Conference on*, pages 352–357. IEEE, 2013. doi: 10.1109/PST.2013.6596088.

[25] Matthias Bolte, Michael Sievers, Georg Birkenheuer, Oliver Niehörster, and André Brinkmann. Non-intrusive virtualization management using libvirt. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 574–579. European Design and Automation Association, 2010.

[26] Rich Bourdeau. Controlling vm sprawl: How to better utilize your virtual infrastructure. 2010. URL https://www.dabcc.com/resources/controlling-vm-sprawl.pdf. Accessed: 2016-12-20.

[27] Robert Bradford, Evangelos Kotsovinos, Anja Feldmann, and Harald Schiöberg. Live wide-area migration of virtual machines including local persistent state. In *Proceedings of the 3rd international conference on Virtual execution environments*, pages 169–179. ACM, 2007.

[28] Timothy Broomhead, Laurence Cremean, Julien Ridoux, and Darryl Veitch. Virtualize everything but time. In *OSDI*, volume 10, pages 1–6, 2010.

[29] Sven Bugiel, Stefan Nürnberger, Thomas Pöppelmann, Ahmad-Reza Sadeghi, and Thomas Schneider. Amazonia: when elasticity snaps back. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 389–400. ACM, 2011.

[30] Daniele Catteddu and Giles Hogben. Cloud computing risk assessment. *European Network and Information Security Agency (ENISA)*, pages 583–592, 2009. URL https://www.enisa.europa.eu/publications/cloud-computing-risk-assessment. Accessed: 2015-04-03.

[31] Shirlei Chaves, Carlos Westphall, Carla Westphall, and Guilherme Geronimo. Customer security concerns in cloud computing. In *ICN 2011, The Tenth International Conference on Networks*, pages 7–11, 2011.

[32] Shengdong Chen, Yang Zhang, and Jun-Liang Chen. Malware process detecting via hypervisor. In *Preceeding of the International Conference on Electrical, Control and Automation,ECAE*, 2013.

[33] Yanpei Chen and Randy H. Katz. Glimpses of the brave new world for cloud security, 2011. URL http://www.hpcinthecloud.com/hpccloud/2011-02-22/glimpses_of_the_brave_new_world_for_cloud_security.html. Accessed: 2016-12-03.

[34] Yanpei Chen, Vern Paxson, and Randy H Katz. What's new about cloud computing security. *University of California, Berkeley Report No. UCB/EECS-2010-5 January*, 20(2010):2010–5, 2010.

[35] Yao Chen and Radu Sion. On securing untrusted clouds with cryptography. In *Proceedings of the 9th annual ACM workshop on Privacy in the electronic society*, pages 109–114. ACM, 2010.

[36] Richard Chow, Philippe Golle, Markus Jakobsson, Elaine Shi, Jessica Staddon, Ryusuke Masuoka, and Jesus Molina. Controlling data in the cloud: Outsourcing computation without outsourcing control. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, CCSW '09, pages 85–90, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-784-4. doi: 10.1145/1655008.1655020. URL http://doi.acm.org/10.1145/1655008.1655020.

[37] Mihai Christodorescu, Reiner Sailer, Douglas Lee Schales, Daniele Sgandurra, and Diego Zamboni. Cloud security is not (just) visualization security: a short paper. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 97–102. ACM, 2009.

[38] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 273–286. USENIX Association, 2005.

[39] Dino Dai Zovi. Hardware virtualization rootkits. *BlackHat Briefings USA*, 2006. URL https://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Zovi.pdf. Accessed: 2016-10-13.

[40] Shirlei Aparecida de Chaves, Carlos Becker Westphall, and Flavio Rodrigo Lamin. Sla perspective in security management for cloud computing. In *Networking and Services (ICNS), 2010 Sixth International Conference on*, pages 212–217. IEEE, 2010.

[41] Mamadou H Diallo, Michael August, Roger Hallman, Megan Kline, Henry Au, and Vic Beach. Nomad: A framework for developing mission-critical cloud-based applications. In *Availability, Reliability and Security (ARES), 2015 10th International Conference on*, pages 660–669. IEEE, 2015.

