SECURE VIRTUAL MACHINE MIGRATION IN CLOUD DATA CENTERS

ARASH EGHTESADI

A THESIS

IN

THE CONCORDIA INSTITUTE FOR INFORMATION SYSTEMS ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF APPLIED SCIENCE IN INFORMATION SYSTEMS SECURITY

CONCORDIA UNIVERSITY

MONTRÉAL, QUÉBEC, CANADA

SEPTEMBER 2013

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Arash Eghtesadi**

Entitled: **Secure Virtual Machine Migration in Cloud Data Centers**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science in Information Systems Security

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Dr. Chun Wang _____ Chair

Dr. Lingyu Wang _____ Examiner

Dr. Yan Liu _____ External (to program)

Dr. Makan Pourzandi _____ Supervisor

Dr. Mourad Debbabi _____ Co-supervisor

Approved _____

Dr. Rachida Dssouli, Director

Concordia Institute for Information Systems Engineering

_____ 20 _____ _____

Dr. Christopher Trueman, Dean

Faculty of Engineering and Computer Science

Abstract

Secure Virtual Machine Migration in Cloud Data Centers

Arash Eghtesadi

While elasticity represents a valuable asset in cloud computing environments, it may bring critical security issues. In the cloud, virtual machines (VMs) are dynamically and frequently migrated across data centers from one host to another. This frequent modification in the topology requires constant reconfiguration of security mechanisms particularly as we consider, in terms of firewalls, intrusion detection/prevention as well as IPsec policies. However, managing manually complex security rules is time-consuming and error-prone. Furthermore, scale and complexity of data centers are continually increasing, which makes it difficult to rely on the cloud provider administrators to update and validate the security mechanisms.

In this thesis, we propose a security verification framework with a particular interest in the abovementioned security mechanisms to address the issue of security policy preservation in a highly dynamic context of cloud computing. This framework enables us to verify that the global security policy after the migration is consistently preserved with respect to the initial one. Thus, we propose a systematic procedure to verify security compliance of firewall policies, intrusion detection/prevention, and IPsec configurations after VM migration. First, we develop a process algebra called cloud calculus, which allows specifying network topology and security configurations. It also enables specifying the virtual machines migration along with their security policies.

Then, the distributed firewall configurations in the involved data centers are defined according to the network topology expressed using cloud calculus. We show how our verification problem can be reduced to a constraint satisfaction problem that once solved allows reasoning about firewall traffic filtering preservation. Similarly, we present our approach to the verification of intrusion detection monitoring preservation as well as IPsec traffic protection preservation using constraint satisfaction problem.

We derive a set of constraints that compare security configurations before and after migration. The obtained constraints are formulated as constraint satisfaction problems and then submitted to a SAT solver, namely Sugar [102], in order to verify security preservation properties and to pinpoint the configuration errors, if any, before the actual migration of the security context and the virtual machine. In addition, we present case studies for the given security mechanisms in order to show the applicability and usefulness of our framework, and demonstrate the scalability of our approach.

Acknowledgments

I would like to thank my supervisors Dr. Mourad Debbabi and Dr. Makan Pourzandi for giving me the opportunity to pursue my Master thesis in their research group, for their exceptional support and guidance, which directed me to the path of success. I am grateful for their feedbacks on my progress as well as encouragement to achieve the best. This dissertation would not have been possible without their advise. Furtheremore, I am very much thankful to Dr. Yosr Jarraya who is not only the best colleague I have ever had, but also a wonderful researcher who directed my research activities in Computer Security Laboratory. I learned a lot while discussing and brainstorming ideas with her, which are indispensable to my thesis completion. In addition, I appreciate the unconditional love and support from my wife, Pegah to whom I am deeply indebted. She was always there and stood by me through the good and bad. Finally, I would like to thank my parents for supporting me throughout my studies. I would like to express my deepest gratitude to them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recent developments in virtualization have made cloud computing an increasingly important research area. There are many IT infrastructures who are migrating into the cloud in order to benefit from this new way of delivering computing resources. Elasticity and rapid resource provisioning and scalability allow growth of resources in an on-demand and pay-as-you-go manner. These features are beneficial and attractive to businesses from small size to governmental organizations. Cloud computing service model has a layer service model [35]. These services are categorized into Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS) [35]. IaaS refers to infrastructure such as physical servers, virtual machines and so on. Example of IaaS provider are Amazon EC2 [8] and Rackspace Cloud [86]. PaaS on the other hand, refers to operating systems that provide hosting and enable deployment of applications. An example of PaaS is Google App Engine [42]. In addition, SaaS refers to the applications that run on top of the other service layers for example Microsoft Office 365 [70].

1.1 Motivation and Problem Statement

In spite of its drastic advantages, cloud computing has also specific security challenges [39]. Virtualization enables a dynamic computing infrastructure supporting the elastic nature of service provisioning and de-provisioning as requested by users while maintaining high levels of reliability and security [65]. In this setting, Virtual Machines (VMs) are software implementations created within a virtualization layer and their capability to be easily moved, copied, and reassigned between host servers is a key-enabler technology that enhances load balancing, scheduled maintenance, as well as power management. However, VM migration creates new security challenges in data centers. VMs are protected using various security mechanisms including firewalls, intrusion detection/prevention systems, etc. Specifically, firewalls are used to allow only authorized traffic to reach the protected VMs. While VMs migrate around, not only the memory and the states on the hypervisor need to be migrated, but also the network states including firewall rules. Failing to do so, may expose the running services on the migrated VM to security problems. These problems are particularly important when a VM changes data center or it changes its access points. In this latter case, there is need for modifying access rules for the firewalls. An illustrative example of this case is firewall access control list (ACL). Assume that a VM migrates to a new location under a different firewall configuration. On one hand, if the ACLs at the new location are more permissive than those at the original location, some packets that should be blocked might be allowed. This may open up several security vulnerabilities to the VM. On the other hand, if they are less permissive, some packets that should be allowed might be blocked. Furthermore, some virtual machine's running services might require specific filtering rules.

In order to ensure a secure cloud computing environment, firewalls should not be the only line of defense. Cloud providers have to offer adequate intrusion detection/prevention systems (IDPS) in order to monitor and alert on attacks targeting either cloud providers' infrastructure or the hosted VMs. Unlike in traditional data centers, IDPS systems deployed in the physical network cannot inspect VM-to-VM traffic that does not leave the physical server. Various approaches are being proposed by research initiatives [38] and standardization bodies [7] to inspect the VM-to-VM traffic. One of the proposed approaches is to rely on a virtual security appliance within the virtualized layer. Other VM-centric approaches propose the deployment of IDS functionalities within each VM. Some other initiatives propose a dedicated security VM to be deployed at each physical server with specific privileged access to the hypervisors' APIs that plays the role of IDPS. In this thesis, we consider an IDS architecture where a virtual appliance dedicated for intrusion monitoring, called security monitor (SM), is attached to the hypervisors of the hosts. In addition, we assume having a hardware-based IDS connected to the host through a network interface. In addition to IDPS, tenants can ask for the deployment of a secure Virtual Private Network (VPN) between their corporation networks and their networked VMs running in the cloud. This is to enable protecting information in transit over insecure networks or leveraging the cloud services as an extension of their corporate data centers.

Elasticity of cloud computing implies mobility, or even addition and removal (a.k.a scale up and scale down respectively) of VMs and consequently, requires reconfiguration of network nodes with respect to the new architecture, including security appliances. Some research initiatives have proposed solutions to address the implementation of dynamic reconfiguration in the cloud. For instance, an approach to automate the reconfiguration of VPN endpoints to support WAN migration

of VMs is proposed in [112]. In [95], a framework to control network flows is proposed in order to guarantee that network packets are being inspected by some security devices. However, dynamic reconfiguration is error-prone, and if not properly performed may cause security configurations inconsistencies, thus exposing the VMs as well as the whole infrastructure to serious security threats. Manually managing complex security rules can be time-consuming and error-prone. Furthermore, scale and complexity of data centers are continually increasing, which makes it difficult to rely on the administrators to update and validate the security mechanisms. Therefore, a verification framework at the cloud management layer to verify and validate security policies in different enforcement points is essential as it allows the cloud provider to make sure that the same policy is enforced after VMs migration.

1.2 Scope

In this thesis, we are mainly concerned with security issues in cloud IaaS that arise during the migration of virtual machines, which may cause inconsistency in firewall, intrusion detection and IPsec configurations. Migration of virtual machines, the process of transitioning VMs between distinct physical machines, has opened new opportunities in computing. VM migration can help in many ways such as high-availability of services, data center maintenance, transparent mobility, consolidated management, and workload balancing. While virtualization and VM migration provide important benefits, their combination may introduce new security challenges. In addition, given the large number of security rules in modern complex networks, it is very difficult to make sure that those rules are complying with determined security policies without the help of formal

verification. In this thesis, we propose a verification approach for checking the consistency of security configurations, specifically related to firewalls, intrusion detection/prevention systems and IPsec VPNs. Our framework is based on comparing the security policy before migration with the one that is deployed after migration. The main goal is to detect security problems, and to provide a useful feedback to correct them before the actual migration takes place. The verification spans both source and destination data centers in case of cross data centers migration.

1.3 Contributions

In this thesis, we concentrate on the security issues raising due to elastic nature of cloud computing, to be specific virtual machines migration in the cloud. The main contributions of this work are as follows:

- We develop a formal framework called cloud calculus, which is a process algebra that enables us to formally specify network topologies as well as security appliances location and configurations. It also enables specifying the virtual machines migration along with their security policies.

- We define the concept of firewall filtering preservation in dynamic cloud computing environment, which includes firewall filtering preservation in source data center, destination data center as well as the migrating virtual machine.

- We define the concepts of intrusion monitoring preservation as well as IPsec VPN traffic protection preservations in cloud data centers. The intrusion monitoring preservation includes source and destination hosts as well as migrating virtual machine. In case of IPsec

VPN protection preservation, we cover both data centers, migrating virtual machine as well as customer network configurations.

- We elaborate a systematic verification approach based on Constraint Satisfaction Problem (CSP), a well-established mathematical framework, to prove security preservation after migration for both the migrating VM, and other VMs in the involved data centers. Security preservation includes firewall filtering, intrusion monitoring, and IPsec VPN protection preservations.

- We derive a set of formulas, which verification enables us to conclude on security preservation, and develop a framework that describes these security preservation problems in terms of constraint satisfaction problems based on Sugar, a SAT-based constraint solver. This approach enables cloud providers to automatically verify that each time migration takes palce, security level of the hosted VMs (including the migrating VM) is preserved. It also helps in detecting and correcting the configuration errors if the verified properties are violated.

1.4 Thesis Organizations

The rest of the thesis is organized as follows: Chapter 2 provides the background information regarding cloud computing, security mechanisms, as well as formal verification and constraint satisfaction. Chapter 3 provides a detailed literature review of the main areas of research that are related to our work. Chapter 4 is dedicated to present the cloud calculus, which is a process calculus that aims at specifying cloud topology and expressing virtual machine migration. In Chapter 5, we detail the idea of firewall composition and we elaborate on reducing firewall configuration to

constraint satisfaction problems. Therein, we also present our approach to verify firewall filtering preservation and apply our approach on a case study to show the applicability of our approach. Chapter 6 is dedicated to reducing IDS and IPsec configurations to constraint satisfaction problems, and verifying intrusion monitoring preservation, as well as IPsec protection preservation. There is also a case study to illustrate the migration of a VM in cloud data centers secured through IPsec tunnel, and monitored using IDS. Finally, Chapter 7 concludes the thesis and gives some directions for future research.

Chapter 2

Background

In this chapter, we present the background information required for understanding the remainder of this thesis. The explanation of cloud computing given in Chapter 1 will be extended with more technical details and clarifications. Virtualization that plays a fundamental role in cloud computing, will be explained from a technical perspective. Then, we will discuss the security mechanisms that are used by infrastructure providers in order to make more secure cloud infrastructure and resources. More specifically, we will discuss firewalls, intrusion detection and preventions, and IPsec VPN. Finally, we will present some background about formal methods specifically bounded model checking and constraint satisfaction problem. Based on them, we developed our approach for verification of security preservation regarding configurations of security mechanisms.

2.1 Cloud Computing

This section presents a general overview of cloud computing, including its definition and comparison with related concepts and technologies.

In this thesis, we adopt the definition of cloud computing provided by the National Institute of Standards and Technology (NIST) [76]: "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction". This cloud model is composed of essential characteristics, service models, and deployment models that we will explain further in this Chapter. Cloud computing is enabled through virtualization technology. Virtualization is a technology that abstracts away the details of physical hardware, and provides virtualized resources for high-level applications. A virtualized server is commonly called a virtual machine (VM). Virtualization plays a fundamental role in cloud computing, as it provides the capability of pooling computing resources from clusters of servers, and dynamically assigning or reassigning virtual resources to applications on-demand.

Cloud computing shares certain aspects with some other technologies such as grid computing, and utility computing. Grid computing is a distributed computing paradigm that coordinates networked resources to achieve a common computational objective. The motivation behind the development of grid computing was basically scientific applications which are usually computation-intensive. Cloud computing is similar to grid computing in that it also employs distributed resources to achieve application-level objectives. However, cloud computing takes one step further

by leveraging virtualization technologies at multiple levels (hardware and application platform) to realize resource sharing and dynamic resource provisioning. Utility computing represents the model of providing resources on-demand and pricing model, which is based on usage rather than a flat rate. Cloud computing can be understood as a realization of utility computing. It adopts a utility-based pricing scheme entirely for economic reasons. With on-demand resource provisioning and utility-based pricing, service providers can maximize resource utilization and minimize their operating costs.

2.1.1 Characteristics of Cloud Computing

Cloud computing is increasingly gaining ground among a variety of users including enterprises, service providers, as well as governmental and educational entities. Cloud computing platform allows hosting of multiple services on a globally shared resource pool where resources are allocated to services on demand. Recent advances in the server virtualization technologies have improved flexibility and versatility of resource provisioning. A crucial technique that has recently emerged for data centers and cluster systems is the live migration of virtual machines. It is basically moving a virtual machine from one physical server to another, while keeping the VM's active network connections such as TCP and higher layer's sessions. This migration enables operators and administrators of the cloud to perform load balancing, workload isolation, performance, and resource management as well as non-disruptive low-level system maintenance with no perceivable effect to the end user. VM live migration primarily happens within the same data center but virtual machine migration between data denters might be also required.

Besides virtual machine mobility, cloud computing provides several interesting features [66]

that make it attractive to business owners such as:

- Multitenancy: Unlike previous computing models in which resources are dedicated to a single user, cloud computing is based on a shared resources business model meaning that multiple customers use the same resources at the network, host, and application levels. In a cloud environment, different services and resources belong to multiple providers can be located in the same data center. At the same time, responsibilities are also devided among service providers.

- Shared resource pools: There is a pool of storage and computing resources offered by infrastructure providers that can be assigned to cloud customers on demand. This dynamic capability enables infrastructure providers to manage their resource usage and operating costs. For example, an IaaS provider can take advantege of VM migration technology to get a high degree of server consolidation, while maximizing resource utilization, and minimizing cost in terms of power consumption and cooling.

- Service-orientation: Cloud computing utilizes a service-oriented approach with emphasis on service management. In the cloud environment, the services from IaaS, PaaS, or SaaS are offered according to the Service Level Agreement (SLA) that the provider and customer have agreed upon.

- Dynamic resource provisioning: In cloud computing, resources can be granted and released on demand. Such dynamic resource assignment is called resource provisioning. It enables customers to use as much resources as their actual needs are and pay just for the resources they use. When their usage drop down, then they easily release resources and pay less.

- Scalability: Cloud computing provides the ability to scale to thousands of systems, as well as the ability to massively scale bandwidth and storage space. In such architecture, users are not limited to the specific number of systems as in traditional architectures.

- Elasticity: This capability allows users to scale up and down (increase and decrease) their computing resources as required. In cloud computing, users benefit very much from such scheme as they scale up and down based on their usage and requirement over the course of time.

- Pay-as-you-go: In cloud computing, the pricing scheme varies from service to service. No matter which pricing scheme the providers take, users pay only for resources they actually use and only for the time period they need them.

2.1.2 Cloud Computing Service Models

Cloud computing services are divided into three categories, according to the abstraction level of the capability provided and the service model of providers, which are: (1) Infrastructure as a Service (IaaS), (2) Platform as a Service (PaaS), and (3) Software as a Service (SaaS) [21]. One can view these abstraction levels as a layered architecture. IaaS provides resources (such as storage, computation, etc.), and resource provisioning on demand. In PaaS, developers are given the possibility to build their own applications on top of the provided platforms. Developers do not need to be concerned about the memory usage and the processing unit their applications are using. SaaS provides the services through web portals to the end users. These online software services are more attractive than those installed on local computers and provide the same functionalities. This

model of delivery facilitates the process of software maintenance for customers, and testing and development process for developers.

2.1.3 Cloud Computing Deployment Models

Cloud deployment model consists of public, private, community, and hybrid clouds. Public clouds are those cloud environments which are publicly available to multiple tenants, while private clouds are dedicated virtualized resources for particular organizations. In the same way, community clouds are made for a specific groups of customers. In the case of public cloud, a cloud provider makes resources available and offers them as pay per use model to the tenants. On the other hand, private cloud is more secure since the control is within the organization rather than being in the cloud provider side. However, there is downside to this setting because the organization who owns the cloud is responsible for managing all the resources instead of passing the responsibility to a cloud provider. A hybrid cloud consists of at least one priviate and one public cloud. It is usually in form of a partnership. For instance, a public cloud provider forms a partnership with a company that owns a private cloud. Therefore, an organization can use the private cloud for storing confidential customer data, while it can benefit from an external public cloud to perform computations.

2.1.4 Network Security Challenges in Cloud Computing

Virtualization enables a dynamic computing infrastructure supporting the elastic nature of service provisioning and de-provisioning as requested by users while maintaining high levels of reliability and security [65]. In this setting, virtual machines (VMs) are software implementations created within a virtualization layer and their capacity to be easily moved, copied, and reassigned between

host servers is a key-enabler technology that enhances load balancing, scheduled maintenance, as well as power management. However, VM migration creates new security challenges in data centers. It is desirable to assign a permanent IP address to the VM so that when it migrates, still keeps its IP address [73]. In addition, it is suggested that the VM migration be transparent to the running applications. The migration of VMs not only takes place within a single data center, but also in some cases it happens that the VMs should be moved from one data center to another. According to [45], there are cases in which it becomes expensive to extend a particular data center. As a result, sometimes there is need to use computing and storage resources of multiple data centers to support a single service. For instance, there is a cloud operational strategy called follow-the-sun [35] that shifts processing around geographicaly dispersed data centers to balance demand proximity with low energy costs. In such cases, it is possible that at the beginning, VMs are created in a single data center. After a while, the overhead on the physical servers in that data center increases. Even for some reasons (e.g., hardware failure), a physical server is switched off. As a consequence, all the VMs in that machine have to be migrated to another physical server that could possibly be located at another data center. Here the migration between data centers becomes a need.

VMs are protected using various security mechanisms including firewalls, intrusion detection and prevention, IPsec VPN, etc. While VMs migrate around, not only the memory and the states on the hypervisor need to be migrated, but also the network states including security policy. Failing to do so, may expose the running services on the migrated VM to security problems. A simple example of this case is firewall access control lists (ACLs). Assume that a VM migrates to a new location under a different firewall configuration. On one hand, if the ACLs at the new location

are more permissive than those at the original location, some packets that should be blocked may be allowed. This may open up several security vulnerabilities to the VM. On the other hand, if they are less permissive, some packets that should be allowed may be blocked. Furthermore, some virtual machines running services might require specific filtering rules. As virtual machines are dynamically and frequently moved between hosts, manually managing the complex firewall rules can be time-consuming and error-prone. The similar issue arise when dealing with intrusion detection and preventions configurations as well as IPsec VPN settings. Furthermore, scale and complexity of data centers are continually increasing, which makes it difficult to rely on the administrators to update and validate the security mechanisms.

2.1.5 Physical vs. Virtual Security Appliances

Hypervisor is a small software application that runs either directly on top of the physical machine hardware, or on top of a guest operating system. The first architecture is called bare metal virtualization while the later one is called hosted virtualization. The hypervisor enables virtual machines to run on any architecture, and is responsible for isolation among them and also it manages VMs access to hardware (e.g. CPU, memory, etc.). There are several implementations of hypervisors by different vendors such as: Xen [113] that is the open source standard for virtualization, ESX from VMware [111], and KVM [62] which is a linux virtualization system.

Traditional hardware security appliances such as firewalls, intrusion detection and prevention systems are fundamental to provide the security and access control for the cloud infrastructure. However, these appliances no longer need to be a physical piece of hardware. In addition, the traffic that is between co-located VMs normally remains at the virtual level and passes through

virtual switches, which makes the hardware appliance blind to this type of traffic. A virtual firewall for example, can perform the same functionality as a physical firewall, but has been virtualized to work with the hypervisor. Cisco, for example, provides a virtual firewall and security gateway that secures host computers containing virtual machines [26]. Similarly, a virtualized intrusion detection and prevention system can monitor traffic on the virtual level and has better visibility over the VM-to-VM traffic.