[42] Paul Ducklin. Openssl website defacement. http://nakedsecurity.sophos.com/2014/01/03/openssl-website-defacement-it-wasnt-a-hypervisor-hack-after-all/, 2014. Accessed: 2016-12-20.

[43] Vinod Durairaj and Vasudevu Lakhinana. Finding shortest path in multi-access nodes in cloud service, January 3 2017. US Patent 9,537,748.

[44] Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. An updated performance comparison of virtual machines and linux containers. In *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On*, pages 171–172. IEEE, 2015.

[45] Peter Ferrie. Attacks on more virtual machine emulators. *Symantec Technology Exchange*, 2007. URL http://www.symantec.com/avcenter/reference/Virtual_Machine_Threats.pdf.

[46] Sebastian Fiebig, Melanie Siebenhaar, Christian Gottron, and Ralf Steinmetz. Detecting vm live migration using a hybrid external approach. In *CLOSER*, pages 483–488, 2013.

[47] Yangchun Fu and Zhiqiang Lin. Space traveling across vm: Automatically bridging the semantic gap in virtual machine introspection via online kernel data redirection. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 586–600. IEEE, 2012.

[48] Lee Garber. The challenges of securing the virtualized environment. *Computer*, 45(1):17–20, 2012. ISSN 0018-9162. doi: 10.1109/MC.2012.27.

[49] Pedro Garcia-Teodoro, J Diaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, 28(1):18–28, 2009.

[50] Tal Garfinkel, Mendel Rosenblum, et al. A virtual machine introspection based architecture for intrusion detection. In *NDSS*, 2003.

[51] Tal Garfinkel, Keith Adams, Andrew Warfield, and Jason Franklin. Compatibility is not transparency: Vmm detection myths and realities. In *HotOS*, 2007.

[52] Gartner. Why a no-cloud policy will become extinct. http://www.gartner.com/smarterwithgartner/cloud-computing-predicts/, 2016. Accessed: 2016-12-21.

[53] Gartner. Gartner says worldwide public cloud services market is forecast to reach 204 billion dollars in 2016. http://www.gartner.com/newsroom/id/3188817, 2016. Accessed: 2016-12-21.

[54] Ashish Gehani, Gabriela F Ciocarlie, and Natarajan Shankar. Accountable clouds. In *Technologies for Homeland Security (HST), 2013 IEEE International Conference on*, pages 403–407. IEEE, 2013.

[55] Craig Gentry. A fully homomorphic encryption scheme [ph.d. thesis]. *International Journal of Distributed Sensor Networks, Stanford University*, 2009.

[56] Robert P Goldberg and Peter S Mager. Virtual machine technology: A bridge from large mainframes to networks of small computers. In *Compcon Fall 79. Proceedings*, pages 210–213. IEEE, 1979.

[57] Google. Transparent maintenance. https://cloud.google.com/compute/docs/zones, 2015. Accessed: 2016-02-26.

[58] Google. Google compute engine uses live migration technology to service infrastructure without application downtime. http://goo.gl/VUM0Q6, 2015. Accessed: 2016-02-26.

[59] Christian Gottron, Sebastian Fiebig, André König, Andreas Reinhardt, and Ralf Steinmetz. Visualizing the migration process of virtual machines. *Proceedings of the 12th Euroview*, 2012.

[60] Top Threats Working Group et al. The notorious nine: cloud computing top threats in 2013. *Cloud Security Alliance*, 2013. URL https://downloads.cloudsecurityalliance.org/initiatives/top_threats/The_Notorious_Nine_Cloud_Computing_Top_Threats_in_2013.pdf.

[61] Irfan Gul and M Hussain. Distributed cloud intrusion detection model. *International Journal of Advanced Science and Technology*, 34(38):135, 2011.

[62] Udit Gupta. Comparison between security majors in virtual machine and linux containers. *CoRR*, abs/1507.07816, 2015. URL http://arxiv.org/abs/1507.07816.

[63] U Gurav and R Shaikh. Virtualization: a key feature of cloud computing. In *Proceedings of the International Conference and Workshop on Emerging Trends in Technology*, pages 227–229. ACM, 2010.

[64] Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg. Side channels in cloud services: Deduplication in cloud storage. *Security & Privacy, IEEE*, 8(6): 40–47, 2010.

[65] Keiko Hashizume, David G Rosado, Eduardo Fernández-Medina, and Eduardo B Fernandez. An analysis of security issues for cloud computing. *Journal of Internet Services and Applications*, 4(1):1, 2013.

[66] Michael R Hines, Umesh Deshpande, and Kartik Gopalan. Post-copy live migration of virtual machines. *ACM SIGOPS operating systems review*, 43(3):14–26, 2009.

[67] Jennia Hizver and Tzi-cker Chiueh. Automated discovery of credit card data flow for pci dss compliance. In *Reliable Distributed Systems (SRDS), 2011 30th IEEE Symposium on*, pages 51–58. IEEE, 2011.

[68] Jennia Hizver and Tzi-cker Chiueh. Real-time deep virtual machine introspection and its applications. In *Proceedings of the 10th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 3–14. ACM, 2014.

[69] Judith Hurwitz, Robin Bloor, Marcia Kaufman, and Fern Halper. *Cloud computing for dummies*, volume 1. John Wiley & Sons, 2009.

[70] Jing-Jang Hwang, Hung-Kai Chuang, Yi-Chang Hsu, and Chien-Hsing Wu. A business model for cloud computing based on a separate encryption and decryption service. In *2011 International Conference on Information Science and Applications*, pages 1–7. IEEE, 2011.

[71] Kai Hwang, Jack Dongarra, and Geoffrey C Fox. *Distributed and cloud computing: from parallel processing to the internet of things*. Morgan Kaufmann, 2013.

[72] Fujitsu Research Institute. Personal data in the cloud: A global survey of consumer attitudes. *Fujitsu Review*, 2010. URL http://www.fujitsu.com/downloads/SOL/fai/reports/fujitsu/personal-data-in-the-cloud.pdf.

[73] Iulia Ion, Niharika Sachdeva, Ponnurangam Kumaraguru, and Srdjan Čapkun. Home is safer than the cloud!: privacy concerns for consumer cloud storage. In *Proceedings of the Seventh Symposium on Usable Privacy and Security*, page 13. ACM, 2011.

[74] Amarnath Jasti, Payal Shah, Rajeev Nagaraj, and Ravi Pendse. Security in multi-tenancy cloud. In *Security Technology (ICCST), 2010 IEEE International Carnahan Conference on*, pages 35–41. IEEE, 2010.

[75] Meiko Jensen, Jörg Schwenk, Nils Gruschka, and Luigi Lo Iacono. On technical security issues in cloud computing. In *Cloud Computing, 2009. CLOUD'09. IEEE International Conference on*, pages 109–116. IEEE, 2009.

[76] Michael Jordon and James Forshaw. Dirty disks raised new questions about cloud security. http://contextis.co.uk/research/blog/dirty-disks-raise-new-questions-about-cloud-securi/, 2012. URL http://contextis.co.uk/research/blog/dirty-disks-raise-new-questions-about-cloud-securi/. Accessed: 2014-01-05.

[77] Ann Mary Joy. Performance comparison between linux containers and virtual machines. In *Computer Engineering and Applications (ICACEA), 2015 International Conference on Advances in*, pages 342–346. IEEE, 2015.

[78] Muhammad Kazim, Rahat Masood, and Muhammad Awais Shibli. Securing the virtual machine images in cloud computing. In *Proceedings of the 6th International Conference on Security of Information and Networks*, pages 425–428. ACM, 2013.

[79] Muhammad Kazim, Rahat Masood, Muhammad Awais Shibli, and Abdul Ghafoor Abbasi. Security aspects of virtualization in cloud computing. In *Computer Information Systems and Industrial Management*, pages 229–240. Springer, 2013.