Virtualized data centers rely on a hypervisor, which isolates the virtual machines from the physical network. This creates a virtual network within the hypervisor that connects the VMs and enables them to communicate with each other without the traffic crossing the physical network. As a result, security threats are isolated from the traditional network security tools such as intrusion detection and prevention. Co-located VMs can communicate across the virtual switch without having the traffic pass through physical network where the security tools reside. As a consequence, if any virtual machine is compromised, other VMs running at the same physical server will be at risk without the security tools at the physical network have any visibility on them. In addition, when it comes to the VM migration, if the migrating VM has been compromised or contains malicious code, and this malicious activity has not been detected at the source location, when the VM is migrated to another host, the destination location is at risk and could be compromised as well. For all these reasons, the recent trend in industry as well as research communities, is to provide different security mechanisms at the virtual level being the level of the hypervisor.

2.2 Security Mechanisms

In this Section, we provide some background about the security mechanisms namely firewalls, intrusion detection and prevention, and IPsec VPN. We consider in this thesis, that these security mechanisms have been deployed by cloud IaaS providers to protect and secure the cloud resources from attacks at the network level.

2.2.1 Firewalls

Firewalls are security mechanisms that impose restrictions on network services in such a way that only authorized traffic are allowed access, while unauthorized traffic are blocked. Firewalls are normally considered as perimeter defence because they are first line of defence in the network architecture. Different types of firewall technologies are available. The most basic firewall technology is packet filtering (a.k.a stateless firewall). It uses only transport layer information, and makes the decision based on the five tupples being source IP address, destination IP address, source port, destination port, and the protocol. It can perform traffic filtering with incoming or outgoing interfaces, which are called ingress and egress filtering respectively. Packet filtering does not examine higher layer context for example matching return packets with outgoing flow. On the other hand, stateful firewalls address this need.

Stateful frewall not only examines packet information in transport layer, but also offers advanced inspection for application layer such as the packet that initializes a connection. If the inspected packet matches an existing firewall rule that allows it, the packet is accepted and an entry is added to the state table. From that moment on, since the packets in that specific communication

session match an already existing state table entry, they are allowed access without any more inspection in the application layer. Those packets only need to have their IP address and TCP/UDP port number verified against the information stored in the state table to confirm that they are indeed part of the current exchange. This method will increase firewall performance because only new packets that initiate a connection need to be unencapsulated the whole way to the application layer.

A newer technology in stateful firewalling is the addition of a stateful protocol analysis capability, sometimes called deep packet inspection (DPI) [78]. Stateful protocol analysis improves upon standard stateful inspection by adding basic intrusion detection technology. This allows a firewall to accept or deny access based on how an application is running over the network. For example, an application firewall can detect if a type of attachment in an email message is not permitted by the organization such as files with `.exe` extentions, or if instant messaging (IM) is being used over port 80, which is reserved typically for HTTP. It can also be used to allow or deny web pages that contain particular types of active content, such as ActiveX. There are also other types of advanced firewalls including application proxy gateway, which provides a trasparent communication between two parties however, they are actually not connected directly to each other. There is also dedicated proxy servers that usually act as application-specific firewall such as HTTP proxy.

2.2.2 Intrusion Detection and Prevention Systems

An intrusion detection system (IDS) is a system to detect intrusive activities, which normally exploit vulnerabilities. Intrusion detection is an important security component in network security architecture. It monitors computer and network systems in order to detect possible attacks and

trigger alarms at the occurrence of malicious activities. They can be categorized as signature-based intrusion detection, and anomaly-based intrusion detection. Moreover, one can classify them as host-based intrusion detection and network-based intrusion detection systems. Signature-based IDSs detect intrusions that match the signature of known attacks. This type of IDS is unable to detect unknown attacks (a.k.a zero-day attacks) for which no signature is determined yet. However, they are very effective in detecting known attacks and have low false positive rate.

On the other hand, anomaly-based IDS builds a model for normal behaviour and any violation of that behaviour would be considered as an attack. This kind of IDS is able to detect zero-day attacks but, it has higher false positive rate. Artificial intelligence and neural networks play important role in the development of these types of IDS. Network-based IDSs are located at important points in the network to monitor the network traffic while, host-based IDSs run on a single host. Intrusion detection and prevention systems (IDPS) are network security appliances that monitor networks with intention to not only detect malicious activities and log information about them, but also to stop those activities in order to prevent the attack. IDPS are considered extensions of intrusion detection systems because they have intrusion detection functionality as well as additional prevention mechanisms such as sending an alarm, dropping the malicious packets, resetting the connection, and blocking the traffic from the malicious IP address. In this thesis, we use the term IDS to represent a typical intrusion detection and prevention system.

2.2.3 IPSec VPN

IP security (IPsec) is a suite of protocols developed by IETF to implement security at the IP layer [77]. IPsec is often used in order to implement Virtual Private networks (VPNs). A VPN

is a network that uses a public telecommunication infrastructure, such as the Internet, in order to provide secure access for users or remote offices to their organization's network. VPN relies on IPsec to create a tunnel between the two endpoints. The traffic within the VPN tunnel is encrypted to protect from other users of the public Internet who may intercept the communications. IPsec provides three security functionality namely authentication, confidentiality, and key management. It can be used for instance to secure connections from a branch office to the Internet user. In this thesis, it could be connectivity between the corporation network and VMs deployed in the public cloud, which enables a secure remote access. In terms of traffic encryption for the confidentiality, IPsec supports two modes: transport and tunnel. Transport mode encrypts only the data portion being the payload of the packets. Transport mode is usually used to secure end-to-end communication between two hosts. On the other hand, tunnel mode encrypts both the header and the payload. Tunnel mode is used between two security gateways, such as firewalls or routers that implements IPsec. At the reciever side, there will be an IPSec-compliant device, which decrypts every packet. For IPsec to work, the sending and receiving devices must share a public key. This is accomplished through a protocol known as Internet Key Exchange (IKE).

2.3 Formal methods

In the broad range of formal methods techniques for specification, and verification of software and hardware systems, we are interested in model checking. Model checking [27] refers to the algorithms that exhaustively explore the state space of a transition system in order to address the following problem: Having a model of the system and a given property, whether the model meets

that property.

Among model checking techniques, we focus on Symbolic Model Checking [19, 68], which was introduced around 1990, and Bounded Model Checking [14] introduced in 1999. In symbolic model checking, Bibary Decision Diagrams (BDDs) [17] are traditionally used to form a symbolic representation of a system. BDDs enable a breadth first search of the sate space, which is effective specifically when the number of system states grows [27], but they are limited to handle systems with hundreds of state variables. However, for larger systems there is possibility for space explosion problem [14].

On the other hand, another type of model checking techniques is called bounded model checking, that combines model checking with satisfiability solving. Comparing to symbolic model checking with BDDs, it does not have space explosion problem, and can handle problems with thousands of variables [14]. Satisfiability (often written as SAT) addresses the following problem: Given a Boolean formula, determine if variables of that formula can be assigned in a way that make the formula evaluate to *TRUE*. In that case we could say the formula is satisfiable. If there is no such assignment, the formula is evaluated to *FALSE* for every variable assignments. Therefore, the formula is said to be unsatisfiable. There are algorithms called SAT solvers, that can efficiently solve a large subset of SAT instances.

In the following, we will present some background about bounded model checking and constraint satisfaction problem, based on that we develop our approach for verification of security preservation regarding configurations of security mechanisms.

2.3.1 Bounded Model Checking

Bounded Model Checking (BMC), was first introduced by Biere et al. in 1999 [14]. As experiments have shown, it can solve many cases that cannot be solved by techniques that are based on BDDs. What BMC does, is basically to search for a counterexample execution whose length is bounded by a natural number n, and only tries to find counter examples (paths) that consist of no more than n transitions. In bounded model checking, a Boolean formula is constructed that is satisfiable if and only if there is a finite sequence of state transitions that reaches certain states of interest. It has been shown in [13] that if the bound n is small enough (less than 80 cycles), the SAT solver outperforms BBD-based techniquess. In fact there are classes of problems that although considered hard for BDDs, in most of the cases can be solved with SAT-based techniques. Bounded model checking consists of two phases [27]. In the first phase, the sequential behavior of a transition system over a finite interval is encoded as a propositional formula. In the second phase, a propositional decision procedure, i.e., a satisfiability solver is used to process that formula in order to either obtain a satisfying assignment, or to prove that there is no such assignment. Each satisfying assignment that is found can be decoded into a state sequence, which reaches states of interest. The BMC problem can be efficiently reduced to a propositional satisfiability problem, and therefore can be solved by SAT methods rather than BDDs [13].

A propositonal formmula is a logical expression defined over boolean variables [31]. An assignment to a set V of Boolean variables is a map $\sigma : V \to \{0, 1\}$. Note that a map is an association of an element in the range with each element in the domain. A satisfying assignment for F is a truth assignment σ such that F evaluates to 1 under σ. We will be interested in propositional formulas in a certain special form: F is in conjunctive normal form (CNF) if it is a conjunction ($\wedge$)

of clauses, where each clause is a disjunction ($\vee$) of literals, and each literal is either a variable or its negation ($\neg$). For example, $F = (a \vee \neg b) \wedge (c \vee d)$ is a CNF formula with four variables and two clauses. The Boolean Satisfiability Problem (SAT) is the following: Given a CNF formula F, does F have a satisfying assignment? All practical satisfiability algorithms, known as SAT solvers, do produce such an assignment if it exists. Unlike BBD-based methods, SAT algorithms do not suffer from the space explosion problem. Recently there has been a great advancement in the performance of SAT solvers [31]. In fact, modern SAT solvers are able to handle propositional satisfiability problems with hundreds of thousands of variables or even more [13].

2.3.2 Constraint Satisfaction Problem

Constraint satisfaction is the process of finding a solution to a propositional reasoning problem that is specified using a vector of variables that must satisfy a set of constraints. A solution is therefore a vector of values that satisfies all constraints. Many problems including those of scheduling, test generation, and verification can be encoded in CSP. Constraint satisfaction problems are typically identified with problems based on constraints on a finite domain. More formally, a CSP is defined by a set of variables $\{x_i\}_{1 \leq i \leq n}$ and a set of constraints $\{C_j\}_{1 \leq j \leq m}$. Each variable x_i is defined within a domain D_i of possible values. Each constraint C_j involves all or a subset of the variables and specifies the acceptable combinations of values for these variables. A state of the problem is defined by an assignment of values to some or all of the variables. A consistent or legal assignment is one that does not violate any constraint. A complete assignment is one in which all variables are assigned values. A solution to a CSP is a complete assignment that satisfies all the constraints. There exist programs that solve CSP problems and are called constraint's solvers. We use Sugar

CSP solver [102], a SAT-based constraint solver based on a new SAT-encoding method named "order encoding". Sugar accepts Lisp-like expressions. For instance, the constraint $C_1 \wedge C_2$ is equivalent to the expression *(and* C_1 C_2*)* in Sugar syntax. The complete language accepted by Sugar can be found in [101]. After submitting a problem to Sugar, two possible conclusion are output: either *satisfiable* (denoted hereafter as SAT), if all constraints are satisfied or *unsatisfiable* (denoted hereafter as UNSAT), otherwise. For instance, for a conjunction of constraints $c_1 \wedge \cdots \wedge c_n$, a SAT conclusion allows to infer that $\{c_i\}_{1 \leq i \leq n}$ are not disjoint whereas UNSAT conclusion asserts that they are indeed disjoint.

An example of firewall rules encoding in Sugar syntax is shown in Figure 1. The firewall rules are encoded using five tupples, which are protocol (pr), source and destination IP address (sip and dip respectively), source port (ps) and destination port (pd). Since we assume the default action is "deny" then all the rules have "allow" action. Therefore, we did not encode the field action in Sugar. However, in case when we do not consider that assumption, we can easily encode the action field in Sugar similarly.

```
(int pr 0 255)
(int sip1 0 255) (int sip2 0 255) (int sip3 0 255) (int sip4 0 255)
(int dip1 0 255) (int dip2 0 255) (int dip3 0 255) (int dip4 0 255)
(int ps 0 65535)
(int pd 0 65535)

(or

; rule 1
( and (= pr 6) (and(>= sip1 0) (<= sip1 255)) (and(>= sip2 0) (<= sip2
255))
(and(>= sip3 0) (<= sip3 255)) (and(>= sip4 0) (<= sip4 255))
(= dip1 192) (= dip2 168) (= dip3 10) (= dip4 15)
(>= ps 0) (<= ps 65535) (= pd 80))

( and (= pr 6) (and(>= sip1 0) (<= sip1 255)) (and(>= sip2 0) (<= sip2
255))
(and(>= sip3 0) (<= sip3 255)) (and(>= sip4 0) (<= sip4 255))
(= dip1 192) (= dip2 168) (= dip3 10) (= dip4 16)
(>= ps 0) (<= ps 65535) (= pd 80))

( and (= pr 6) (and(>= sip1 0) (<= sip1 255)) (and(>= sip2 0) (<= sip2
255))
(and(>= sip3 0) (<= sip3 255)) (and(>= sip4 0) (<= sip4 255))
(= dip1 192) (= dip2 168) (= dip3 10) (= dip4 17)
(>= ps 0) (<= ps 65535) (= pd 80))

; rule 2
( and (= pr 6) (= sip1 190) (= sip2 160)
(and(>= sip3 0) (<= sip3 255)) (and(>= sip4 0) (<= sip4 255))
(= dip1 192) (= dip2 168) (= dip3 10) (= dip4 15)
(>= ps 0) (<= ps 65535) (= pd 22))

( and (= pr 6) (= sip1 190) (= sip2 160)
(and(>= sip3 0) (<= sip3 255)) (and(>= sip4 0) (<= sip4 255))
(= dip1 192) (= dip2 168) (= dip3 10) (= dip4 16)
(>= ps 0) (<= ps 65535) (= pd 22))

; end
)
```

Figure 1: Firewall Encoding in Sugar

Chapter 3

Literature Review

In this section, we present the literature review with respect to three main axes of research that can be connected to our work: Network security in cloud computing, analysis of security policy consistency, and verification of security policy compliance with respect to security requirements. On one hand, policy consistency analysis initiatives focus mainly on detecting and resolving anomalies and conflicts within a given security policy configuration. On the other hand, approaches on the verification of policy compliance target the assessment of a security policy implementation with respect to security requirements specified by the network administrator. In addition, we have identified relevant research that discuss network security issues in cloud computing and presented them briefly. In the following, we will present some of the most relevant contributions in all three research areas, but we will mainly focus on proposals aiming at security policies compliance verification, which are in the same axis as our work.

3.1 Network Security in the Cloud

In spite of the benefits elasticity brings to the cloud computing model, it may cause critical security issues to arise particularly after VMs migration. The security challenges in dynamic cloud environment are emphasised in [39]. The work of Subashini and Kavitha [100] presents the various security issues of cloud computing with special attention to its service delivery models. Moreover, co-residency of VMs in a single physical machine brings new security challenges. Research has shown that a malicious VM can take advantage of shared access to hardware, and extract valuable information from other VMs, which are residing in the same machine as the attacker.

In this regards, Zhang et al. [118] discuss side channel attacks when a malicious virtual machine extract information from other co-located VMs. Prior to that, Zhang et al. [117] introduced HomeAlone, which is a system that checks for a specific tenant whether its virtual machines are isolated from the ones of other tenants. Vaquero et al. [107] present an analysis of security issues related to multitenancy attribute of cloud computing. They present some of the threats and solutions to address these issuses provided by the literature. Jasti et al. [58] also attend to the security issues concerning multitenancy for example scenarios when a malicious user controls a VM, and try to gain access to other VM's resources or try to steal the data of other users located on the same physical machine by compromising hypervisor. In these research contributions, migration of virtual machines and preservation of security is not considered as it is our primary concern. However, similar to our work they consider security challenges in the dynamic cloud environment.

In a given data center, multiple network security mechanisms including packet filters, stateful firewalls, IPsec and IPS/IDS are deployed in order to protect the data center resources as well as

all the hosted VMs. As virtual machines are dynamically and frequently moved between hosts, manually managing complex security configurations can be time consuming and error-prone. Furthermore, scale and complexity of data centers are continually increasing, which makes it difficult to rely on the administrators to update and validate security configurations. Research initiatives [49, 103, 109, 114] supported by industry acknowledged the challenge and the importance of security context migration as part of cloud elasticity mechanism. Additionally, other contributions [56, 85] claim the need for automated security management tools to maintain, for instance, firewall protection with dynamic virtual server migrations. Furthermore, many Internet drafts [46, 47, 81, 110] have been published by the IETF Network Working Group in order to investigate potential solutions for security state (context) migration. However, while these security context migration mechanisms are needed, it is of a paramount importance to ensure that they achieve the intended outcome. At the moment they lack the verification for preservation of security as we consider in this thesis.

Matthews et al. [67] propose virtual machine contracts for automating the communication and management of VM requirements including access to a particular network segment or storage system. Hajjat et al. [48] tackle the challenges in migrating enterprise services into hybrid cloud-based deployments. They address the complexity of enterprise applications, and then focus on transaction delays, wide-area communication, and cost resulted from migration. They also consider the ACL migration and the reachability policies, and provide an algorithm for ACL migration. Although these works draw attention to virtual machine migrations, but their focus is mainly performance rather than security. Thus, they do not provide any formal proof for the security policy preservation.

Some other works such as [80] draw attention to the security issues that are related to the migration process and possibility of man-in-the-middle attacks. Huan Liu [63] presents a Denial of Service Attack (DoS) in a cloud infrastructure and demonstrates how an attacker is able to shut down a subnet in the cloud data center, and provides a method to avoid such attack using dynamic provisioning capability of the cloud.

With respect to intrusion detection/prevention mechanism, a number of initiatives focused mainly on proposing a solution to handle inspection specifically designed to the cloud. For instance, [38] proposes a novel architecture for virtual machines introspection. A trade-off solution for the deployment location of the IDS (either within the host or in the network) is proposed. Modi et al. [71] present a survey on various intrusion detection techniques in the cloud. Roschke et al. [91] discuss the requirements for an IDS in the cloud. They propose an IDS management architecture for distributed IDS solutions that aims at integrating and handling different types of sensors, which collect and synthesize alerts generated from multiple hosts. In [72], another architecture for dynamic security monitoring and enforcement targeting cloud computing is proposed. This work is implemented using finite-state machines to increase the performance.

Azmandian et al. [9] present an approach that relies on the hypervisor-level data for detection of intrusive activities in the virtual machine using data mining algorithms. In [105], security issues related to the virtualization technology are reviewed, and a comparison between traditional and modern monitoring techniques is presented along with the weaknesses as well as protection and assurance levels. Dhage et al. [30] propose an IDS architecture to be deployed in a distributed cloud computing environment, where separate instances are installed for each user and a single controller is proposed to manage them.

Amani S. Ibrahim et al. [57] present CloudSec, which is a monitoring module based on virtualization technologies that performs security monitoring for virtual machines in the cloud infrastructure. In [6], an Intrusion Detection System as a Service (IDSaaS) is proposed. The latter is a network and signature-based IDS for the cloud that monitors and logs network activities between virtual machines within a pre-defined Amazon Virtual Private Cloud (VPC). These research works particularly propose new architectures and techniques for intrusion monitoring in the cloud while do not consider security issues in terms of inconsistencies generated after migration of virtual machines, which we tackle in this thesis.

3.2 Security Policy Consistency Analysis

An important body of research work focuses mainly on the classification of policy anomalies and conflicts as well as on the detection of these issues [1,3,5,22,24,25,32,33,41,50,54,87,89,90,108]. However, they mainly focus on packet filtering stateless firewalls. Very few works considered the analysis of stateful firewalls policies conflicts (e.g. [20, 29, 43]).

Al-Shaer et al. [3] proposed a classification of policy anomalies and conflicts for both centralized and distributed firewalls architectures. Hamed and Al-Shaer [50] present a classification of security policy conflicts occurring in stateless firewalls and IPsec devices in enterprise networks. In [32], an algorithm to automatically resolve conflicts between network devices is presented. Villemaire and Halle [108] show that anomalies are spatio-temporal properties of rule-based filters that can be expressed using the spatio-temporal language RL and then verified using RL model-checking for anomalies detection. Capretta et al. [22] present a conflict detection algorithm proved

to be correct using Coq proof system.

In [5], mechanisms to discover and resolve anomalies in a network protected by both firewalls and intrusion detection systems are proposed. In [89, 90], Rezvani and Aryan propose a formal language to specify security policy for firewalls for the detection of anomalies. In [54], a rule-based segmentation technique is adopted to convert a list of rules into a set of disjoint network packet spaces to detect anomalies. They extend this work in [55], by following a technique based on rule segmentation, to detect policy anomalies and extract resolutions for the identified anomalies. Chen et al. [25] present an approach based on set theory for detection of anomalies in packet filtering rule sets. Other approaches (e.g. [1, 41, 87]) propose the use of data mining techniques on log files in order to detect firewall anomalies. In [37], a management tool implemented as a web service, named MIRAGE, is developed for the analysis and deployment of configuration policies over network security components including stateless firewalls, intrusion detection systems, and VPN routers. Some other works [74, 82] propose the use of process algebra and more specifically mobile ambient [23] for the analysis and the specification of network security policies for conflict detection and analysis in single and distributed stateless firewalls.