[80] Michael Kerrisk. http://man7.org/linux/man-pages/index.html. Accessed: 2015-02-25.

[81] Samuel T King and Peter M Chen. Subvirt: Implementing malware with virtual machines. In *Security and Privacy, 2006 IEEE Symposium on*, pages 14–pp. IEEE, 2006.

[82] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. kvm: the linux virtual machine monitor. In *Proceedings of the Linux symposium*, volume 1, pages 225–230, 2007.

[83] André König and Ralf Steinmetz. Detecting migration of virtual machines. In *Proceedings of the 10th Würzburg Workshop on IP: Joint ITG, ITC, and Euro-NF Workshop Visions of Future Generation Networks (EuroView 2011), Julius-Maximilians-Universität Würzburg, Lehrstuhl für Informatik III*, 2011.

[84] Marcos Laureano, Carlos Maziero, and Edgard Jamhour. Intrusion detection in virtual machine environments. In *Euromicro Conference, 2004. Proceedings. 30th*, pages 520–525. IEEE, 2004.

[85] Jianxin Li, Bo Li, Tianyu Wo, Chunming Hu, Jinpeng Huai, Lu Liu, and KP Lam. Cyberguarder: A virtualization security assurance architecture for green cloud computing. *Future Generation Computer Systems*, 28(2):379–390, 2012.

[86] Jin Li, Qian Wang, Cong Wang, Ning Cao, Kui Ren, and Wenjing Lou. Fuzzy keyword search over encrypted data in cloud computing. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–5. IEEE, 2010.

[87] Yen-Han Li, Yeu-Ruey Tzeng, and Fang Yu. Viso: Characterizing malicious behaviors of virtual machines with unsupervised clustering. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 34–41. IEEE, 2015.

[88] Fang Liu, Jin Tong, Jian Mao, Robert Bohn, John Messina, Lee Badger, and Dawn Leaf. Nist cloud computing reference architecture. *NIST special publication*, 500(2011):292, 2011.

[89] Alert Logic. Cloud security report- 2015. https://www.alertlogic.com/resources/cloud-security-report-2015/, 2015. Accessed: 2016-03-04.

[90] Alina Madalina Lonea, Daniela Elena Popescu, and Huanglory Tianfield. Detecting ddos attacks in cloud computing environment. *International Journal of Computers Communications & Control*, 8(1):70–78, 2013.

[91] Brian Lowans and Neil MacDonald. Tackle six security issues before encrypting data in the cloud. https://www.gartner.com/doc/2364716/tackle-security-issues-encrypting-data, 2013. Accessed: 2014-02-25.

[92] Shengmei Luo, Zhaoji Lin, Xiaohua Chen, Zhuolin Yang, and Jianyong Chen. Virtualization security for cloud computing service. In *Cloud and Service Computing (CSC), 2011 International Conference on*, pages 174–179. IEEE, 2011.

[93] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks: Revealing the secrets of smart cards*, volume 31. Springer Science & Business Media, 2008.

[94] Sean Marston, Zhi Li, Subhajyoti Bandyopadhyay, Juheng Zhang, and Anand Ghalsasi. Cloud computing—the business perspective. *Decision support systems*, 51(1):176–189, 2011.

[95] Leonardo A Martucci, Albin Zuccato, Ben Smeets, Sheikh Mahbub Habib, Thomas Johansson, and Nahid Shahmehri. Privacy, security and trust in cloud computing: The perspective of the telecommunication industry. In *Ubiquitous Intelligence & Computing and 9th International Conference on Autonomic & Trusted Computing (UIC/ATC), 2012 9th International Conference on*, pages 627–632. IEEE, 2012.

[96] Peter Mell and Timothy Grance. The nist definition of cloud computing. *National Institute of Standards and Technology*, 53(6):50, 2009. URL http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf. Accessed: 2016-12-12.

[97] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2, 2014.

[98] Trend Micro. Encryption is not enough for cloud security. http://blog.trendmicro.com/encryption-is-not-enough-for-cloud-security/, 2011. Accessed: 2013-09-25.