As far as stateful firewall is concerned, only few initiatives studied the security policy consistency problem of a single or distributed stateful firewalls. Unlike packet filters, the decision in a stateful firewall on whether a packet should be allowed or blocked does not only depends on the rules, but also on a state table that allows specific traffic to be temporarily accepted as it is related to some previously initiated authorized connections. Gouda and liu [43] propose a model of stateful firewalls that allows analyzing properties of stateful firewalls including conforming, grounded, and proper. These are mainly used to verify that a stateful firewall is ”truly stateful”. The work

of Cuppens et al. [29] is dedicated to anomalies and conflicts in stateful firewalls. The innovative approach in this work is determining new configuration anomalies specific to stateful firewalls that are called intra-state rule anomalies. Algorithms to detect and resolve these anomalies were implemented as part of their tool MIRAGE [37]. However, only a single stateful firewall is considered. In [20] a technique for modeling a state as a subset of the firewall rule-set is presented. The authors employed static analysis techniques and BDDs for detecting misconfigurations in firewalls.

As for IPsec policy analysis and conflicts detection, to best of our knowledge, there are very few research initiatives done towards these objectives. There is a belief that VPN tunnels are created on demand and therefore there could not be any erroneous IPSec configurations. But this is not the case in the cloud when VMs move around. Anomalies in IPsec policy are first studied in [34], and a methodology is proposed to detect and resolve security policy conflicts in both intra-domain and inter-domain environments. Furthermore, security requirements for IPsec are formally specified in a high-level language. Satisfaction of all of these requirements implies the correctness of the IPsec policy. Hamed et al. [51] model IPsec policy using ordered binary decision diagrams (OBDDs), then intra-policy conflicts (anomalies in a single IPsec device) as well as inter-policy conflicts (anomalies between multiple IPsec devices) are classified. Niksefat and Sabaei [75] extend [51] and propose an algorithm for detecting and resolving conflicts in IPsec policy. In this work, binary decision diagrams (BDDs) are used for representing IPsec policy and rules evaluations. In [60], a data flow-oriented model for detecting security conflicts in IPsec is proposed. In order to automate the conflicts detection analysis, the model is specified in hierarchical colored Petri nets.

As we observed through studying the state of the art, important bodies of research work propose efficient approaches for detecting and resolving anomalies and inconsistencies in security

policies of stateless firewalls, whereas only few works handle stateful firewalls as well as IPsec and intrusion detection/prevention. However, detecting and resolving anomalies are not enough to ensure compliance of the security implementation with the specified policy. In fact, our work is different from the above mentioned body of research in the way that we want to make sure that there is no inconsistency in the security policy of every virtual machines including those that stay at the source location, due to VM migration. Moreover, in the case of an elastic cloud computing environment, we are dealing with VMs migrating from one host to another, or even one data center to another, and there is a huge potential for security policy misconfiguration, which can easily go to a large scale. Therefore, relying on manual methods to check and identify possible errors is out of question, and having an automated verification approach is essential.

3.3 Formal Verification of Security

In this section, first we describe research contributions on the formal verification of stateless firewalls. A test case generation approach is proposed by Brucker et al. [16] for testing firewall configuration based on a formal model of firewall policies expressed in higher-order logic. Equivalence checking have been investigated in [52, 53, 69] to verify the compliance between high-level security policy and a firewall access control rules. In [52], high-level security policy is represented as Access Control Matrix (ACM) based on which all possible communication paths are extracted. ACM is then encoded to binary format and low-level firewall rules are encoded as boolean expressions. The verification consists in the evaluation of the resulting boolean expressions. Hassan and Hudec [53] propose to extend the latter approach using Role-Based Network Security (RBNS)

as an intermediary model to generate low-level rules from high-level policy. Mejri et al. [69] elaborated a formal language for stateless firewall configuration specification endowed with a denotational semantics. A congruence relation on firewall configurations, expressed in the proposed language, is defined that allows to formally reason about firewall configuration's compliance with respect to a global security policy. This is performed by formally proving the soundness and the completeness of firewall configuration with respect to a security policy. The defined congruence relation is mainly used to detect inconsistencies.

Other research proposals rely on model-checking to solve the verification problem. Al-Shaer et al. [4] propose symbolic model checking, implemented within ConfigChecker tool, to verify both network reachability and security requirements expressed as Computation Tree Logic (CTL) properties. The network model specified as a state machine and the semantics of access control policies are encoded as BDDs. Kotenko and Polubelova [61] propose to use SPIN model checker for detection and resolution of filtering anomalies in the specification of security policy, where anomalies are expressed as LTL formulas. Gouda et al. [44] propose to verify the correctness of configurations regarding a network of stateless firewalls with tree topologies using their own defined formal model of firewall networks, namely firewall decision diagram.

Acharya and Gouda [2] demonstrated the equivalence of firewall verification and firewall redundancy checking problems for stateless firewalls and showed that any algorithm that can be used to solve either problem can be also used to solve the other problem with the same time and space complexities. Gawanmeh and Tahar [40] proposed an approach based on domain restriction implemented in Event-B and used invariant checking to verify the consistency of firewall configurations.

Reitblatt et al. [88] consider the issue of network configuration inconsistencies due to configuration changes, and propose a verification approach that guarantees preservation of well-defined behaviors when transitioning between configurations. Security policies are expressed in CTL, and verified using model checking. In addition, Bleikertz et al. [15] address security-related challenges encountered in virtualized infrastructures such as: zone isolation, secure migration, and absence of single point of failure, and demonstrate how these problems can be analyzed using model checking technique. When they address the issue of secure migration, they consider whether an intruder who has enough privileges could migrate the VM through an insecure network, or to a physical machine under his control. On the other hand, in this thesis, we consider that migration of the VM is legitimately done, and we would like to make sure that the migration does not introduce misconfiguration in different security mechanisms.

Satisfiability verification are also investigated and proposed in some proposals [10–12, 59, 83, 115]. Jeffrey and Samak [59] use bounded model checking based on a SAT solver for the analysis of reachability and cyclicity in a network of stateless firewall policy configurations. Reachability means that each rule of the policy can be fired by at least one packet, whereas cyclicity means there is a packet that causes the firewall to enter an infinite loop, without accepting or rejecting the packet. Ben Youssef et al. [10] propose an automated approach for verification of the conformance of a distributed firewall configuration to a predefined security policy based on Satisfiability Modulo Theories (SMT) technique. Their approach detects conflicts within the security policy, and returns key elements for the correction of flawed firewall configurations.

Finally, few works [83, 115] consider constraint satisfaction problems (CSP) to solve the problem of stateless firewall rules compliance with a security policy. Both works consider translating

the security policy and the firewall rules into CSPs in order to detect compliance or violation of pre-defined security requirements. In contrast to these works, we do not compare a security implementation with a security policy, but we compare the current security configuration with the last known secure configuration, which will be shown to be well-suited for frequently changing environments. Both [83, 115] consider a single stateless firewall rules, whereas we support a more complicated case with distributed stateless firewalls. Zhang et al. [116] propose a technique based on Booleans satisfiability in order to compare two firewall configurations, and verify whether they are equal or one is included in the other. They present a method for firewall synthesis using Quantified Boolean Formula (QBF) solver. In this work, the case of having stateful firewall is not considered. Bera et al. [12] propose a framework based on Quantified Boolean Satisfiability Checking (QSAT) problem in order to verify the enforcement of security policy in terms of ACL rules that are distributed across network interfaces.

With respect to intrusion detection/prevention, few works [94, 104] propose model checking techniques to model and analyze IDS configurations. Ben Tekaya et al. [104] employed a formal verification technique based on model checking of temporal logic formulas to verify the correctness of the intrusion detection system using the SMV model checker. The properties are either verified if the behavior is normal, or violated if the behavior is intrusive. In [94] a model checking verification approach is presented to detect specification errors in attack signatures of intrusion detection. The attack signatures are transformed to PROMELA [84], and the model-checker SPIN [98] is used for the verification. Uribe et al. [106] propose an approach for modeling and reasoning about the configurations of a combination of network intrusion detection systems (NIDS) and firewalls. They employed constraint logic programming to model the network as well as NIDS and firewall

configurations. The approach can be applied to generate NIDS configurations from event specifications, and to detect multi-step attacks. A tool is implemented that is able to process Cisco PIX firewall rules, and analyze abstract NIDS configurations to determine whether a detection policy is enforced or not.

Song et al. [97] propose a formal framework for the analysis of intrusion detection systems that employs declarative rules for attack recognition. The main goal of this framework is to reason about the effectiveness of an IDS, therefore to prove that a given IDS can detect all attacks that would violate security requirements of a given system. Couture et al. [28] present an intrusion detection technique based on temporal logics. The proposed logic can express timing, safety, and repetition properties useful to address stateful intrusion detection. Stakhanova et al. [99] present a framework for the analysis of host-based and network-based intrusion detection systems for the purpose of conflict detection in the rule-sets. In [92] a framework based on Event Calculus (EC) for formal analysis of intrusion detection systems is presented. This framework checks that security requirements are preserved at run-time by monitoring the satisfaction of the corresponding EC formulas. This is done by observing the network at run-time, and checking observations against specified network behavior.

To the best of our knowledge, there is no work tackling the verification of stateful firewalls, as well as IPsec VPN configuration. In addition, most of the existing proposals addressing compliance verification of security policies, successfully compare a given security configuration implementation against a set of security requirements. However, in a scenario where VMs have to migrate frequently and promptly, such an approach can become a real bottleneck if not appropriately tailored to such a fast changing environment. For instance, applying such an approach means that

any time a migration occurs, one has to consider each single security requirement of the co-located VMs in the updated data centers in addition to the requirements of the migrating VM. In contrast to these methodologies, we propose to divide the problem into two main sub-problems: verification of security of the migrating VM, and verification of security of the non-migrating VMs all together. Therefore, we can verify the security of all non-migrating VMs at once, which accelerate the process. To this end, we propose to compare the new security configuration with the "last known secure configuration" deployed in the source data center.

Chapter 4

Cloud Calculus

In this chapter, we aim at providing a formal framework to express and verify the VM deployment and migration process in the cloud from the security point of view. More precisely, we propose a framework that allows expressing the deployment and migration of VMs along with their related security policies, and then verifying the preservation of the security after migration. This thesis is the first initiative that employs a process algebraic approach for this matter. The majority of existing research initiatives mainly focus on performance evaluation, downtime, and cost analysis of virtual machines migration process.

Cloud calculus is a process algebra that aims at specifying cloud topology, and expressing virtual machines migration within the same as well as across data centers along with their corresponding security policies. Cloud calculus is built upon a subset of the Mobile Ambients (MA) [23] and the Non-interfering Boxed Ambients [18], and extends them with new constructs. These constructs allow expressing specific sort of ambients, such as security ambient and packet ambient, in addition to security policies at specific locations in the network, the selection of those policies

that need to be migrated with their corresponding VMs, and the update of the destination location with the new policies. The choice of the ambient concept as a foundation for the cloud calculus is justified as it was successfully used by other researchers to represent a network as a graph of nested nodes (e.g. [79]). Ambients allow representing any type of resources including firewalls, switches, routers, gateways, physical hosts, and VMs. Particularly, MA calculus is based on the concepts of hierarchy and grouping, which allows to fully represent the topology of a network. Furthermore, ambients have capabilities of moving around and communicating with other ambients.

4.0.1 Syntax

The cloud calculus comprises six syntactic categories: terms T, processes P, capabilities M, firewall policies G, ambient names A, and locations η. In the following, we briefly explain each construct. Ambient names can be a fixed name n or a variable u. Locations η are used to indicate where the communications take place: either locally within the same ambient, or across ambient boundaries (between a parent and a child). The location $\approx$ means towards the parent, the location A means towards the sub-ambient named A, and $\rightsquigarrow$ (generally omitted) means local communication. Terms enable us to specify the type of data that can be communicated between ambients. A term T can be a capability M, an ambient A, a security policy G, or a variable x. The syntax of cloud calculus is provided in Figure 2 and Figure 3. In the following we detail the constructs related to processes, capabilities, and security policies.

A process can be defined using the following constructors:

- The process 0 represents the inactive process, that does nothing.

Terms

T	::=	M	*capability*
	|	A	*ambient*
	|	G	*rules*
	|	x	*variable*
	|	$f(\overrightarrow{T})$	*function application*

Locations

η	::=	A	*child*
	|	$\twoheadleftarrow$	*parent*
	|	$\leftrightsquigarrow$	*local*

Processes

P, Q	::=	0	*inactivity*
	|	$(\nu n)P$	*restriction*
	|	$P \mid Q$	*composition*
	|	$!P$	*replication*
	|	$M.P$	*capability*
	|	$A[P]$	*ambient*
	|	$h \rhd A[P]$	*packet ambient*
	|	$G :: A[P]$	*security ambient*
	|	$(x)^{\eta}.P$	*input*
	|	$\langle T \rangle^{\eta}.P$	*output*

Capabilities

M, N	::=	ϵ	*empty path*
	|	x	*variable*
	|	*in* A	*enter A*
	|	*out* A	*exit A*
	|	$M.N$	*path*
	|	$\downarrow (x, A, A')$	*export rules*
	|	$\uparrow (G)$	*import rules*

Ambients Names

A	::=	u	*variable*
	|	n	*name*

Figure 2: Syntax of the Cloud Calculus - Part1

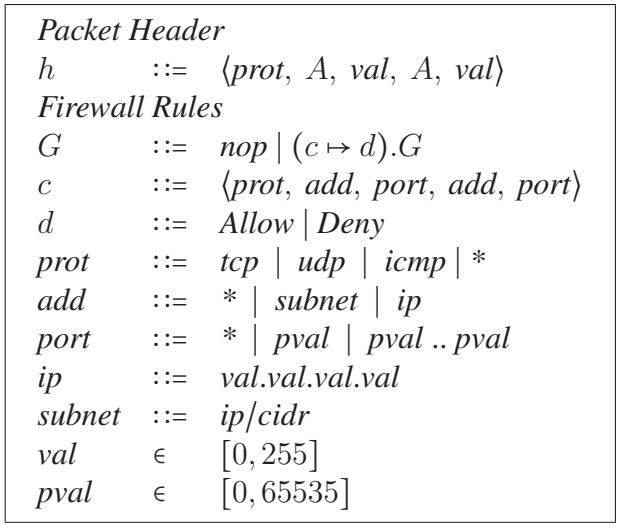

Figure 3: Syntax of the Cloud Calculus - Part 2

- The process $(\nu n)P$ denotes the restriction operator that creates a new (unique) name n within the scope P. It can be used to name ambients and operate on ambients by name.

- The process $P \mid Q$: denotes the parallel composition of two processes P and Q.

- $!P$: represents the unbounded replication of the process P that allows defining iteration and recursion.

- $G :: A[P]$: denotes an (security) ambient named A containing a running process P, and protected by a security policy defined in G.

- $h \rhd A[P]$: is the packet ambient named A that possesses a header h, and contains a running process P. The entity h denotes a five tuple consisting of protocol, source ambient, source port, destination ambient, and destination port.

- $M.P$: is the process that executes a capability M, and then continues as P.

- $(x)^{\eta}.P$: is the anonymous polyadic input of values that is a binder of the variable x with continuation P.

- $\langle T \rangle^{\eta}.P$: is the anonymous polyadic output of T.

Capabilities are obtained from names or variables. They can be described as follows:

- x: denotes a capability variable.

- *in* A: is the capability to enter into an ambient A,

- *out* A: is the capability to exit an ambient A,

- $M.N$: is a composition of capabilities forming a path,

- ϵ: denotes the empty path,

- $\downarrow (x, A, A')$: allows exporting security policies of the enclosing ambient that specifically concerns the ambient A and bound it into the variable x. The ambient A' represents the destination security ambient.

- $\uparrow (G)$: allows importing security policies G into the enclosing ambient.

A security policy denoted by G represents a sequence of rules defining constrains on the mobility of ambients and on the communication between them. Security policies can be expressed as follows:

- *nop*: denotes the empty security policy.

- $(c \mapsto d).G$ denotes a sequence of access control rules. A given access control rule is defined by the tuple c and the decision d (either *allow* or *deny*). The tuple c is formed by the protocol $prot$, the source ambient represented by an IP value or a range add, the source port represented by a port number value or a range $port$, the destination ambient add, and the destination port $port$. Note that, in practice, the name of an ambient corresponds to an IP address.

The free names and free variables, noted *fn*() and *fv*() respectively, have standard definitions, and will not be detailed here. The notation $P[T/x]$ denotes the free substitution defined only if T and x are of the same arity. A closed process contains no free variable. We also omit the inactivity 0 in process expression such as $P \mid 0$ and $M.0$.

4.0.2 Operational Semantics

The operational semantics is defined in terms of reduction and structural congruence. The structural congruence, noted ≡, is the least congruence that satisfies the laws in Figure 4.

Before presenting the cloud reduction rules, we present two auxiliary sub-reductions, namely $\rightarrow_{ev}$ and $\rightarrow_{exp}$. First one is needed for evaluation of the security policy G while having a packet ambient with a header h trying to move from a source to a destination. The second sub-reduction is needed for determining the rules to be exported to the new destination ambient given the name of the migrating ambient A and the destination ambient A'. Given a sequence of firewall rules G, we define the auxiliary function ev that evaluates G in the sequential order. We define $[\![c]\!](h)$ as the evaluation of the tuple c of a specific rule $(c \mapsto d)$ against the values in the tuple h. The function $[\![c]\!](h)$ returns $true$ if the header h matches c, and returns $false$ otherwise. Figure 5 illustrates

MONOID

$P|0 \equiv P$

$P|Q \equiv Q|P$

$P|(Q|R) \equiv (P|Q)|R$

REPL

$!P \equiv P|!P$

RES INACT

$(\nu n)0 \equiv 0$

RES RES

$(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P \qquad n \neq m$

RES PAR

$(\nu n)(P|Q) \equiv P|(\nu n)Q \qquad n \notin \mathit{fn}(P)$

PATH ASSOC

$(M.N).P \equiv M.(N.P)$

RES AMB

$(\nu n)m[P] \equiv m[(\nu n)P] \qquad n \neq m$

RES SEC AMB

$(\nu n)G :: m[P] \equiv G :: m[(\nu n)P] \qquad n \neq m$

RES PCKT AMB

$(\nu n)h \rhd m[P] \equiv h \rhd m[(\nu n)P] \qquad n \neq m$

SEC AMB

$\mathit{nop} :: n[P] \equiv n[P]$

Figure 4: Structural Congruence

the reduction rules for $\rightarrow_{ev}$. Rule SEQ allows to evaluate the firewall rules in the sequential order. Rule MATCH is used when the currently evaluated firewall rule matches with h and has a decision d. Rule NEXT is used when the currently evaluated firewall rule doesn't match with h. Finally, rule NONE is used when we reach the end of the firewall rules sequence with no match. We denote by $\Downarrow_{ev}$ the reduction to normal form of the expression $h \Vdash ev(G)$.

In order to select the firewall rules corresponding to a migrating ambient A, we define an auxiliary function *exp* that processes each firewall rule and decides either to completely move (Rule MOVE) the rule, to copy (Rule COPY) the rule, or to do nothing (Rule NEXT). Rule SEQ and LAST are used to process the security rules in sequential order, and to evaluate the last firewall rule

SEQ	$h \Vdash ev((c \mapsto d).G) \rightarrow_{ev} ev((c \mapsto d)).G$
MATCH	$h \Vdash ev((c \mapsto d)).G \rightarrow_{ev} d$
	if $[\![c]\!](h) = true$
NEXT	$h \Vdash ev((c \mapsto d)).G \rightarrow_{ev} ev(G)$
	if $[\![c]\!](h) = false$
NONE	$h \Vdash ev(nop) \rightarrow_{ev} Deny$

Figure 5: Reduction Rules for Firewall Rules Evaluation

respectively. Note that we have to copy a firewall rule when the rule concerns more than one VM such that c integrates c' and c'', written $c = c' + c''$. We define a set of five projection functions, denoted by π_i^5, that takes a tuple c and returns the i^{th} element of the tuple. The function *Sub*(n,n') returns $true$ if n is a sub-ambient of n' and $false$ otherwise. Figure 6 provides the reduction rules for $\rightarrow_{exp}$. We denote by $\Downarrow_{exp}$ the reduction to normal form of the expression $A', A \Vdash \langle exp(G),\ \text{nop}\rangle$, where A is the migrating ambient, and A' is the destination firewall. The normal form is of the form $\langle G, G'\rangle$, where G' are the rules to be migrated and G are the rules that remain within the enclosing ambient.

SEQ	$A', A \Vdash \langle exp((c \mapsto d).G),\ G\rangle \rightarrow_{exp} \langle exp((c \mapsto d)).G,\ G\rangle$
MOVE	$A', A \Vdash \langle exp((c \mapsto d)).G,\ G\rangle \rightarrow_{exp} \langle exp(G),\ (c \mapsto d).G\rangle$
	if $(\pi_2(c) = A$ and $Sub(\pi_4(c), A')$) or $(\pi_4(c) = A$ and $Sub(\pi_2(c), A'))$
COPY	$A', A \Vdash \langle exp((c \mapsto d)).G,\ G\rangle \rightarrow_{exp} \langle (c'' \mapsto d).exp(G), (c' \mapsto d).G\rangle$
	if [$d = Allow$ and ($(\pi_2(c) = A$ and $!Sub(\pi_4(c), A')$) or
	$(\pi_4(c) = A$ and $!Sub(\pi_2(c), A')$) or $(A \subset \pi_2(c) \cup \pi_4(c)))$] where $c' + c'' = c$
NEXT	$A', A \Vdash \langle exp((c \mapsto d)).G,\ G\rangle \rightarrow_{exp} \langle (c \mapsto d).exp(G), G\rangle$
	if $(A \notin \pi_2(c)$ and $A \notin \pi_4(c))$ or
	$(d = Deny$ and $(A \subset \pi_2(c)$ or $A \subset \pi_4(c))$ or
	$(d = Deny$ and $A = \pi_2(c)$ and $!Sub(\pi_4(c), A')$) or
	$(d = Deny$ and $A = \pi_4(c)$ and $!Sub(\pi_2(c), A'))$)
LAST	$A', A \Vdash \langle G.exp(nop),\ G\rangle \rightarrow_{exp} \langle G, G\rangle$

Figure 6: Reduction Rules for Firewall Rules Export

The cloud calculus reduction relation uses the two aforementioned sub-reductions and it is

defined by the rules for mobility, communication, firewall rules manipulation, and structural rules given in Figure 7. Therein, contexts are processes with one hole defined as follows:

$$\mathcal{C}[\cdot] ::= [\cdot] \mid P|\mathcal{C}[\cdot] \mid (\nu n)\mathcal{C}[\cdot] \mid G :: A[\mathcal{C}[\cdot]] \mid h \rhd A[\mathcal{C}[\cdot]]$$

The reduction relation $\rightarrow^*$ denotes the reflexive and transitive closure of $\rightarrow$. In the following, we explain the reduction rules:

- Rule (ENTER) allows a (non-packet) ambient n containing the process *in* $m.P$ to enter a sibling ambient m. If no sibling m exists, the operation blocks until a time when such a sibling exists. If more than one m sibling exists, any one of them can be chosen.

- Rules (ENTER ALLOW) and (ENTER DENY) are used with a packet ambient p containing a process *in* $m.P$ that is willing to enter m, a sibling security ambient containing the firewall rules G_m. The first rule applies if there is a firewall rule with decision allow that matches with h, which results in the packet p entering the ambient m. The second rule applies if there is either a matching security rule with decision deny or no matching rule at all, which results in dropping the packet ambient p.

- Rule (EXIT) is used to allow a (non-packet) ambient n containing the process *out* $m.P$ to exit a parent ambient m. If the parent ambient is not named m, the operation blocks until such a condition holds.

- Rule (EXIT ALLOW) and (EXIT DENY) are used with a packet ambient p containing a process *out* $m.P$ that is willing to exit m, a parent security ambient containing the firewall rules G_m. The first rule applies if there is a firewall rule with decision allow that matches with h, which

results in the packet p exiting the ambient m. The second rule applies if there is either a matching security rule with decision deny, or no matching rule at all, which results in dropping the packet ambient p.

- Rule (LOCAL) allows a local communication between a local input of x and a local output T within the same ambient to take place. This results in the substitution of all occurrences of variable x in P with the value T.

- Rule (INPUT n) allows a communication between a parent and its child by an output of T from a child ambient n, and an input of x from its parent ambient.

- Rule (OUTPUT n) allows a communication between a parent and its child by an output of T from a parent ambient, and an input of x from its child ambient n.

- Rule (IMPORT) defines an import operation (dual of export) of a sequence of firewall rules G' to a security ambient n. We define an auxiliary function *Merge* that allows to merge two sequences of firewall rules such that G'_n=*Merge*(G_n, G') represents the new firewall rules sequence of n.

- Rule (EXPORT) defines an export operation of a sequence of firewall rules G' that are related to a migrating virtual machine m, and having as destination a security ambient m'. This results in the substitution of all occurrences of variable x in P with the value G'.

Mobility:

(ENTER) $G_n :: n[\textbf{\textit{in}}\ m.P \mid Q] \mid G_m :: m[R] \ \rightarrow\ G_m :: m[G_n :: n[P \mid Q] \mid R]$

(ENTER ALLOW) $$\frac{h \ \Vdash\ ev(G_m) \Downarrow_{ev} Allow}{h \rhd p[in\ m.P] \mid G_m :: m[Q] \ \longrightarrow\ G_m :: m[h \rhd p[P] \mid Q]}$$

(ENTER DENY) $$\frac{h \ \Vdash\ ev(G_m) \Downarrow_{ev} Deny}{h \rhd p[in\ m.P] \mid G_m :: m[Q] \ \longrightarrow\ G_m :: m[Q]}$$

(EXIT) $G_n :: n[G_m :: m[\textbf{\textit{out}}\ n.P \mid Q] \mid R] \ \rightarrow\ G_m :: m[P \mid Q] \mid G_n :: n[R]$

(EXIT ALLOW) $$\frac{h \ \Vdash\ ev(G_m) \Downarrow_{ev} Allow}{G_m :: m[h \rhd p[in\ m.P] \parallel Q] \ \longrightarrow\ h \rhd p[in\ m.P] \mid G_m :: m[Q]}$$

(EXIT DENY) $$\frac{h \ \Vdash\ ev(G_m) \Downarrow_{ev} Deny}{G_m :: m[h \rhd p[in\ m.P] \mid Q] \ \longrightarrow\ G_m :: m[Q]}$$

Communication:

(LOCAL) $(x).P \mid \langle T \rangle .Q \ \rightarrow\ P\{T/x\} \mid Q$

(INPUT n) $(x)^n.P \mid G_n :: n[\langle T \rangle^{\wedge} .Q \mid R] \ \rightarrow\ P\{T/x\} \mid G_n :: n[Q \mid R]$

(OUTPUT n) $\langle T \rangle^n .P \mid G_n :: n[(x)^{\wedge}.Q \mid R] \ \rightarrow\ P \mid G_n :: n[Q\{T/x\} \mid R]$

Firewall Rules:

(IMPORT) $$\frac{G'_n = \textbf{\textit{Merge}}(G_n, G')}{G_n :: n[\uparrow (G').P] \rightarrow G'_n :: n[P]}$$

(EXPORT) $$\frac{m', m \Vdash \langle \textbf{\textit{exp}}(G_n),\ G \rangle \Downarrow_{exp} \langle G'_n,\ G' \rangle}{G_n :: n[\downarrow (x, m, m').P] \ \rightarrow\ G'_n :: n[P\{G'/x\}]}$$

Structural and Context:

(STRUCT) $$\frac{P \ \equiv\ P' \qquad P' \ \longrightarrow\ Q' \qquad Q' \ \equiv\ Q}{P \ \longrightarrow\ Q}$$

(CTX) $P \ \longrightarrow\ Q \ \Rightarrow\ \mathcal{C}[P] \ \longrightarrow\ \mathcal{C}[Q]$

Figure 7: Cloud Calculus Reduction Rules

4.1 Security Mechanisms Syntax

In this section, we present the cloud calculus syntax regarding stateless firewall, IDS, and IPsec security mechanisms.

4.1.1 Stateless Firewall Syntax

The syntax of a firewall configuration language in BNF is presented in Figure 8.

Firewall Configuration			
F	::=	$\{m : L, \cdots, m : L\}$	*multiple ACL*
L	::=	*nop*	*empty*
	|	$(p \mapsto d).L$	*sequence*
d	::=	*allow*	*allow decision*
	|	*deny*	*deny decision*
	|	*Jump m*	*link to ACL m*
ip	::=	*val.val.val.val*	
subnet	::=	*ip/cidr*	
val	$\in$	$[0, 255]$	
pval	$\in$	$[0, 65535]$	
prot	::=	*tcp* | *udp* | *	
$\bowtie$	$\in$	$\{=, \subset\}$	

Rule Predicate			
p	::=	*sip* $\bowtie$ *add*	*source addr*
	|	*ps* $\bowtie$ *port*	*source port*
	|	*dip* $\bowtie$ *add*	*destination addr*
	|	*pd* $\bowtie$ *port*	*destination port*
	|	*pr = prot*	*protocol*
	|	$p \wedge p$	*conjunction*
add	::=	*	
	|	*subnet*	
	|	*ip*	
port	::=	*	
	|	*pval*	
	|	*pval .. pval*	

Figure 8: Firewall Syntax

The symbol *, depending on its position, denotes the range of possible values in terms of IP, port, or protocol. A firewall configuration F can be composed of a number of ACLs L. For a given category of firewalls, the configuration language, rules' organization, and the interaction between multiple ACLs are the main variation factors between firewalls from different vendors. An ACL, denoted by L, is associated with a name m. The latter allows naming ACLs in order to link an ACL to another using the construct *Jump m*. The firewall rules in a given ACL are organized in

sequential order. Let f_i be a single firewall rule, denoted by $p_i \mapsto d_i$ where p_i is a predicate representing the filtering condition of the rule, and d_i is the corresponding decision. The predicate p_i is the conjunction of the set of predicates on the proceeded packet's attributes. The commonly used attributes in the packet header are the protocol, the source IP address, the destination IP address, the source port number, and the destination port number. For instance, the topology of the data center $DC1$ depicted in Figure 9 can be expressed using cloud calculus as follows:

$D_1 = F1 :: G1[\ F2 :: S1[\ S3[\ PS1[P_{WEB1}]\ ]\ ]\ |\ P]$

where $P = F3 :: S2[\ S3\ |\ S4[\ PS2[P_{AP1}]\ ]\ ]$ and $P_{WEB1} = VM_5\ |\ VM_6\ |\ VM_7$

The topology of the data center $DC2$ can be expressed as follows:

$D_2 = F4 :: G2[\ F5 :: S5[\ S6[\ PS3[P_{DB1}]\ |\ PS4[VM_1\ |\ VM_2\ |\ VM_3\ |\ VM_4]]\ ]]$

The cloud calculus operational semantics is defined in terms of reduction rules and structural congruence. Firewall rules migration can be described using the cloud calculus reduction rules.

4.1.2 IDS Syntax

IDS rules differ from one intrusion detection system to another. In our case, we consider Snort [96] intrusion detection system, which is an open source IDS that is widely deployed in many networks. The syntax of an intrusion detection configuration language in BNF is presented in Figure 10. The proposed syntax allows us to express IDS in cloud calculus. The bit-length of the allocated network prefix is denoted by *cidr*, which represents Classless Inter-Domain Routing (CIDR) that is a method for specifying IP addresses and their associated routing prefix. The symbol *, depending on its position, denotes the range of possible values in terms of IP, port, or protocol. An intrusion detection configuration I is composed of a sequence of rules. A given rule is of the form $prd \mapsto act$

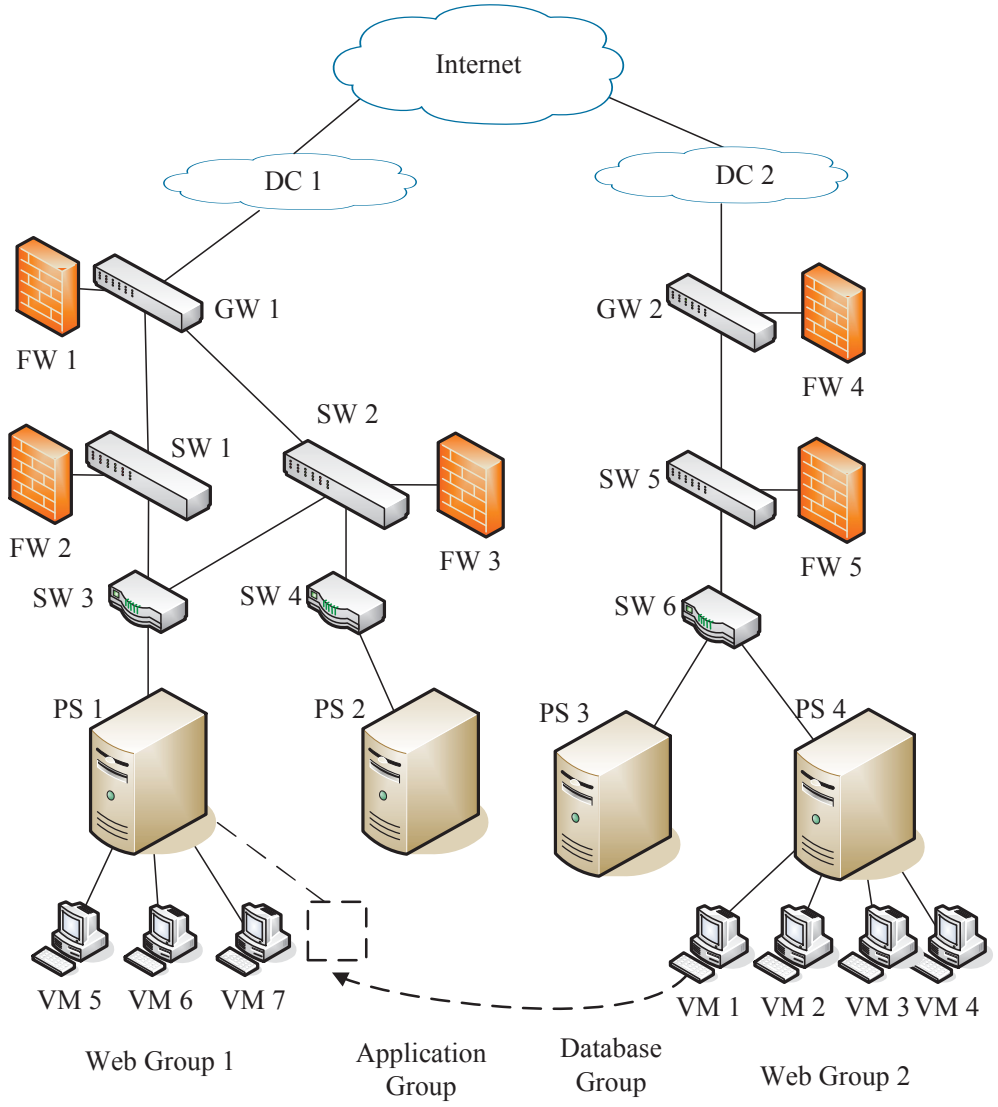

Figure 9: Cloud Network Model

where prd is a predicate over packets attributes (i.e. protocol, source and destination address, source and destination port), and the direction of the communication denoted by dr, and the Snort rule's option opt which represents the content identifying the signature of the attack.

IDS Configuration

I	::=	*nop*	*empty*
	|	$(prd \mapsto act).I$	*sequence*
act	::=	*alert*	*alert*
	|	*log*	*log*
	|	*pass*	*pass*
ip	::=	*val.val.val.val*	
subnet	::=	*ip/cidr*	
val	$\in$	$[0, 255]$	
pval	$\in$	$[0, 65535]$	
prot	::=	*tcp* | *udp* | *icmp* | *ip* | *	
dir	$\in$	$\{\rightarrow, \leftrightarrow\}$	
$\bowtie$	$\in$	$\{=, \subset\}$	

Rule Predicate

prd	::=	*pr = prot*	*protocol*
	|	*sip* $\bowtie$ *add*	*source addr*
	|	*ps* $\bowtie$ *port*	*source port*
	|	*dr = dir*	*direction*
	|	*dip* $\bowtie$ *add*	*destination addr*
	|	*pd* $\bowtie$ *port*	*destination port*
	|	*opt = option*	*rule options*
	|	$prd \wedge prd$	*conjunction*
add	::=	*	
	|	*subnet*	
	|	*ip*	
port	::=	*	
	|	*pval*	
	|	*pval .. pval*	

Figure 10: IDS Syntax

4.1.3 IPsec Syntax

The syntax related to IPsec VPN configurations is illustrated in Figure 11. An IPsec configuration, denoted by E, is a sequence of ACL rules $p \mapsto d_p$ so that d_p represents the action and encryption parameters for the protected traffic. The predicate p is the condition on the packet's header attributes, and it mainly includes source and destination IP addresses with source and destination ports as well as the protocol. Three actions d_p are possible: deny, bypass, or protect. In the case of action "protect", specific transforms are specified to be applied on the traffic matching p. An IPsec

transform t_e is any cryptographic service that can be used to protect network traffic. A transform is composed of a security service sec_prot, that is either IPsec AH and ESP protocols and operating either in transport or tunnel mode denoted by $mode$ along with the cryptographic algorithm and the necessary cryptographic parameters specified by $param$.

IPsec Configuration

E	::=	*nop*	*empty*
	|	$(p \mapsto d_p).E$	*sequence*
d_p	::=	*protect* t_e	*secure*
	|	*bypass*	*insecure*
	|	*discard*	*drop traffic*
t_e	::=	$(sec\_prot, mode, param)$	
sec_prot	::=	*AH*	
	|	*ESP*	
$mode$	::=	*Transport*	
	|	*Tunnel* ip	
$param$	$\in$	$\{3DES, \cdots\}$	

Figure 11: IPsec Syntax

Chapter 5

Stateless Firewalls Filtering Preservation

In this chapter, first we present distributed firewalls composition which enable us to consider architecture with more than one firewalls. Then, we show how we encode a single firewall as well as a distributed firewalls configuration in constraint satisfaction problem (CSP). After that, we present our verification approach for firewall filtering preservation in dynamic cloud computing environment. We present firewall filtering preservation concept, and then we formally define firewall filtering preservation in source and destination data centers as well as for the migrating VM. Afterward, we elaborate on the CSP constrains which satisfiabilities allow verifying the defined firewall filtering preservation. Finally, we describe the proposed verification procedure, and explain the interpretation of the outcome of Sugar solver by demonstrating a case study.

5.1 Firewall Composition

In the case of a well-engineered network with distributed firewalls, multiple paths may exist to reach a given destination, and dynamic routing is used in order to improve performance and reliability. Packets crossing different paths may be processed by different firewalls rules. Consequently, a packet could traverse different ACL at different times. In this section, we define a language to express distributed firewalls composition, denoted by T. Two firewalls may be either composed in serial or in parallel. The syntax of T is provided in BNF as follows:

$$T ::= F \mid T \odot T \mid T \oplus T$$

where F is a single firewall configuration, $T_1 \odot T_2$ denotes serial composition of T_1 and T_2, and $T_1 \oplus T_2$ is the parallel composition of two firewall configurations T_1 and T_2. In serial composition, $T_1 \odot T_2$ means that a packet that survives filtering of rules of T_1 is then necessarily filtered by T_2. The operator $\odot$ is associative and distributive over $\oplus$. With respect to parallel composition, $T_1 \oplus T_2$ means that a packet is either filtered by T_1 or by T_2. The operator $\oplus$ is commutative and associative. The parallel firewalling operation is useful for better high availability. In this case if one of the links goes down, the second is used to carry the traffic to the destination.

Given a cloud calculus term expressing the topology of a cloud data center, one can define a function that parses the expression in order to infer the resulting firewalls composition expressed in the above syntax. Thus, we denote by $\mathcal{P}$ such a function that takes as input a cloud calculus term and an ambient name A and returns the firewall composition expression.

5.2 Encoding Firewall Configuration in CSP

In the following, we present how we encode a single firewall, and then a distributed firewall configuration in CSP. The CSP variables are the set of integer variables V needed to encode the condition filters of a given firewall rule. In order to represent an IP address, 4 integer variables within the range of $[0, 255]$ are used. A source (resp. destination) IP address is represented by $\{sip_i\}_{1 \leq i \leq 4}$ (resp. $\{dip_i\}_{1 \leq i \leq 4}$). The integer variable $pr \in [0, 255]$ represents the protocol number. We also define two integer variables to encode the source and destination port numbers, respectively ps and pd within the range of values $[0, 65535]$. Thus, the set of integer variables is $V = \{pr, sip1, sip2, sip3, sip4, ps, dip1, dip2, dip3, dip4, pd\}$.

In the syntax of CSP for Sugar constraint solver, declaring an integer variable for instance pr within the range $[0, 255]$ is denoted by *(int pr 0 255)*. Each single firewall rule predicate p is encoded as a CSP constraint. It is a conjunctive logical formula over the variables in V. The corresponding CSP constraint is written as: $pr = v_1 \wedge sip1 = v_2 \wedge sip2 = v_3 \cdots \wedge dip4 = v_{10} \wedge pd = v_{11}$ where v_i is to be replaced by the actual value in the corresponding firewall rule. The firewall ACL is encoded as a constraint $\mathcal{C}$ built as a disjunctive logical formula over all firewall rules formulas. Thus, the ordered sequence of firewall rules $(p_n \rightarrow d_n).(p_{n-1} \rightarrow d_{n-1}). \cdots .nop$ are encoded as the logical formula $p_1 \vee p_2 \vee \cdots \vee p_n$. In Sugar syntax, this is denoted by *(or* $p_1 \ \ldots \ p_n$ *)*.