[99] Chirag Modi, Dhiren Patel, Bhavesh Borisaniya, Hiren Patel, Avi Patel, and Muttukrishnan Rajarajan. A survey of intrusion detection techniques in cloud. *Journal of Network and Computer Applications*, 36(1):42–57, 2013.

[100] motherboard. Hackers stole account details for over 60 million dropbox users. http://motherboard.vice.com/read/hackers-stole-over-60-million-dropbox-accounts. Accessed: 2016-09-01.

[101] Debajyoti Mukhopadhyay, Gitesh Sonawane, Parth Sarthi Gupta, Sagar Bhavsar, and Vibha Mittal. Enhanced security for cloud storage using file encryption. *CoRR*, abs/1303.7075, 2013. URL http://arxiv.org/abs/1303.7075.

[102] Jon Oberheide, Evan Cooke, and Farnam Jahanian. Exploiting live virtual machine migration. *BlackHat DC Briefings, Washington DC*, 2008. URL https://www.blackhat.com/presentations/bh-dc-08/.../bh-dc-08-oberheide.pdf. Accessed: 2016-11-13.

[103] International Organisation of Standards. *ISO/IEC 27001: Information Technology, Security Techniques, Information Security Management Systems, Requirements*. ISO/IEC, 2005.

[104] International Organisation of Standards. *ISO/IEC 27018:2014, Information technology – security techniques – code of practice for protection of personally identifiable information (PII) in public clouds acting as PII processors*. ISO/IEC, 2014.

[105] International Organisation of Standards. *Information technology – Security techniques – Code of practice for information security controls based on ISO/IEC 27002 for cloud services*. ISO/IEC, 2015.

[106] Anderson Santana de Oliveira, Jakub Sendor, Alexander Garaga, and Kateline Jenatton. Monitoring personal data transfers in the cloud. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, volume 1, pages 347–354. IEEE, 2013.

[107] Bryan D Payne, MDP De Carbone, and Wenke Lee. Secure and flexible monitoring of virtual machines. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pages 385–397. IEEE, 2007.

[108] Michael Pearce, Sherali Zeadally, and Ray Hunt. Virtualization: Issues, security threats, and solutions. *ACM Computing Surveys (CSUR)*, 45(2):17, 2013.

[109] Siani Pearson and Azzedine Benameur. Privacy, security and trust issues arising from cloud computing. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 693–702. IEEE, 2010.

[110] Robert K Perrons and Adam Hems. Cloud computing in the upstream oil & gas industry: A proposed way forward. *Energy Policy*, 56:732–737, 2013.

[111] Krešimir Popović and Željko Hocenski. Cloud computing security issues and challenges. In *Proceedings of the 33rd International Convention. IEEExplore, Opatija*, pages 344–349, 2010.

[112] Thomas Raffetseder, Christopher Kruegel, and Engin Kirda. Detecting system emulators. In *Information Security*, pages 1–18. Springer, 2007.

[113] JR Raphael. The worst cloud outages of 2013 (so far). *Info World, July*, 2013. URL http://www.infoworld.com/article/2622201/cloud-computing/the-10-worst-cloud-outages--and-what-we-can-learn-from-them-.html?page=2.

[114] Edward Ray and Eugene Schultz. Virtualization security. In *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies*, page 42. ACM, 2009.

[115] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. Flip feng shui: Hammering a needle in the software stack. In *Proceedings of the 25th USENIX Security Symposium*, 2016.

[116] Gartner Press Release. Gartner says 60 percent of virtualized servers will be less secure than the physical servers they replace through 2012. http://www.gartner.com/newsroom/id/1322414, 2010. Accessed: 2016-02-06.

[117] Wolfgang Richter, Glenn Ammons, Jan Harkes, Adam Goode, Nilton Bila, Eyal De Lara, Vasanth Bala, and Mahadev Satyanarayanan. Privacy-sensitive vm retrospection. In *Proceedings of the Third USENIX Workshop on Hot Topics in Cloud Computing, HotCloud*, volume 11, 2011.

[118] Wolfgang Richter, Canturk Isci, Barry Gilbert, Jan Harkes, Vasanth Bala, and Mahadev Satyanarayanan. Agentless cloud-wide streaming of guest file system updates. In *Cloud Engineering (IC2E), 2014 IEEE International Conference on*, pages 7–16. IEEE, 2014.

[119] Rightscale. Cloud computing trends: 2016 state of the cloud survey. http://www.rightscale.com/blog/cloud-industry-insights/cloud-computing-trends-2016-state-cloud-survey#enterpriseworkloads, 2016. Accessed: 2016-03-05.

[120] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 199–212. ACM, 2009.

[121] Anthony Roberts, Richard McClatchey, Saad Liaquat, Nigel Edwards, and Mike Wray. Poster: Introducing pathogen: a real-time virtualmachine introspection framework. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1429–1432. ACM, 2013.

[122] Chunming Rong, Son T Nguyen, and Martin Gilje Jaatun. Beyond lightning: A survey on security challenges in cloud computing. *Computers & Electrical Engineering*, 39(1):47–54, 2013.

[123] Sebastian Roschke, Feng Cheng, and Christoph Meinel. Intrusion detection in the cloud. In *Dependable, Autonomic and Secure Computing, 2009. DASC'09. Eighth IEEE International Conference on*, pages 729–734. IEEE, 2009.

[124] Keyun Ruan, Joe Carthy, Tahar Kechadi, and Mark Crosbie. Cloud forensics. In *IFIP International Conference on Digital Forensics*, pages 35–46. Springer, 2011.

[125] Thomas Ruebsamen and Christoph Reich. Supporting cloud accountability by collecting evidence using audit agents. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, volume 1, pages 185–190. IEEE, 2013.

[126] Joanna Rutkowska. Subverting vistatm kernel for fun and profit. *Black Hat Briefings*, 2006. URL http://www.blackhat.com/presentations/bh-jp-06/BH-JP-06-Rutkowska.pdf.

[127] Joanna Rutkowska and Rafal Wojtczuk. Qubes os architecture. *Invisible Things Lab Tech Rep*, page 54, 2010.

[128] Mohd Izuan Mohd Saad, Kamarularifin Abd Jalil, and Mazani Manaf. Data provenance trusted model in cloud computing. In *Research and Innovation in Information Systems (ICRIIS), 2013 International Conference on*, pages 257–262. IEEE, 2013.

[129] Reiner Sailer, Enriquillo Valdez, Trent Jaeger, Ronald Perez, Leendert Van Doorn, John Linwood Griffin, Stefan Berger, Reiner Sailer, Enriquillo Valdez, Trent Jaeger, et al. shype: Secure hypervisor approach to trusted virtualized systems. *Techn. Rep. RC23511*, 2005.

[130] Vijay Sarathy, Purnendu Narayan, and Rao Mikkilineni. Next generation cloud computing architecture: Enabling real-time dynamism for shared distributed physical infrastructure. In *Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE), 2010 19th IEEE International Workshop on*, pages 48–53. IEEE, 2010.

[131] Matthias Schmidt, Lars Baumgartner, Pablo Graubner, David Bock, and Bernd Freisleben. Malware detection and kernel rootkit prevention in cloud computing environments. In *Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on*, pages 603–610. IEEE, 2011.

[132] Bruce Schneier. *Secrets and lies: digital security in a networked world*. John Wiley & Sons, 2011.

[133] Anupma Sehrawat and Neha Bishnoi. Security: A key requirement of cloud. *International Journal*, 3(6), 2013.

[134] Yasir Shoaib and Olivia Das. Pouring cloud virtualization security inside out. *CoRR*, abs/1411.3771, 2014.

[135] Yogesh L Simmhan, Beth Plale, and Dennis Gannon. A survey of data provenance in e-science. *ACM Sigmod Record*, 34(3):31–36, 2005.

[136] Kevin Skapinetz. Virtualisation as a blackhat tool. *Network Security*, 2007(10): 4–7, 2007.