Since we consider that all rules have "allow" decisions, this constraint represents the set of packets accepted by the firewall configuration. Sugar parses $\mathcal{C}$ and returns *satisfiable* with a complete assignment solution of the problem, which is a packet that matches one of the firewall rules predicate. A $!\mathcal{C}$ represents the set of packets denied by the firewall configuration. With respect to

distributed firewalls, we consider each possible path to the migrating VM in source and destination data centers. Given the cloud calculus term, one can infer the firewall composition of the data center topology expressed in the syntax defined in Section 5.1. Therein, paths are composed using the $\oplus$ operator, i.e. $P_1 \oplus \cdots \oplus P_n$, where P_i consists of the serial composition of k firewalls, denoted by $F_1 \odot \cdots \odot F_k$.

5.3 Approach

In this section, we present our approach to verify firewall filtering preservation in dynamic cloud computing environment. First, we present firewall filtering preservation concept and summarize our assumptions. Then, we formally define firewall filtering preservation in source and destination data centers as well as for a migrating VM. Afterward, we elaborate on the CSP constrains, which satisfiabilities allow verifying the defined firewall filtering preservation. Finally, we describe the proposed verification procedure, and explain the interpretation of the outcome of Sugar constraint solver.

In dynamic cloud computing environment, a virtual machine may leave a data center D_1, called source data center, to be relocated in another data center D_2, called destination data center. During migration, the security enforcement rules located initially in D_1 should follow the VM. Thus, they have to be removed from the source data center, and then reinforced at the destination data center. Thus, it is very important to ensure each time, that the migrating VM firewall filtering requirements have not been compromised. Furthermore, as this involves modification of firewall filtering rules in both source and destination data centers, we also have to ensure that firewall filtering requirements

of the non-migrating virtual machines located therein have not been compromised. We suppose that all firewalls are anomaly-free, and decisions of all the rules are of type ”allow” and the default one is a ”deny” rule. Although, in case of existence of both deny and allow rules, one can use the algorithm defined in [64], which computes the effective representation of the firewall rules consisting of the equivalent allow rules. Furthermore, we assume the initial configurations before migration in both data centers are compliant with the pre-defined security policy.

5.3.1 Firewall Filtering Preservation

In the following, we formally define firewall filtering preservation in dynamic cloud computing environment. Let A^b_{src}, A^a_{src}, A_v be the accepted traffic in the source data center before migration, the accepted traffic in the source data center after migration, and the accepted traffic destined to the migrating VM v, respectively. Intuitively, firewall filtering is preserved in the source data center if the only difference between traffic accepted before and after migration is the one destined to the migrating VM v. This is defined formally as follows:

Definition 5.3.1. Firewall Filtering Preservation in Source Data Center

Firewall filtering is preserved in source data center if and only if for any path, we have $A^a_{src} = A^b_{src} \setminus A_v$ and $A_v \neq \varnothing$. □

Note here that we require $A_v \neq \varnothing$ otherwise, this will be a trivial case where no rule is migrated. In the destination data center, firewall filtering preservation is defined as follows:

Definition 5.3.2. Firewall Filtering Preservation in Destination Data Center

Firewall filtering is preserved in destination data center if and only if for any path, we have $A^b_{dst} =$

$A^a_{dst} \setminus A_v$ and $A_v \neq \varnothing$. □

Since we are assuming that firewall filtering requirements of the migrating VM are met in the source data center, its firewall filtering is preserved if the traffic accepted to that VM in the destination data center after migration and the source data center before migration are equal.

Definition 5.3.3. Firewall Filtering Preservation for the Migrating VM

Firewall filtering is preserved for the migrated VM if and only if for any path in the destination data center $A^a_{dst} \setminus A^b_{dst} = A^b_{src} \setminus A^a_{src}$. □

5.3.2 Mapping Firewall Filtering Preservation into CSP

In order to verify firewall filtering preservation in both data centers and for the migrating VM, we encode all firewall rules in both data centers as explained in Section 5.2 and use the aforementioned definitions in order to infer the corresponding equivalent constraint satisfiability problem.

Let $\mathcal{C}^b_{src}(P_i)$ (resp. $\mathcal{C}^b_{dst}(P_j)$) be the constraint that encodes the filtering conditions of the firewall at the source (resp. destination) data center before migration on path P_i (resp. P_j). The constraints $\mathcal{C}^b_{src}(P_i)$ and $\mathcal{C}^b_{dst}(P_j)$ represent the encoding in CSP of A^b_{src} and A^b_{dst}, respectively. Let $\mathcal{C}^a_{src}(P_i)$ (resp. $\mathcal{C}^a_{dst}(P_j)$) be the constraint that encodes the filtering conditions of the firewall at the source (resp. destination) data center after migration. The constraints $\mathcal{C}^a_{src}(P_i)$ and $\mathcal{C}^a_{dst}(P_j)$ represent the encoding in CSP of A^a_{src} and A^a_{dst}, respectively. Let $\mathcal{C}_v$ be the constraint that specifies the packets that are destined to the migrating VM v. According to the set theory, two sets A and B are equal, denoted $A = B$, if and only if $A \subseteq B$ and $B \subseteq A$. We use this concept in order to prove firewall filtering preservation as defined in Definition 5.3.1, Definition 5.3.2, and Definition 5.3.3

using CSP framework. From Definition 5.3.1, $A^a_{src} = A^b_{src} \setminus A_v$ if and only if $A^a_{src} \subseteq A^b_{src} \setminus A_v$ and $A^b_{src} \setminus A_v \subseteq A^a_{src}$. This condition holds if the following CSP problems are unsatisfiable for all paths:

$$\mathcal{C}^a_{src}(P_i) \wedge !(\mathcal{C}^b_{src}(P_i) \wedge !\mathcal{C}_v) \tag{1}$$

$$\mathcal{C}^b_{src}(P_i) \wedge !\mathcal{C}_v \wedge !C^a_{src}(P_i) \tag{2}$$

Equation (1) is equivalent to $(\mathcal{C}^a_{src}(P_i) \wedge !\mathcal{C}^b_{src}(P_i)) \vee (\mathcal{C}^a_{src}(P_i) \wedge \mathcal{C}_v)$.

The condition $A_v \neq \varnothing$ is verified if $\mathcal{C}^b_{src}(P_i) \wedge !\mathcal{C}^a_{src}(P_i)$ is satisfiable. Therefore, proving firewall filtering preservation in source data center is equivalent to prove for all paths that:

- $\mathcal{C}_1 = \mathcal{C}^b_{src}(P_i) \wedge !\mathcal{C}^a_{src}(P_i)$ is satisfiable
- $\mathcal{C}_2 = \mathcal{C}^a_{src}(P_i) \wedge !\mathcal{C}^b_{src}(P_i)$ is unsatisfiable
- $\mathcal{C}_3 = \mathcal{C}^a_{src}(P_i) \wedge \mathcal{C}_v$ is unsatisfiable
- $\mathcal{C}_4 = \mathcal{C}^b_{src}(P_i) \wedge !\mathcal{C}_v \wedge !\mathcal{C}^a_{src}(P_i)$ is unsatisfiable

From Definition 5.3.2, $A^b_{dst} = A^a_{dst} \setminus A_v$ if and only if $A^b_{dst} \subseteq A^a_{dst} \setminus A_v$ and $A^a_{dst} \setminus A_v \subseteq A^b_{dst}$. This condition holds if the following CSP problems are unsatisfiable for all paths:

$$\mathcal{C}^b_{dst}(P_j) \wedge !(\mathcal{C}^a_{dst}(P_j) \wedge !\mathcal{C}_v) \tag{3}$$

$$\mathcal{C}^a_{dst}(P_j) \wedge !\mathcal{C}_v \wedge !\mathcal{C}^b_{dst}(P_j) \tag{4}$$

Equation (3) is equivalent to $\mathcal{C}^b_{dst}(P_j) \wedge !\mathcal{C}^a_{dst}(P_j) \vee \mathcal{C}^b_{dst}(P_j) \wedge \mathcal{C}_v$. The unsatisfiability of formula

$\mathcal{C}^b_{dst}(P_j) \wedge \mathcal{C}_v$ states that before migration none of the rules concern v. This trivially holds thus, we do not consider it in the verification process. The condition $A_v \neq \varnothing$ is verified if $\mathcal{C}^a_{dst}(P_i) \wedge !\mathcal{C}^b_{dst}(P_i)$ is satisfiable. Thus, proving firewall filtering preservation in the destination data center is equivalent to prove for all paths that:

- $\mathcal{C}_5 = \mathcal{C}^a_{dst}(P_j) \wedge !\mathcal{C}^b_{dst}(P_j)$ is satisfiable.
- $\mathcal{C}_6 = \mathcal{C}^b_{dst}(P_j) \wedge !\mathcal{C}^a_{dst}(P_j)$ is unsatisfiable.
- $\mathcal{C}_7 = \mathcal{C}^a_{dst}(P_j) \wedge !\mathcal{C}_v \wedge !\mathcal{C}^b_{dst}(P_j)$ is unsatisfiable.

For Definition 5.3.3, $A^a_{dst} \setminus A^b_{dst} = A^b_{src} \setminus A^a_{src}$ hold if and only if $A^a_{dst} \setminus A^b_{dst} \subseteq A^b_{src} \setminus A^a_{src}$ and $A^b_{src} \setminus A^a_{src} \subseteq A^a_{dst} \setminus A^b_{dst}$. This condition holds if the following CSP problems are unsatisfiable for all paths P_j, with a reference path in the source data center P_{ref}:

$$\mathcal{C}_8 = \mathcal{C}^b_{src}(P_{ref}) \wedge !\mathcal{C}^a_{src}(P_{ref}) \wedge !(\mathcal{C}^a_{dst}(P_j) \wedge !\mathcal{C}^b_{dst}(P_j)) \tag{5}$$

$$\mathcal{C}_9 = \mathcal{C}^a_{dst}(P_j) \wedge !\mathcal{C}^b_{dst}(P_j) \wedge !(\mathcal{C}^b_{src}(P_{ref}) \wedge !\mathcal{C}^a_{src}(P_{ref})) \tag{6}$$

Thus, proving firewall filtering preservation for the migrating VM is equivalent to prove for all paths in destination data center that both $\mathcal{C}_8$ and $\mathcal{C}_9$ are unsatisfiable. The satisfiability of any one of them implies that there is discrepancy between the migrated rules from source data center and the rules migrated into the destination data center for path P_j. Figures 12, 13, 14 illustrate the verification approach, which consists of three steps: firewall filtering preservation in source data center, firewall filtering preservation in destination data center, and then firewall filtering preservation of the migrated VM. Note that the horizontal bar in this figure means that all conditions

have to hold before concluding on the firewall filtering preservation. For instance, in Figure 12, $\mathcal{C}_1$ has to be satisfiable and $\mathcal{C}_2$, $\mathcal{C}_3$, and $\mathcal{C}_4$ have to be unsatisfiable in order to conclude on the firewall filtering preservation in source data center.

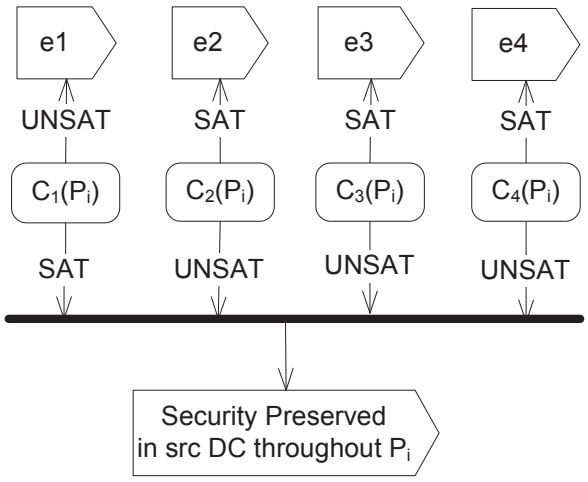

Figure 12: Step 1 for a Path P_i in Source Data Center

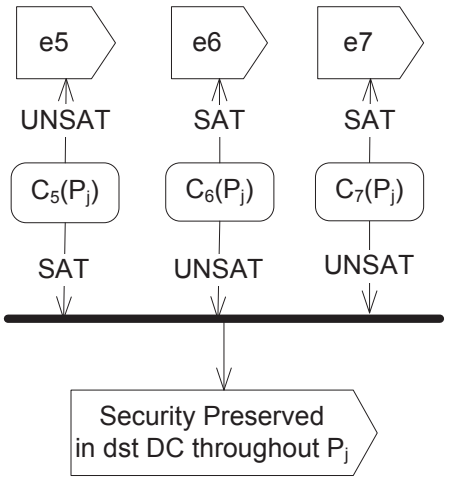

Figure 13: Step 2 for a Path P_j in Destination Data Center

The evaluation of these constraints is interpreted relatively to a given path in source or destination data center. In the case of a constraint satisfiability, the CSP solver provides a solution that can

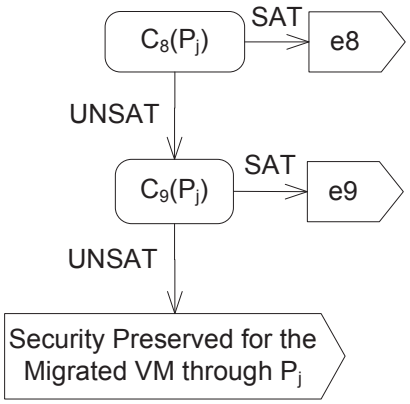

Figure 14: Step 3 for a Path P_j in Destination Data Center

be used to identify the problematic rule(s). The interpretation of the undesired outputs as identified in Figures 12, 13, 14 are summarized in Table 1.

Table 1: Interpretations of the Unexpected Constraints' Satisfaction Values

e1	$A^b_{src} \subseteq A^a_{src}$
e2	$A^b_{src} \subset A^a_{src}$
e3	$\exists p \mid p \in A_v$ and $p \in A^a_{src}$
e4	$\exists p \mid p \in A^b_{src}$ and $p \notin A_v$ and $p \notin A^a_{src}$
e5	$A^a_{dst} \subseteq A^b_{dst}$
e6	$A^a_{dst} \subset A^b_{dst}$
e7	$\exists p \mid p \in A^a_{dst}$ and $p \notin A_v$ and $p \notin A^b_{dst}$
e8	$A_{v,dst} \subset A_{v,src}$, some rules from source data center did not migrate
e9	$A_{v,src} \subset A_{v,dst}$, more rules than migrated in destination data center

5.4 Stateless Distributed Firewall Case Study

To better illustrate our approach, we present a case study consisting of two data centers $DC1$ and $DC2$ with distributed firewall settings as depicted in Figure 15, inspired from Amazon Elastic Compute Cloud (Amazon EC2) [8]. The service is built using a three-tier architecture: web, application, and database.

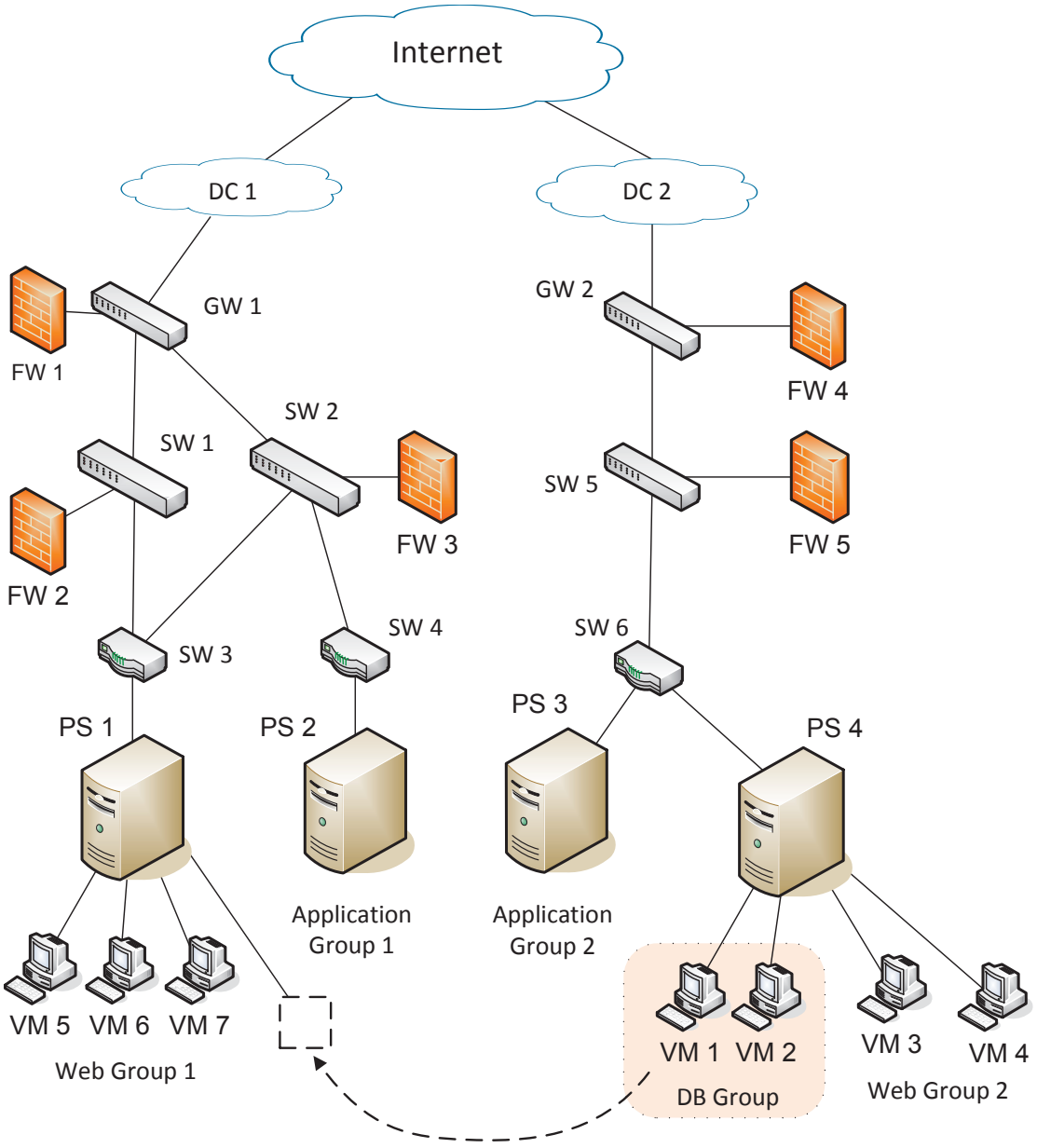

Figure 15: Stateless Distributed Firewall Case Study - Before Migration

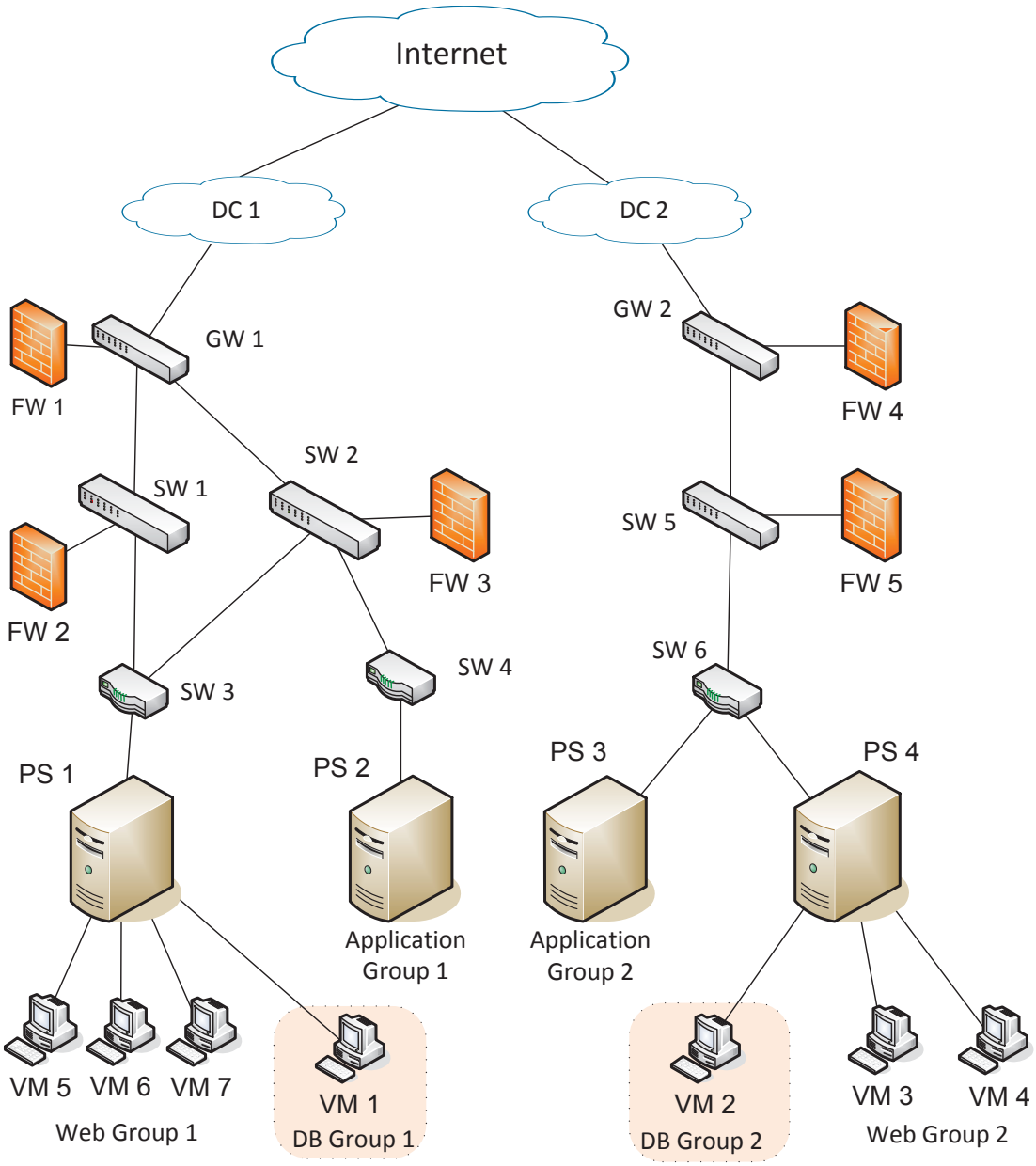

Figure 16: Stateless Distributed Firewall Case Study - After Migration

Table 2 summarizes the firewall rules in both data centers before migration. The need for virtual machine mobility across data centers, has been expressed by many providers as it serves several reasons including data center infrastructure maintenance, disaster avoidance, or data center expansion to address power, cooling, and space constraints. Even though the technology is not widespread at the moment, but we believe it is coming in the near future. In this case study, we suppose that *VM1* that belongs to the database group has to be migrated from the physical server $PS4$ in data center $DC2$ to $PS1$ in data center $DC1$. Therefore, traffic destined to application group one in $PS2$ has to traverse $FW1$ and $FW3$, whereas traffic destined to web group one and the recently created database group one containing *VM1*, can either traverse $FW1$ followed by $FW2$, or $FW1$ followed by $FW3$. Thus, all firewall configurations along the paths to the destination physical server $PS1$ in data center $DC1$ have to be updated as the result of this migration. This means that after *VM1* migration, the port 3306 in $FW1$ that was previously blocked, has to be opened to allow the traffic for database group one. In addition, all previously allowed traffic to database group should be accepted through both $FW2$ and $FW3$. Moreover, the required updates have to be done for the firewalls in the source being $FW4$ and $FW5$. In this case, there is no change in $FW4$, but the rules concerning *VM1* have to be removed from $FW5$. As it is shown in Figure 16, after migration there is one database group in each data center, one contains *VM1*, and the other contains *VM2*.

In order to demonstrate the applicability of our verification approach, we consider three scenarios:

Scenario 1 - Migration Error 1. The administrator correctly updated $FW1$, $FW2$, and $FW5$, but omitted to add the rules to $FW3$. In such a scenario, $FW3$ rules after migration will be the same

Table 2: Firewall Rules in Data Centers $DC1$ and $DC2$ - Before Migration

FW1						
1.	TCP	*.*.*.*	ANY	*.*.*.*	80	Allow
2.	TCP	*.*.*.*	ANY	*.*.*.*	443	Allow
3.	TCP	*.*.*.*	ANY	*.*.*.*	22	Allow
4.	TCP	*.*.*.*	ANY	*.*.*.*	8000	Allow
FW2						
1.	TCP	*.*.*.*	ANY	VM5,VM6,VM7	80	Allow
2.	TCP	*.*.*.*	ANY	VM5,VM6,VM7	443	Allow
3.	TCP	CorpIP	ANY	VM5,VM6,VM7	22	Allow
FW3						
1.	TCP	*.*.*.*	ANY	VM5,VM6,VM7	80	Allow
2.	TCP	*.*.*.*	ANY	VM5,VM6,VM7	443	Allow
3.	TCP	CorpIP	ANY	VM5,VM6,VM7	22	Allow
4.	TCP	VM3,VM4,VM5,VM6,VM7	ANY	AP1	8000	Allow
5.	TCP	CorpIP	ANY	AP1	22	Allow
FW4						
1.	TCP	*.*.*.*	ANY	*.*.*.*	80	Allow
2.	TCP	*.*.*.*	ANY	*.*.*.*	443	Allow
3.	TCP	*.*.*.*	ANY	*.*.*.*	22	Allow
4.	TCP	*.*.*.*	ANY	*.*.*.*	3306	Allow
5.	TCP	*.*.*.*	ANY	*.*.*.*	8000	Allow
FW5						
1.	TCP	*.*.*.*	ANY	VM3,VM4	80	Allow
2.	TCP	*.*.*.*	ANY	VM3,VM4	443	Allow
3.	TCP	CorpIP	ANY	**VM1**,VM2	22	Allow
4.	TCP	CorpIP	ANY	VM3,VM4	22	Allow
5.	TCP	CorpIP	ANY	AP2	22	Allow
6.	TCP	AP1	ANY	**VM1**,VM2	3306	Allow
7.	TCP	AP2	ANY	**VM1**,VM2	3306	Allow
8.	TCP	VM3,VM4,VM5,VM6,VM7	ANY	AP2	8000	Allow

as before.

Scenario 2 - Migration Error 2. The administrator correctly updated the rules in $FW1$, $FW2$, and $FW5$, but missed some rules in $FW3$.

Scenario 3 - Migration Error 3. The administrator correctly migrated the firewall rules to $FW1$, $FW2$, and $FW3$, but forgot to update $FW5$.

Scenario 4 - Correct Migration. The firewall rules are correctly migrated on every path of the network and are provided in Table 3. Note that $FW4$ do not need to be modified after migration.

In order to verify firewall filtering preservation, we translate the distributed firewall configuration for each scenario into CSP, and use Sugar SAT-solver to verify the satisfiability of the CSP constraints. The verification results for the four scenarios are summarized in Table 4. Therein, we

Table 3: Updated Firewall Rules in Data Centers $DC1$ and $DC2$ - After Migration

FW1						
1.	TCP	*.*.*.*	ANY	*.*.*.*	80	Allow
2.	TCP	*.*.*.*	ANY	*.*.*.*	443	Allow
3.	TCP	*.*.*.*	ANY	*.*.*.*	22	Allow
4.	TCP	*.*.*.*	ANY	*.*.*.*	8000	Allow
5.	**TCP**	*.*.*.*	**ANY**	*.*.*.*	**3306**	**Allow**
FW2						
1.	TCP	*.*.*.*	ANY	VM5,VM6,VM7	80	Allow
2.	TCP	*.*.*.*	ANY	VM5,VM6,VM7	443	Allow
3.	TCP	CorpIP	ANY	VM5,VM6,VM7	22	Allow
4.	TCP	CorpIP	ANY	**VM1**	22	Allow
5.	TCP	AP1	ANY	**VM1**	3306	Allow
6.	TCP	AP2	ANY	**VM1**	3306	Allow
FW3						
1.	TCP	*.*.*.*	ANY	VM5, VM6,VM7	80	Allow
2.	TCP	*.*.*.*	ANY	VM5, VM6,VM7	443	Allow
3.	TCP	CorpIP	ANY	VM5, VM6,VM7	22	Allow
4.	TCP	VM3,VM4,VM5,VM6,VM7	ANY	AP1	8000	Allow
5.	TCP	CorpIP	ANY	AP1	22	Allow
6.	TCP	CorpIP	ANY	**VM1**	22	Allow
7.	TCP	AP1	ANY	**VM1**	3306	Allow
8.	TCP	AP2	ANY	**VM1**	3306	Allow
FW5						
1.	TCP	*.*.*.*	ANY	VM3,VM4	80	Allow
2.	TCP	*.*.*.*	ANY	VM3,VM4	443	Allow
3.	TCP	CorpIP	ANY	VM2	22	Allow
4.	TCP	CorpIP	ANY	VM3,VM4	22	Allow
5.	TCP	CorpIP	ANY	AP2	22	Allow
6.	TCP	AP1	ANY	VM2	3306	Allow
7.	TCP	AP2	ANY	VM2	3306	Allow
8.	TCP	VM3,VM4,VM5,VM6,VM7	ANY	AP2	8000	Allow

show only the results for the path that has a firewall configuration error which are $FW1 \odot FW3$ for scenario one and scenarion two, and $FW4 \odot FW5$ for scenario four. Bold values in Table 4 show the constraints, which satisfactions are not as expected. When the solver returns satisfiable for a constraint expected to be unsatisfiable, a solution is provided that pinpoints one of the rules that makes security requirements fails. This indicates a possible error that should be investigated in order to correct the firewall configuration. In order to have an assessment of the performance over-

Table 4: Sugar CSP Solver Results for the Three Scenarios

	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$	$\mathcal{C}_4$	$\mathcal{C}_5$	$\mathcal{C}_6$	$\mathcal{C}_7$	$\mathcal{C}_8$	$\mathcal{C}_9$
Scen- 1	*SAT*	*UNSAT*	*UNSAT*	*UNSAT*	**UNSAT**	*UNSAT*	*UNSAT*	**SAT**	*UNSAT*
Scen- 2	*SAT*	*UNSAT*	*UNSAT*	*UNSAT*	*SAT*	*UNSAT*	*UNSAT*	**SAT**	*UNSAT*
Scen- 3	**UNSAT**	*UNSAT*	**SAT**	*UNSAT*	*SAT*	*UNSAT*	*UNSAT*	*UNSAT*	**SAT**
Scen- 4	*SAT*	*UNSAT*	*UNSAT*	*UNSAT*	*SAT*	*UNSAT*	*UNSAT*	*UNSAT*	*UNSAT*

head, we performed a set of experiments on an Intel Core i7 2.67 GHz processor with 12Gbytes of RAM. The verification performance depends on the total number of firewall rules. Table 5 summarizes the result in terms of CPU time for an increased number of rules, which is mostly due to increasing the number of VMs. The result shows that the CPU time consumption increases approximately linearly, also confirmed by [31].

Table 5: Performance Evaluation for Stateless FW

Number of VMs	Number of Rules	CPU Time (seconds)
5	22	0.278
50	202	0.324
500	2002	0.340
5000	20002	0.762
10000	40002	2.150

Chapter 6

IDS Monitoring and IPsec Protection Preservation

In a dynamic cloud computing environment, a VM may leave a physical machine called source host, to be relocated in another host, called destination host. During migration, the security rules that are associated with the migrating VM should follow the VM. Therefore, they should be removed from the source location, and applied at the destination location. Thus, it is very important to ensure each time that security of the migrating VM as well as of the other co-located VMs has not been compromised by the migration process. In this chapter, we consider two other security mechanisms, namely intrusion detection and prevention, and IPsec, which play an important role in the security of the cloud's infrastructure, and its resources, and VMs. We suppose that all IDSs and IPsec devices are correctly configured before migration, and that the initial configurations in both hosts are compliant with the pre-defined security policies. In this thesis, we consider an IDS architecture where a virtual appliance dedicated for intrusion monitoring, called security monitor

(SM), is attached to the hypervisors of the hosts. In addition, we assume having hardware IDS appliances connected to the hosts through network interfaces. It is the responsibility of the hypervisor to determine the decision of which part of the traffic should be monitored by the virtual appliance, and which one by its hardware counterpart. We also consider IPsec endpoints located at the edges of the data centers and the customer network.

6.1 Encoding IDS and IPsec Configuration in CSP

As far as the input format of IDS configuration files is concerned, we consider Snort [96] intrusion detection system, which is an open source IDS that is widely used in many networks. A Snort rule consists of two sections, a rule header and some rule options. The rule header contains criteria for matching a rule against data packets, and the action to be taken. The options part usually contains an alert message as well as information about the parts of the packet that should be used to generate the alert message. The options part may also contain additional criteria for matching a rule against data inside the packets. There are three major action directives that Snort supports when a packet matches a specified rule pattern: *pass*, *log*, or *alert*. Pass rules simply drop the packet. Log rules write the full packet to the logging routine. Alert rules generate an event notification using the user-specified method, and then log the full packet using the selected logging mechanism for later analysis. In the following, we present how we encode IDS configurations in Sugar.

The CSP variables are the set of integer variables V needed to encode the monitoring attributes of IDS rules. In order to represent an IP address, 4 integer variables within the range $[0, 255]$ are used. A source (resp. destination) IP address is represented by $\{sip_i\}_{1 \leq i \leq 4}$ (resp. $\{dip_i\}_{1 \leq i \leq 4}$). The

integer variable $pr \in [0, 255]$ represents the protocol number. We also define two other integer variables to encode the source and destination port numbers, respectively ps and pd within the range of values $[0, 65535]$. We encode the direction (ingress, egress or bidirectional) using an integer variable $dr \in [0, 1]$ so that 1 represents bidirectional traffic and 0 represents unidirectional traffic such that we switch the IP addresses and the ports from source to destination and vice versa to represent ingress or egress traffic. The action is encoded using a variable $act \in [0, 2]$ so that 0 represents *pass*, 1 represents *log*, and 2 represents *alert*. To encode the rule options we use a variable $opt \in [0, 40000]$ so that each value represents a unique option value. Note that here we assume an IDS which is enabled to recognize the signatures of maximum 40000 attacks. This number is only the maximum number that we are sure not to bypass in our case study. Our experiments show that we can cover up to 5000 attack signatures for the moment. Since it is possible to have more than one option, we encode them as a logical conjunction formula of all options. The action in the rule header is invoked only when all criteria in the options are true.

In the case of IDS rules the set of integer variables in CSP is $V = \{act, pr, sip1, sip2, sip3, sip4, ps, dr, dip1, dip2, dip3, dip4, pd, opt\}$. Each single IDS rule predicate p is encoded as a CSP constraint. The latter is a conjunctive logical formula over the variables in V with their corresponding values specified in the IDS rule. More precisely, a CSP constraint is written as $act = v_1 \wedge pr = v_2 \wedge sip1 = v_3 \cdots \wedge dip4 = v_{12} \wedge pd = v_{13} \wedge opt = v_{14}$ where v_i is to be replaced by the actual value in the corresponding IDS rule. The IDS is then encoded as a constraint $\mathcal{C}$ built as a disjunctive logical formula over all constraints of the IDS rules. Thus, the list of IDS rules $(p_n \rightarrow d_n).(p_{n-1} \rightarrow d_{n-1}). \cdots .nop$ are encoded as the logical formula $p_1 \vee p_2 \vee \cdots \vee p_n$. In Sugar syntax, this is denoted by *(or* $p_1 \ \ldots \ p_n$*)*.

In order to encode IPsec configuration in Sugar, we use a subset of the aforementioned CSP variables in order to encode the filtering attributes of IPsec rules. The representation of source (and destination) IP address, source (and destination) port, and protocol number are exactly the same as the presentation in IDS. For the IPsec protocols ESP and AH, we define a variable called $ipsec$ having the values 50 and 51, respectively. To encode the mode (transport or tunnel), we define a variable md, which values are in $\{0, 1\}$ such that 0 encodes transport mode and 1 encodes the tunnel mode. In terms of action field in IPsec, we use a variable act, which has values in $\{0, 1, 2\}$ such that 0 encodes *discard*, 1 encodes *bypass*, and 2 encodes *protect*. For the case of tunnel mode, the destination gateway is encoded by a variable gw and its values are in $\{1, m\}$ such that m is the maximum number of gateways in the network. Also a variable $param$ is defined to encode the authentication or cryptographic algorithms being used such as 3DES, MD5 and so on. Its values are in $\{1, n\}$ such that n is the number of authentication and cryptographic parameters available in the configuration. In the case of an empty gateway configuration, the corresponding constraint will be the truth value *FALSE*.

6.2 Approach

In this section, we present our approach to verify intrusion monitoring preservation and IPsec protection preservation properties in dynamic cloud computing environment. First, we define the meaning of intrusion monitoring preservation property and then derive the needed formulas to verify this property in source and destination hosts as well as for the migrating VM. Second, we present the concept of IPsec protection preservation, and then derive the corresponding formulas

to verify this property in source and destination data center, in customer network, and for the migrating virtual machine. Afterward, we elaborate on the CSP constraints, which satisfiabilities allow verifying the defined security preservation properties. Finally, we describe the proposed verification procedure, and explain the outcome of the Sugar solver. In dynamic cloud computing environment, a VM may leave a physical machine called source host, to be relocated in another host, called destination host. During migration, the security rules should follow the VM and thus, should be removed from the source location and applied at the destination location. Thus, it is very important to ensure each time that the security of the migrating VM as well as of the other co-located VMs have not been compromised by the migration process. We suppose that all IDSs and IPsec devices are correctly configured before migration, and that the initial configurations in both hosts are compliant with the pre-defined security policies.

6.2.1 Intrusion Monitoring Preservation

In the following, we define the meaning of intrusion monitoring preservation in dynamic cloud computing environment. Let M_s^b and M_s^a be the monitoring rules in the source host before and after migration, respectively. Let M_v be the monitoring rules with respect to the migrating VM v. Intuitively, monitoring is preserved in the source host if the only difference between monitoring policy before and after migration is the rules with regards to the migrating VM v. Note here that we require $M_v \neq \varnothing$ otherwise, this will be a trivial case where no rule is attached to the migrated VM.

Definition 6.2.1. Intrusion Monitoring Preservation in Source Host

Intrusion monitoring is preserved in source host if and only if, we have $M_s^a = M_s^b \setminus M_v$ and

$M_v \neq \varnothing$. □

Similarly for the destination host, we would like to verify intrusion monitoring preservation property.

Definition 6.2.2. Intrusion Monitoring Preservation in Destination Host

Intrusion monitoring is preserved in destination host if and only if, we have $M_d^b = M_d^a \setminus M_v$ and $M_v \neq \varnothing$ and $M_s^b \subseteq M_d^a$.

Note that Definition 6.2.2 contains an extra condition ($M_s^b \subseteq M_d^a$) if compared to Definition 6.2.1. This condition is important as it is used to compare IDS capabilities of the two hosts in order to ensure that the IDS signatures in destination are at least equal or more up-to-date than those of the source host.

For the migrating VM, we need to ensure that the migrated VM is monitored at least at the same level (or more) in the destination host after migration. Thus, intrusion monitoring is preserved for the migrating VM if the monitoring policy for that VM in the destination host after migration is a superset of those at the source host before migration. This is stated in the following definition.

Definition 6.2.3. Intrusion Monitoring Preservation for the Migrating VM

Intrusion monitoring is preserved for the migrated VM if and only if we have $M_s^b \setminus M_s^a \subseteq M_d^a \setminus M_d^b$. □

6.2.2 Mapping Intrusion Monitoring Preservation into CSP

In order to verify monitoring preservation in both hosts and for the migrating VM, we encode all IDSs rules in both hosts as explained in Section 6.1, and use the aforementioned definitions in

order to infer the corresponding equivalent constraint satisfaction problems.

Let $\mathcal{C}_s^b$ (resp. $\mathcal{C}_d^b$) be the constraint that encodes the monitoring conditions of the IDS at the source (resp. destination) host before migration by monitoring appliance (either physical or virtual appliances). The constraints $\mathcal{C}_s^b$ and $\mathcal{C}_d^b$ represent the encoding in CSP of M_s^b and M_d^b, respectively. Let $\mathcal{C}_s^a$ (resp. $\mathcal{C}_d^a$) be the constraint that encodes the monitoring conditions of the IDS at the source (resp. destination) host after migration. The constraints $\mathcal{C}_s^a$ and $\mathcal{C}_d^a$ represent the encoding in CSP of M_s^a and M_d^a, respectively. Let $\mathcal{C}_v$ be the constraint that specifies the monitoring policy regarding to the migrating VM v. According to the set theory, two sets A and B are equal, denoted $A = B$, if and only if $A \subseteq B$ and $B \subseteq A$. We use this concept in order to prove monitoring preservation as defined in Definition 6.2.1, Definition 6.2.2, and Definition 6.2.3 using CSP framework. From Definition 6.2.1, $M_s^a = M_s^b \setminus M_v$ if and only if $M_s^a \subseteq M_s^b \setminus M_v$ and $M_s^b \setminus M_v \subseteq M_s^a$. These conditions hold if the following CSP problems are unsatisfiable:

$$\mathcal{C}_s^a \wedge !(\mathcal{C}_s^b \wedge !\mathcal{C}_v) \tag{7}$$

$$\mathcal{C}_s^b \wedge !\mathcal{C}_v \wedge !C_s^a \tag{8}$$

Equation (7) is equivalent to $(\mathcal{C}_s^a \wedge !\mathcal{C}_s^b) \vee (\mathcal{C}_s^a \wedge \mathcal{C}_v)$. The condition $M_v \neq \varnothing$ is verified if $\mathcal{C}_s^b \wedge !\mathcal{C}_s^a$ is satisfiable. Therefore, proving monitoring preservation in source host is equivalent to prove that:

- $\mathcal{C}_1 = \mathcal{C}_s^b \wedge !\mathcal{C}_s^a$ is satisfiable
- $\mathcal{C}_2 = \mathcal{C}_s^a \wedge !\mathcal{C}_s^b$ is unsatisfiable
- $\mathcal{C}_3 = \mathcal{C}_s^a \wedge \mathcal{C}_v$ is unsatisfiable

- $\mathcal{C}_4 = \mathcal{C}_s^b \wedge !\mathcal{C}_v \wedge !\mathcal{C}_s^a$ is unsatisfiable

From Definition 6.2.2, $M_d^b = M_d^a \setminus M_v$ if and only if $M_d^b \subseteq M_d^a \setminus M_v$ and $M_d^a \setminus M_v \subseteq M_d^b$. These conditions hold if the following CSP problems are unsatisfiable:

$$\mathcal{C}_d^b \wedge !(\mathcal{C}_d^a \wedge !\mathcal{C}_v) \tag{9}$$

$$\mathcal{C}_d^a \wedge !\mathcal{C}_v \wedge !\mathcal{C}_d^b \tag{10}$$

Equation (9) is equivalent to $(\mathcal{C}_d^b \wedge !\mathcal{C}_d^a) \vee (\mathcal{C}_d^b \wedge \mathcal{C}_v)$. The unsatisfiability of formula $\mathcal{C}_d^b \wedge \mathcal{C}_v$ states that before migration none of the rules concern v. This trivially holds therefore, we do not consider it in the verification process. The condition $M_v \neq \varnothing$ is verified if $\mathcal{C}_d^a \wedge !\mathcal{C}_d^b$ is satisfiable. From the same definition, condition $M_s^b \subseteq M_d^a$ holds if the following CSP problem is unsatisfiable:

$$\mathcal{C}_s^b \wedge !\mathcal{C}_d^a \tag{11}$$

Thus, proving intrusion monitoring preservation in the destination host is equivalent to prove that:

- $\mathcal{C}_5 = \mathcal{C}_d^a \wedge !\mathcal{C}_d^b$ is satisfiable.
- $\mathcal{C}_6 = \mathcal{C}_d^b \wedge !\mathcal{C}_d^a$ is unsatisfiable.
- $\mathcal{C}_7 = \mathcal{C}_d^a \wedge !\mathcal{C}_v \wedge !\mathcal{C}_d^b$ is unsatisfiable.
- $\mathcal{C}_8 = \mathcal{C}_s^b \wedge !\mathcal{C}_d^a$ is unsatisfiable.