[137] James E Smith and Ravi Nair. The architecture of virtual machines. *Computer*, 38(5):32–38, 2005.

[138] Jason Sonnek and Abhishek Chandra. Virtual putty: Reshaping the physical footprint of virtual machines. *Proc of HotCloud*, 2009.

[139] Sandeep K Sood. A combined approach to ensure data security in cloud computing. *Journal of Network and Computer Applications*, 35(6):1831–1838, 2012.

[140] Vijayaraghavan Soundararajan and Jennifer M Anderson. The impact of management operations on the virtualized datacenter. In *ACM SIGARCH Computer Architecture News*, volume 38, pages 326–337. ACM, 2010.

[141] Jonathan Spring. Monitoring cloud computing by layer, part 1. *IEEE Security & Privacy*, 9(2):66–68, 2011.

[142] David Strauss. The future cloud is container, not virtual machines. *Linux Journal*, 2013(228):5, 2013.

[143] Ivan Studnia, Eric Alata, Yves Deswarte, Mohamed Kaâniche, Vincent Nicomette, et al. Survey of security problems in cloud computing virtual machines. *Proceedings of Computer and Electronics Security Applications Rendez-vous (C&ESAR 2012)*, 2012.

[144] Symantec. Threat severity assessment. https://www.symantec.com/security_response/severityassessment.jsp, 2006. Accessed: 2016-12-30.

[145] Franco Travostino, Paul Daspit, Leon Gommans, Chetan Jog, Cees De Laat, Joe Mambretti, Inder Monga, Bas Van Oudenaarde, Satish Raghunath, and Phil Yonghui Wang. Seamless live migration of virtual machines over the man/wan. *Future Generation Computer Systems*, 22(8):901–907, 2006.

[146] Hsin-Yi Tsai, Melanie Siebenhaar, Andre Miede, Yulun Huang, and Ralf Steinmetz. Threat as a service?: Virtualization's impact on cloud security. *IT Professional Magazine*, 14(1):32, 2012.

[147] James Turnbull. The art of monitoring. https://artofmonitoring.com/TheArtOfMonitoring_sample.pdf, 2016. Accessed: 2016-12-10.

[148] Venkatanathan Varadarajan, Thawan Kooburat, Benjamin Farley, Thomas Ristenpart, and Michael M Swift. Resource-freeing attacks: improve your cloud performance (at your neighbor's expense). In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 281–292. ACM, 2012.

[149] Steven J Vaughan-Nichols. Virtualization sparks security concerns. *Computer*, 41(8):13–15, 2008.

[150] Lizhe Wang, Jie Tao, Marcel Kunze, Alvaro Canales Castellanos, David Kramer, and Wolfgang Karl. Scientific cloud computing: Early definition and experience. In *HPCC*, volume 8, pages 825–830, 2008.

[151] Zhenghong Wang and Ruby B Lee. Covert and side channels due to processor architecture. In *ACSAC*, volume 6, pages 473–482, 2006.

[152] Jonathan Stuart Ward. *Efficient monitoring of large scale infrastructure as a service clouds*. PhD thesis, University of St Andrews, 2015. URL http://hdl.handle.net/10023/6974. Supervised by: Dr. Adam Barker.

[153] Jonathan Stuart Ward and Adam Barker. Varanus: In situ monitoring for large scale cloud systems. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, volume 2, pages 341–344. IEEE, 2013.

[154] Jonathan Stuart Ward and Adam Barker. Observing the clouds: a survey and taxonomy of cloud monitoring. *Journal of Cloud Computing*, 3(1):1, 2014.

[155] Thu Yein Win, Huaglory Tianfield, Quentin Mair, Taimur Al Said, and Omer F Rana. Virtual machine introspection. In *Proceedings of the 7th International Conference on Security of Information and Networks*, page 405. ACM, 2014. doi: 10.1145/2659651.2659710.

[156] Network World. A wakeup call for the cloud. http://www.networkworld.com/article/2366862/iaas/a-wakeup-call-for-the-cloud.html. Accessed: 2016-09-01.