From Definition 6.2.3, $M_s^b \setminus M_s^a \subseteq M_d^a \setminus M_d^b$ hold if we have:

- $\mathcal{C}_9 = \mathcal{C}_s^b \wedge !\mathcal{C}_s^a \wedge !(\mathcal{C}_d^a \wedge !\mathcal{C}_d^b)$ is unsatisfiable.

Thus, the unsatisfiability of $\mathcal{C}_9$ implies that the monitored traffic in source host targeting the migrating VM is at least equal or included in the traffic monitored in destination host. More precisely, this provides a proof that the IDS signatures at the destination are always equal or more updated than those in the source.

Figures 17-19 illustrate our verification steps for IDS. Note that the horizontal bar in the figures means that all conditions have to hold before concluding on the monitoring preservation. For instance, in Figure 17, $\mathcal{C}_1$ has to be satisfiable and $\mathcal{C}_2$, $\mathcal{C}_3$, and $\mathcal{C}_4$ have to be unsatisfiable in order to conclude on the monitoring preservation in source host.

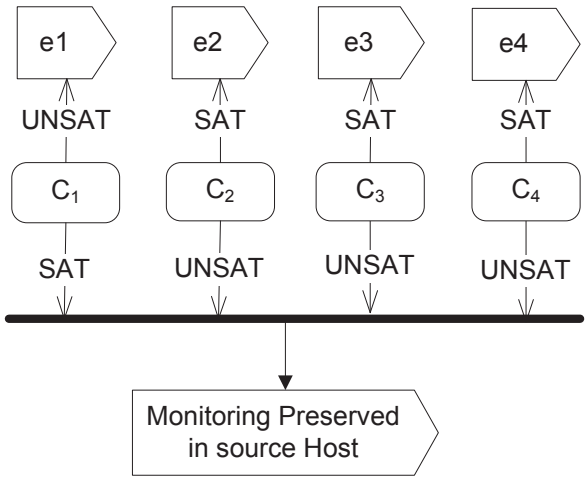

Figure 17: Intrusion Monitoring Verification in Source Host

In the case of a constraint is satisfiable, the CSP solver provides a solution that can be used to identify the problematic rule(s). The interpretation of the undesired satisfiability outputs as identified in Figures 17-19 are summarized in Table 6, where r represents a monitoring rule.

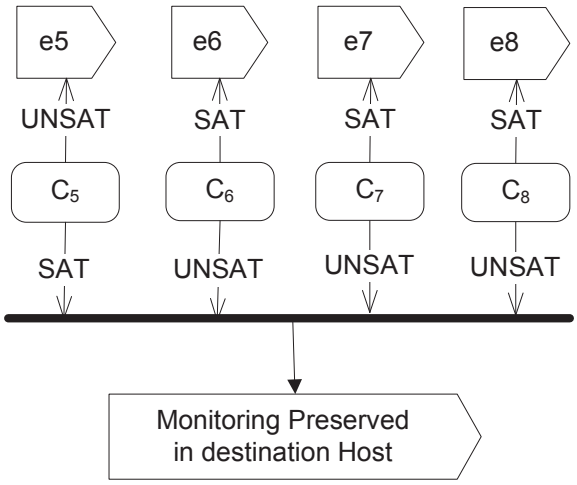

Figure 18: Intrusion Monitoring Verification in Destination Host

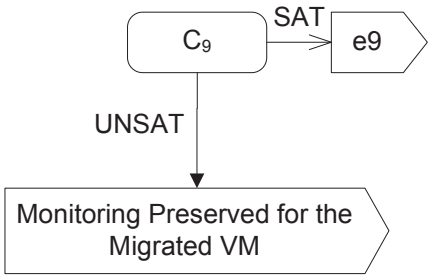

Figure 19: Intrusion Monitoring Verification for the Migrating VM

Table 6: Interpretations of the Unexpected Constraints' Satisfaction Values for IDS

Error	Interpretation
e1	$M_s^b \subseteq M_s^a$
e2	$M_s^b \subset M_s^a$
e3	$M_v \cap M_s^a \neq \varnothing$
e4	$\exists r \mid r \in M_s^b$ and $r \notin M_v$ and $r \notin M_s^a$
e5	$M_d^a \subseteq M_d^b$
e6	$M_d^a \subset M_d^b$
e7	$\exists r \mid r \in M_d^a$ and $r \notin M_v$ and $r \notin M_d^b$
e8	$M_d^a \subset M_s^b$, signatures at destination IDS less than the source
e9	$M_v^d \subset M_v^s$, some rules from source host are not at destination

6.2.3 IPsec Protection Preservation

IPsec functions will be correctly executed only if the related policies are correctly specified and configured at both sites establishing the tunnel. IPsec configuration spans multiple devices including the VM and the servers on the corporate network for authentication, and on the gateways in the source and destination data centers as well as at the customer side. Let E_l^b and E_l^a be the IPsec protected traffic in the location l, $l \in \{s, d, cust\}$ before and after migration, respectively, where s is for source, d is for destination, and $cust$ is for customer network. Let E_v^t be the IPsec traffic concerning the virtual machine v at $t \in \{a, b\}$. Let $E_{GW_d,v}$ be the IPsec traffic between the destination data center and the migrating VM v.

Non-Migrating Virtual Machines. With respect to IPsec protection in the data centers, we should verify that the IPsec configurations of all non-migrating VMs in the source as well as in the destination data centers have not been modified after the migration of v.

Definition 6.2.4. IPsec Protection Preservation in the Data Centers and Customer Network In source data center, in destination data center, and in the customer network, IPsec protection is preserved if we have respectively:

- $E_s^a = E_s^b \setminus E_v^b$
- $E_d^b = E_d^a \setminus E_v^a$
- $E_{cust}^b \setminus E_v^b = E_{cust}^a \setminus E_v^a$ □

Migrating Virtual Machine. With respect to the IPsec traffic of the migrating VM v, we should verify that it is protected equally before and after migration. This implies that the verification should be performed taking into account both the cloud providers and customer gateways.

Intuitively, we need first to verify that the IPsec rules that were added to the destination data center gateway after migration are exactly the same as the ones deleted from the source data center gateway. This holds since no modification is required for the rules' attributes that are on the side of the cloud provider. Secondly, we need to verify at the customer side that the IPsec rules for v have been correctly updated such that the cryptographic parameters have not been changed, and the gateway's IP at the destination data center has correctly replaced the gateway's IP at the source data center after migration. The following definition states formally the meaning of IPsec protection preservation for the migrating VM.

Definition 6.2.5. IPsec Protection Preservation for the Migrating VM For the migrating VM, IPsec protection is preserved if followings hold:

- $E_d^a \setminus E_d^b = E_s^b \setminus E_s^a$
- Independently of the gateways, we have $E_{cust}^a \setminus E_{cust}^b = \varnothing$
- Considering the gateways, we have $E_{cust}^a \setminus E_{cust}^b \subseteq E_{GW_d,v}$ □

6.2.4 Mapping IPsec Protection Preservation into CSP

In this section, we describe the constraints that we built in order to answer the verification problem. Let $\mathcal{C}_{E,l}^t$ be the constraint that encodes the IPsec-enabled appliances configurations where $t \in \{a, b\}$ and $l \in \{s, d, cust\}$. Let $\mathcal{C}_v$ be the constraint encoding the condition that considered traffic corresponds to the virtual machine v. Let $\mathcal{C}'^t_{E,cust}$ be the constraint that encodes the traffic protected independently of the gateways IP involved in the IPsec tunnel at the customer side. Let $\mathcal{C}_{GW_d,v}$ be

the constraint for encoding the traffic between the destination data center gateway GW_d and v. As we are interested in disregarding all encrypted traffic concerning v, we can encode both E_v^b and E_v^a as $\mathcal{C}_v$.

According to Definition 6.2.4 and set theory for the source data center, IPsec protection is preserved if we have $E_s^a \subseteq E_s^b \setminus E_v^b$ and $E_s^b \setminus E_v^b \subseteq E_s^a$. This is equivalent to verify that the following constraints are unsatisfiable:

$$\mathcal{C}_{E,s}^b \wedge \neg \mathcal{C}_{E,s}^a \wedge \neg \mathcal{C}_v \tag{12}$$

$$\mathcal{C}_{E,s}^a \wedge \neg (C_{E,s}^b \wedge \neg \mathcal{C}_v) =$$

$$[C_{E,s}^a \wedge \neg \mathcal{C}_{E,s}^b] \vee [C_{E,s}^a \wedge \mathcal{C}_v] \tag{13}$$

In addition to these conditions, we also add another constraint in order to ensure that there is actually IPsec rules for the migrating VM such that $C_{E,s}^b \wedge \neg \mathcal{C}_{E,s}^a$ is satisfiable. In summary, the constraints that should be verified to ensure IPsec protection preservation in source data center are as follows:

- $\mathcal{C}_{10} = C_{E,s}^b \wedge \neg \mathcal{C}_{E,s}^a$ is satisfiable
- $\mathcal{C}_{11} = C_{E,s}^a \wedge \neg \mathcal{C}_{E,s}^b$ is unsatisfiable
- $\mathcal{C}_{12} = C_{E,s}^a \wedge \mathcal{C}_v$ is unsatisfiable
- $\mathcal{C}_{13} = C_{E,s}^b \wedge \neg \mathcal{C}_{E,s}^a \wedge \neg \mathcal{C}_v$ is unsatisfiable

For the destination data center (Definition 6.2.4), IPsec protection is preserved if $E_d^b \subseteq E_d^a \setminus E_v^a$ and

$E^a_d \setminus E^a_v \subseteq E^b_d$. This is equivalent to verify that the following constraints are unsatisfiable:

$$\mathcal{C}^b_{E,d} \wedge \neg(C^a_{E,d} \wedge \neg\mathcal{C}_v) =$$

$$[C^b_{E,d} \wedge \neg\mathcal{C}^a_{E,d}] \vee [C^b_{E,d} \wedge \mathcal{C}_v] \tag{14}$$

$$\mathcal{C}^a_{E,d} \wedge \neg\mathcal{C}^b_{E,d} \wedge \neg\mathcal{C}_v \tag{15}$$

From Equation 14 the unsatisfiability of formula $C^b_{E,d} \wedge \mathcal{C}_v$ states that in destination data center before migration, none of the rules concern v. This trivially holds so we do not consider it in the verification process. In addition to these conditions, we also add another constraint in order to ensure that there is actually IPsec rules for the migrating VM at the destination data center such that $C^a_{E,d} \wedge \neg\mathcal{C}^b_{E,d}$ is satisfiable. In summary, the constraints that should be verified to ensure IPsec protection preservation in destination data center are as follows:

- $\mathcal{C}_{14} = C^a_{E,d} \wedge \neg\mathcal{C}^b_{E,d}$ is satisfiable
- $\mathcal{C}_{15} = C^b_{E,d} \wedge \neg\mathcal{C}^a_{E,d}$ is unsatisfiable
- $\mathcal{C}_{16} = C^a_{E,d} \wedge \neg\mathcal{C}^b_{E,d} \wedge \neg\mathcal{C}_v$ is unsatisfiable

For the customer network in Definition 6.2.4, we have to verify that $E^b_{cust} \setminus E^b_v \subseteq E^a_{cust} \setminus E^a_v$ and $E^a_{cust} \setminus E^a_v \subseteq E^b_{cust} \setminus E^b_v$. This is equivalent to verify that the following constraints are unsatisfiable:

$$\mathcal{C}^b_{E,cust} \wedge \neg\mathcal{C}_v \wedge \neg(C^a_{E,cust} \wedge \neg\mathcal{C}_v) \tag{16}$$

$$\mathcal{C}^a_{E,cust} \wedge \neg\mathcal{C}_v \wedge \neg(C^b_{E,cust} \wedge \neg\mathcal{C}_v) \tag{17}$$

After expanding the above constraints using the same approach, we obtain the following:

- $\mathcal{C}_{17} = C^{b}_{E,cust} \wedge \neg\mathcal{C}_{v} \wedge \neg\mathcal{C}^{a}_{E,cust}$ is unsatisfiable
- $\mathcal{C}_{18} = C^{a}_{E,cust} \wedge \neg\mathcal{C}_{v} \wedge \neg\mathcal{C}^{b}_{E,cust}$ is unsatisfiable

We have to verify the satisfiability of the following constraints to make sure that there are changes before and after migration, which means that the unsatisfiability of the above constraints is not a trivial case:

- $\mathcal{C}_{19} = C^{b}_{E,cust} \wedge \neg\mathcal{C}^{a}_{E,cust}$ is satisfiable
- $\mathcal{C}_{20} = C^{a}_{E,cust} \wedge \neg\mathcal{C}^{b}_{E,cust}$ is satisfiable

For the migrating VM (Definition 6.2.5), three conditions have to be verified. The first one, from equality $E^{a}_{d} \setminus E^{b}_{d} = E^{b}_{s} \setminus E^{a}_{s}$ and set theory, is equivalent to verify that: $E^{a}_{d} \setminus E^{b}_{d} \subseteq E^{b}_{s} \setminus E^{a}_{s}$ and $E^{b}_{s} \setminus E^{a}_{s} \subseteq E^{a}_{d} \setminus E^{b}_{d}$. This is equivalent to verify that:

- $\mathcal{C}_{21} = C^{a}_{E,d} \wedge \neg\mathcal{C}^{b}_{E,d} \wedge \neg(C^{b}_{E,s} \wedge \neg\mathcal{C}^{a}_{E,s})$ is unsatisfiable
- $\mathcal{C}_{22} = C^{b}_{E,s} \wedge \neg\mathcal{C}^{a}_{E,s} \wedge \neg(C^{a}_{E,d} \wedge \neg\mathcal{C}^{b}_{E,d})$ is unsatisfiable

The second condition, from equality $E^{a}_{cust} \setminus E^{b}_{cust} = \varnothing$ and set theory, is equivalent to verify that:

- $\mathcal{C}_{23} = C'^{a}_{E,cust} \wedge \neg\mathcal{C}'^{b}_{E,cust}$ is unsatisfiable

Finally, from $E^{a}_{cust} \setminus E^{b}_{cust} \subseteq E_{GW_d,v}$ and set theory, we have:

$$\mathcal{C}^{a}_{E,cust} \wedge \neg\mathcal{C}^{b}_{E,cust} \wedge \neg\mathcal{C}_{GW_d,v} \tag{18}$$

Unsatisfiability of above constraint is equivalent to:

- $\mathcal{C}_{24} = C^a_{E,cust} \wedge \neg \mathcal{C}^b_{E,cust}$ is satisfiable
- $\mathcal{C}_{25} = C^a_{E,cust} \wedge \neg \mathcal{C}^b_{E,cust} \wedge \neg \mathcal{C}_{GW_d,v}$ is unsatisfiable

Figures 20-22 illustrates the verification steps for IPsec. In the case of a constraint satisfiability, the CSP solver provides a solution that can be used to identify the problematic rule(s).

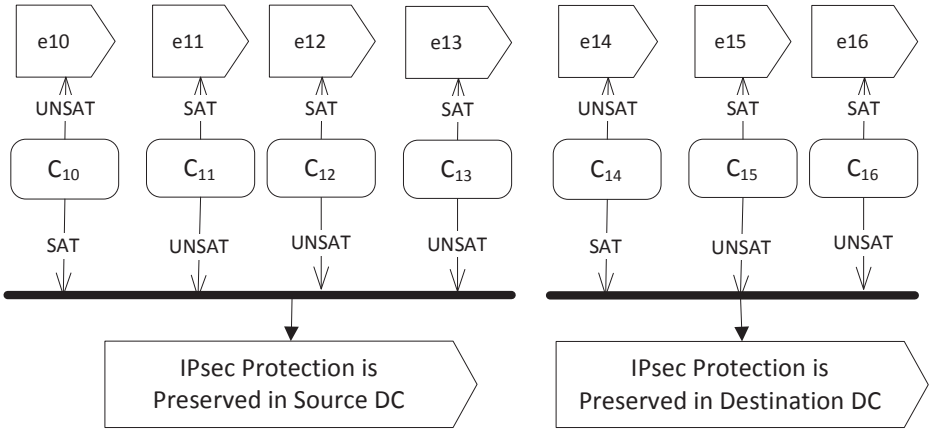

Figure 20: IPsec Protection Verification in Source and Destination Data Centers

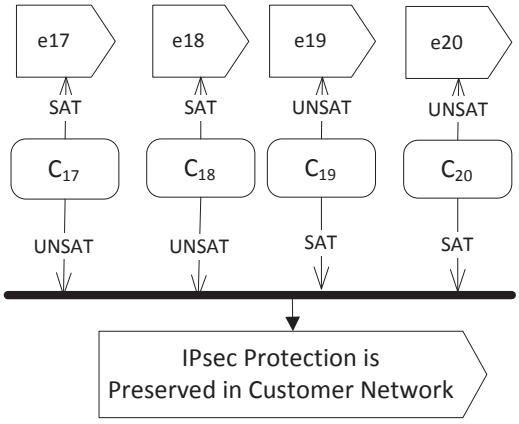

Figure 21: IPsec Protection Verification in Customer Network

The interpretation of the unexpected outputs as identified in Figures 20-22 are summarized in Table 7, where p represents a packet.

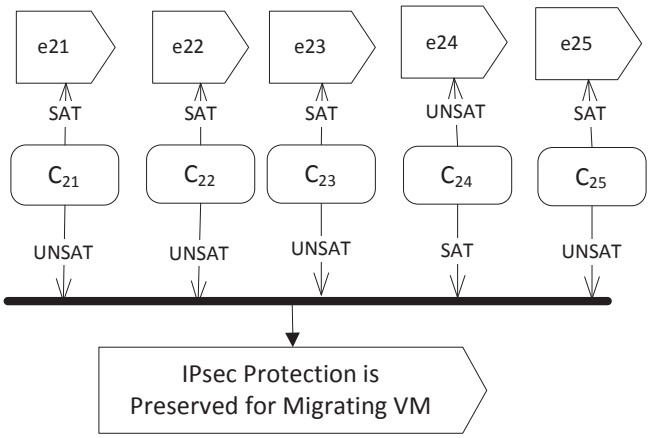

Figure 22: IPsec Protection Verification for the Migrating VM

Table 7: Interpretations of the Unexpected Constraints' Satisfaction Values for IPsec

Error	Interpretation
e10	$E_s^b \subseteq E_s^a$
e11	$E_s^b \subset E_s^a$
e12	$\exists p \mid p \in E_v$ and $p \in E_s^a$
e13	$\exists p \mid p \in E_s^b$ and $p \notin E_v$ and $p \notin E_s^a$
e14	$E_d^a \subseteq E_d^b$
e15	$E_d^a \subset E_d^b$
e16	$\exists p \mid p \in E_d^a$ and $p \notin E_v$ and $p \notin E_d^b$
e17	$\exists p \mid p \in E_{cust}^b$ and $p \notin E_v$ and $p \notin E_{cust}^a$
e18	$\exists p \mid p \in E_{cust}^a$ and $p \notin E_v$ and $p \notin E_{cust}^b$
e19	$E_{cust}^b \subseteq E_{cust}^a$
e20	$E_{cust}^a \subseteq E_{cust}^b$
e21	$E_{v,s} \subset E_{v,d}$, more rules than migrated in destination DC
e22	$E_{v,d} \subset E_{v,s}$, some rules from source DC did not migrate
e23	$E_{cust}^b \subset E_{cust}^a$, independent of the gateway
e24	similar to e20
e25	$\exists p \mid p \in E_{cust}^a$ and $p \notin E_{GW_{d,v}}$ and $p \notin E_{cust}^b$

6.3 IDS and IPsec Case Study

In this section, we apply our approach on a case study to demonstrate the usefulness and applicability of our approach. The case study consists of a cloud computing model with two physical hosts H1 and H2 located in two different data centers, where IPsec gateways and IDS appliances are deployed as depicted in Figure 23. Furthermore, we assume several virtual machines deployed in these data centers.