[157] Xen. How does xen work? http://www-archive.xenproject.org/files/Marketing/HowDoesXenWork.pdf, 2009. Accessed: 2016-25-13.

[158] Yubin Xia, Yutao Liu, Haibo Chen, and Binyu Zang. Defending against vm rollback attack. In *Dependable Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP 42nd International Conference on*, pages 1–5. IEEE, 2012.

[159] MM Younis A Younis and K Kifayat. Secure cloud computing for critical infrastructure: A survey. *Liverpool John Moores University, United Kingdom, Tech. Rep*, 2013.

[160] Younis Younis, Kashif Kifayat, and Madjid Merabti. Cache side-channel attacks in cloud computing. In *International Conference on Cloud Security Management (ICCSM)*, page 138, 2014.

[161] Dong Yu and Deborah Frincke. A novel framework for alert correlation and understanding. In *International Conference on Applied Cryptography and Network Security*, pages 452–466. Springer, 2004.

[162] Su Zhang, Xinwen Zhang, and Xinming Ou. After we knew it: empirical study and modeling of cost-effectiveness of exploiting prevalent known vulnerabilities across iaas cloud. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pages 317–328. ACM, 2014.

[163] Xiangliang Zhang, Zon-Yin Shae, Shuai Zheng, and Hani Jamjoom. Virtual machine migration in an over-committed cloud. In *2012 IEEE Network Operations and Management Symposium*, pages 196–203. IEEE, 2012.

[164] Yinqian Zhang, Ari Juels, Alina Oprea, and Michael K Reiter. Homealone: Co-residency detection in the cloud via side-channel analysis. In *2011 IEEE Symposium on Security and Privacy*, pages 313–328. IEEE, 2011.

[165] Yinqian Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Cross-vm side channels and their use to extract private keys. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 305–316. ACM, 2012.

[166] Wu Zhou, Peng Ning, Xiaolan Zhang, Glenn Ammons, Ruowen Wang, and Vasanth Bala. Always up-to-date: scalable offline patching of vm images in a compute cloud. In *Proceedings of the 26th Annual Computer Security Applications Conference*, pages 377–386. ACM, 2010.

[167] Dimitrios Zissis and Dimitrios Lekkas. Addressing cloud computing security issues. *Future Generation Computer Systems*, 28(3):583–592, 2012.

Appendix A

Selected Code listings

Generating the VM profile for detecting the migration

Listing A.1 The profiling script to form the profiles of the VMs

```
#!/bin/bash
while true
do
 date >> /home/taimur/migration-metrics.log
 # this will print the average latency after executing ping c times
 ping -c 30 104.196.217.206 | tail -1| awk '{print $4}' | cut -d '/' -f 2
    >> /home/taimur/migration-metrics.log
 # This will get the hypervisor type
 sudo virt-what >> /home/taimur/migration-metrics.log
 #This will get the extermal/public IP of the running machine
 wget http://ipinfo.io/ip -qO - >> /home/taimur/migration-metrics.log
 #This will get the processor information
 lscpu >> /home/migration-metrics.log
done
```

To enable monitoring processes

Listing A.2 Adding a rule to auditd to monitor the execve system call

```
sudo auditctl -a always,exit -F arch=b64 -S execve -k executed-program
```

Malicious scripts to be executed in the monitored VMs

To generate a large number of files of variable sizes in the selected VM:

Listing A.3 Malicious script to create thousands of files in a directory

```
#! /bin/bash
for n in {1..10000}; do
    dd if=/dev/urandom of=filexxx$( printf %03d "$n" ).dat bs=1 count=$((
        RANDOM + 1024 ))
done
```

It generates 10000 files in one of the directories in the chosen monitored VM (informant-client3 was chosen).

To copy contents of a directory :

Listing A.4 A script to copy all contents of a directory to another location

```
#!/bin/bash
mkdir /home/taimur/desktopcopy
cd /home/taimur/Desktop/
for f in *.txt
do
   cp -v "$f" /home/taimur/desktopcopy/"${f%.txt}"$(date +%m%d%y).txt
done
```