Table 8 provides a subset of the IDS rules before migration. The IPSec policy before and after migration are given in Table 10 and 11, respectively. We suppose that for the sake of load balancing, VM4 has to be migrated from the H1 to H2. The traffic destined to VM4 after migration will be monitored by the physical intrusion detection IDS2 and the virtual security monitor SM2 whereas, it was monitored by IDS1 and SM1 before migration. Thus, all IDSs configurations have to be updated after migration.

We propose to provide migration scenarios where the IPsec and IDS configurations after migration are not correctly updated in various enforcement endpoints. With respect to IDS, we consider the following scenarios:

Scenario 1 - Error 1. The administrator correctly updated SM2 but left SM1 same as before migration.

Scenario 2 - Error 2. The administrator correctly updated SM1 but missed some rules in the configuration of SM2.

Scenario 3 - Correct Migration. The rules are correctly migrated for every intrusion detection of the network and are provided in Table 9.

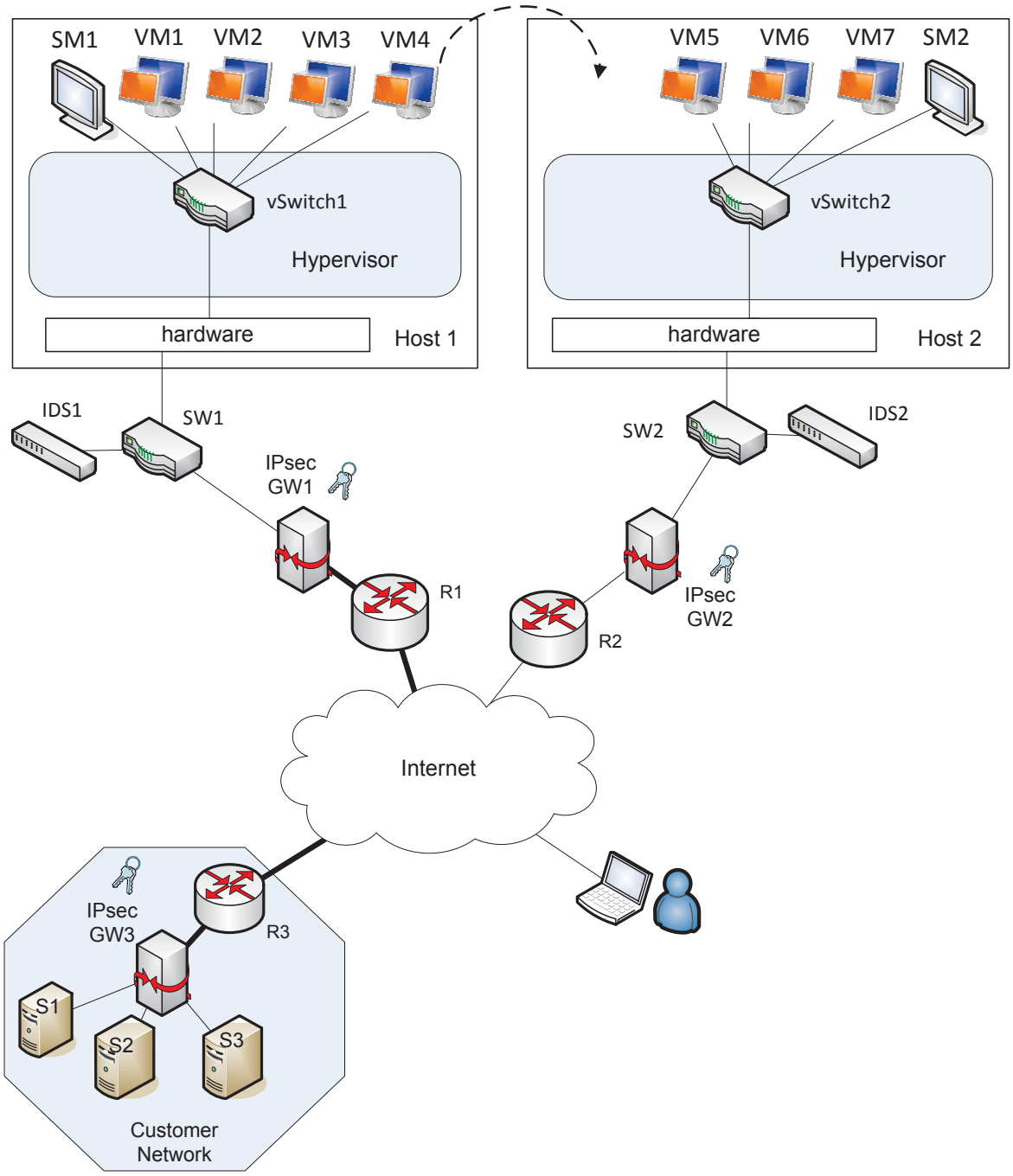

Figure 23: IDS and IPsec Case Study

Table 8: Sample IDS Rules in H1 and H2 - Before Migration

IDS1								
1.	alert	TCP	Ext	ANY	->	H1	[135:139, 445,1025]	(msg:"E2[rb] SHELLCODE x86 0x90 unicode NOOP"; content:"|90 90 90 90 90 90 90 90 90 90|"; ... ; rev:1;)
2.	alert	TCP	Ext	ANY	->	H1	445	(msg: "E2[rb] NETBIOS SMB-DS IPC$ unicode share access"; flow:established,to_server; ... ; sid:22466; rev:7;)
3.	alert	TCP	Ext	!20	->	H1	ANY	(msg:"E3[rb] BLEEDING-EDGE Malware Windows executable sent from remote host"; content: "MZ"; ... ; rev:3;)
4.	alert	UDP	H1	ANY	->	Ext	69	(msg:"E3[rb] TFTP GET from external source"; ... ; rev:1;)
SM1								
1.	alert	TCP	H1	ANY	->	VM1, VM2, VM3, **VM4**	[135:139, 445,1025]	(msg:"E2[rb] SHELLCODE x86 0x90 unicode NOOP"; content:"|90 90 90 90 90 90 90 90 90 90|"; ... ; rev:1;)
2.	alert	TCP	H1	!20	->	VM1, VM2, VM3, **VM4**	ANY	(msg:"E3[rb] BLEEDING-EDGE Malware Windows executable sent from remote host"; content: "MZ"; ... ; rev:3;)
IDS2								
1.	alert	TCP	Ext	ANY	->	H2	[135:139, 445,1025]	(msg:"E2[rb] SHELLCODE x86 0x90 unicode NOOP"; content:"|90 90 90 90 90 90 90 90 90 90|"; ... ; rev:1;)
2.	alert	TCP	Ext	ANY	->	H2	445	(msg: "E2[rb] NETBIOS SMB-DS IPC$ unicode share access"; flow:established,to_server; ... ; sid:22466; rev:7;)
3.	alert	TCP	Ext	!20	->	H2	ANY	(msg:"E3[rb] BLEEDING-EDGE Malware Windows executable sent from remote host"; content: "MZ"; ... ; rev:3;)
SM2								
1.	alert	TCP	H2	ANY	->	VM5, VM6, VM7	[135:139, 445,1025]	(msg:"E2[rb] SHELLCODE x86 0x90 unicode NOOP"; content:"|90 90 90 90 90 90 90 90 90 90|"; ... ; rev:1;)
2.	alert	TCP	H2	!20	->	VM5, VM6, VM7	ANY	(msg:"E3[rb] BLEEDING-EDGE Malware Windows executable sent from remote host"; content: "MZ"; ... ; rev:3;)

Table 9: Updated IDS Rules in H1 and H2- After Migration

IDS1								
1.	alert	TCP	Ext	ANY	->	H1-**VM4**	[135:139, 445,1025]	(msg:"E2[rb] SHELLCODE x86 0x90 unicode NOOP"; content:"|90 90 90 90 90 90 90 90 90 90|"; ... ; rev:1;)
2.	alert	TCP	Ext	ANY	->	H1-**VM4**	445	(msg: "E2[rb] NETBIOS SMB-DS IPC$ unicode share access"; flow:established,to_server; ... ; sid:22466; rev:7;)
3.	alert	TCP	Ext	!20	->	H1-**VM4**	ANY	(msg:"E3[rb] BLEEDING-EDGE Malware Windows executable sent from remote host"; content: "MZ"; ... ; rev:3;)
4.	alert	UDP	H1-**VM4**	ANY	->	Ext	69	(msg:"E3[rb] TFTP GET from external source"; ... ; rev:1;)
SM1								
1.	alert	TCP	H1-**VM4**	ANY	->	VM1, VM2, VM3	[135:139, 445,1025]	(msg:"E2[rb] SHELLCODE x86 0x90 unicode NOOP"; content:"|90 90 90 90 90 90 90 90 90 90|"; ... ; rev:1;)
2.	alert	TCP	H1-**VM4**	!20	->	VM1, VM2, VM3	ANY	(msg:"E3[rb] BLEEDING-EDGE Malware Windows executable sent from remote host"; content: "MZ"; ... ; rev:3;)
IDS2								
1.	alert	TCP	Ext	ANY	->	H2+**VM4**	[135:139, 445,1025]	(msg:"E2[rb] SHELLCODE x86 0x90 unicode NOOP"; content:"|90 90 90 90 90 90 90 90 90 90|"; ... ; rev:1;)
2.	alert	TCP	Ext	ANY	->	H2+**VM4**	445	(msg: "E2[rb] NETBIOS SMB-DS IPC$ unicode share access"; flow:established,to_server; ... ; sid:22466; rev:7;)
3.	alert	TCP	Ext	!20	->	H2+**VM4**	ANY	(msg:"E3[rb] BLEEDING-EDGE Malware Windows executable sent from remote host"; content: "MZ"; ... ; rev:3;)
SM2								
1.	alert	TCP	H2+**VM4**	ANY	->	VM5, VM6, VM7, **VM4**	[135:139, 445,1025]	(msg:"E2[rb] SHELLCODE x86 0x90 unicode NOOP"; content:"|90 90 90 90 90 90 90 90 90 90|"; ... ; rev:1;)
2.	alert	TCP	H2+**VM4**	!20	->	VM5, VM6, VM7, **VM4**	ANY	(msg:"E3[rb] BLEEDING-EDGE Malware Windows executable sent from remote host"; content: "MZ"; ... ; rev:3;)

Table 10: IPSec Policy Before Migration

GW1						
1.	TCP	VM1, VM2, VM3, **VM4**	ANY	CorpNet	ANY	ESP Tunnel GW3 {3DES}
GW3						
1.	TCP	CorpNet	ANY	VM1, VM2, VM3, **VM4**	ANY	ESP Tunnel **GW1** {3DES}

Table 11: IPSec Policy After Migration

GW1						
1.	TCP	VM1, VM2, VM3	ANY	CorpNet	ANY	ESP Tunnel GW3 {3DES}
GW2						
1.	**TCP**	**VM4**	**ANY**	**CorpNet**	**ANY**	**ESP Tunnel GW3 {3DES}**
GW3						
1.	TCP	CorpNet	ANY	VM1, VM2, VM3	ANY	ESP Tunnel GW1 {3DES}
2.	**TCP**	**CorpNet**	**ANY**	**VM4**	**ANY**	**ESP Tunnel GW2 {3DES}**

The verification results for the IDS scenarios are summarized in Table 12. Values in bold show the constraints, which satisfactions are not as expected. When the solver returns satisfiable for a constraint expected to be unsatisfiable, a solution is provided that pinpoints one of the rules that makes security requirements fails. This indicates a possible error that should be investigated in order to correct the configuration error. For IDS scenario 1, $\mathcal{C}_1$ and $\mathcal{C}_3$ have the unexpected satisfiability results of UNSAT and SAT, respectively. The satisfiability of the former can be explained by the fact that every rule in the configuration before is necessarily in the configuration after. The satisfiability of the latter is due to the fact that some rules concerning the migrating VM v have not been removed from the source, which corroborate the former result. The fact that $\mathcal{C}_3$ is SAT, the SAT-solver returned a solution for this constraint, which consists of a vector of values (a value per CSP variable) that represents one of the rules that have not been removed from the source.

With respect to IPsec, we consider the following scenarios:

Scenario 1 - Error 1. The administrator correctly updated GW1, but omitted any change to GW2.

Scenario 2 - Error 2. The administrator correctly updated the IPSec configuration in GW1 and

Table 12: Verification Results for the IDS Three Scenarios

	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$	$\mathcal{C}_4$	$\mathcal{C}_5$	$\mathcal{C}_6$	$\mathcal{C}_7$	$\mathcal{C}_8$	$\mathcal{C}_9$
Scen- 1	**UNSAT**	*UNSAT*	**SAT**	*UNSAT*	*UNSAT*	*UNSAT*	*UNSAT*	*UNSAT*	*UNSAT*
Scen- 2	*SAT*	*UNSAT*	*UNSAT*	*UNSAT*	*SAT*	*UNSAT*	**SAT**	*SAT*	**SAT**
Scen- 3	*SAT*	*UNSAT*	*UNSAT*	*UNSAT*	*SAT*	*UNSAT*	*UNSAT*	*UNSAT*	*UNSAT*

GW2, but the customer gateway GW3 has not been updated.

Scenario 3 - Correct Migration. The IPSec configurations on all the gateways are correctly configured, and the rules are provided in Table 11.

The verification results for the IPSec scenarios are summarized in Table 13 and Table 14. Values in bold show the constraints, which satisfactions are not as expected. For IPsec scenario 1, $\mathcal{C}_{14}$ and $\mathcal{C}_{22}$ have the unexpected satisfiability results of UNSAT and SAT, respectively. The satisfiability of the former can be explained by the fact that at the destination, every rule in the configuration after migration is necessarily in the configuration before. The satisfiability of the latter is due to the fact that some rules concerning the migrating VM v have not been migrated to the destination, which corroborate the former result. The fact that $\mathcal{C}_{22}$ is SAT, the solver returned a solution for this constraint, which consists of a vector of values (a value per CSP variable) that represents one of the rules that have not been migrated from the source.

Table 13: Verification Results for the IPSec Scenarios: the Source DC, Destination DC

	$\mathcal{C}_{10}$	$\mathcal{C}_{11}$	$\mathcal{C}_{12}$	$\mathcal{C}_{13}$	$\mathcal{C}_{14}$	$\mathcal{C}_{15}$	$\mathcal{C}_{16}$
Scen-1	*SAT*	*UNSAT*	*UNSAT*	*UNSAT*	**UNSAT**	*UNSAT*	*UNSAT*
Scen-2	*SAT*	*UNSAT*	*UNSAT*	*UNSAT*	*SAT*	*UNSAT*	*UNSAT*
Scen-3	*SAT*	*UNSAT*	*UNSAT*	*UNSAT*	*SAT*	*UNSAT*	*UNSAT*

In order to have an assessment of the performance overhead, we performed a set of experiments on an Intel Core i7 2.67 GHz processor with 12Gbytes of RAM. The verification performance for

Table 14: Verification Results for the IPSec Scenarios for the Customer Network and the Migrating VM

	$\mathcal{C}_{17}$	$\mathcal{C}_{18}$	$\mathcal{C}_{19}$	$\mathcal{C}_{20}$	$\mathcal{C}_{21}$	$\mathcal{C}_{22}$	$\mathcal{C}_{23}$	$\mathcal{C}_{24}$	$\mathcal{C}_{25}$
Scen- 1	*UNSAT*	*UNSAT*	*SAT*	*SAT*	*UNSAT*	**SAT**	*UNSAT*	*SAT*	*UNSAT*
Scen- 2	*UNSAT*	*UNSAT*	**UNSAT**	**UNSAT**	*UNSAT*	*UNSAT*	*UNSAT*	**UNSAT**	*UNSAT*
Scen- 3	*UNSAT*	*UNSAT*	*SAT*	*SAT*	*UNSAT*	*UNSAT*	*UNSAT*	*SAT*	*UNSAT*

the IDS depends on the total number of attack signatures and also total number of virtual machines. Whereas, the verification performance for the IPsec depends on total number of virtual machines which relatively determines the total number of rules to be configured in the IPsec gatweays. Tables 15 and 16 summarize the results in terms of CPU time for the IDS and IPsec, respectively.

Table 15: Performance Evaluation for IDS

Number of VMs	Number of Attack Signatures	Number of Rules	CPU Time (sec)
3	10	30	0.186
3	50	150	0.232
3	100	300	0.325
3	500	1500	0.389
3	1000	3000	0.482
3	2000	6000	0.637
3	5000	15000	2.167

Table 16: Performance Evaluation for IPsec

Number of VMs	Number of Rules	CPU Time (sec)
10	10	0.154
100	100	0.247
1000	1000	0.279
4000	4000	0.403
8000	8000	0.622
10000	10000	0.917
12000	12000	1.058
14000	14000	1.292
16000	16000	1.589

Chapter 7

Conclusion

In this chapter, we summarize the contributions of this thesis, and provide directions for future work.

7.1 Summary of Contributions

In this thesis, we addressed the issue of security policy preservation in elastic cloud computing environment with firewalls, intrusion detection and IPsec VPN as the principal security mechanisms. We proposed a novel verification and validation approach based on the notion of security policy preservation and the constraint satisfaction problems framework. First, the formal definition of firewall filtering preservation in source and destination data center as well as for the migrating VM was provided. Then, we elaborated a framework that describes these security preservation problems in terms of constraint satisfaction problems. Later on, we demonstrated the feasibility of our approach by verifying several cases using Sugar, a SAT-based constraint solver. In order

to automate the encoding, we proposed to model cloud network topology using cloud calculus, and to use an intermediate syntax in order to express serial and parallel composition of firewalls. The proposed automated approach is helpful for practitioners to tackle the uprising issues of network security in a highly dynamic cloud computing environment. In addition, we presented our approach for verification of intrusion monitoring preservation as well as IPsec protection preservation by solving a set of constraint satisfaction problems. Finally, we presented case studies for the abovementioned security mechanisms namely intrusion detection and prevention, and IPsec in order to show the applicability and usefulness of our approach. This approach is general in the way that it covers multiple security mechanisms, and provides a solid framework for the verification of security configurations in cloud infrastructure.

7.2 Future Work

For future work, we plan to advance in different directions. First of all, we consider to extend our verification framework to cover security challenges presented in cloud related technologies. There is a new trend (at the time of writing this thesis) called cloud networking pioneered by a European funded project called SAIL [93] initiated in 2010. Cloud networking extends network virtualisation beyond the data center by adding two features to cloud computing: the ability to connect the user to services in the cloud, and the ability to interconnect services that are geographically distributed across cloud infrastructures. In cloud computing there are two parties involved being service user and cloud infrastructure provider. The service user (tenant) manually checks whether his/her security requirements are followed by the cloud provider, and only if it is the case he/she moves her

resources to the cloud operator's infrastructure. Since in cloud networking virtual resources are eventually moved from one cloud operator's infrastructure to another, manually checking the conformance of security policy by the tenant is not any more feasible or even reasonable. Therefore, security checks have to be done automatically to make sure that the infrastructure provider follows the tenant's security requirement. Fusenig and Sharma [36] present a security architecture that enables a user of cloud networking to define security requirements, and enforce them in the cloud networking infrastructure. However, this work lacks an auditing technique so that a service user can verify if a security requirement is actually followed by the virtual infrastructure provider. In this regard, we plan to extend our verification framework to the context of cloud networking, and provide a verification framework that enables the tenants to make sure their security requirements are met by virtual infrastructure provider.

Second, we consider to develop a more sophisticated framework for security policy orchestration. While studying the elasticity feature offered in the cloud, we believe that there are still several issues that have to be addressed in the case of virtual machines migration. For example, when we have multiple VMs migrating simultaneously, the posibility of having a security misconfiguration is higher than the case of a single VM migration. Therefore, we can elaborate on this scenario, and evaluate its impact on our approach. In this case, cloud infrastructure providers and also cloud tenants would like to have independent migration with minimum interaction since those migrating VMs may not necessarily belong to a single tenant.

Third, we plan to elaborate a framework that not only pinpoints security policy misconfigurations in the cloud data centers, but also proposes comprehensive solutions to cloud administrators in order to solve inconsistencies.

Bibliography

[1] Muhammad Abedin, Syeda Nessa, Latifur Khan, Ehab AlShaer, and Mamoun Awad. Analysis of Firewall Policy Rules Using Traffic Mining Techniques. *International Journal of Internet Protocol and Technologies*, 5(1/2):3–22, April 2010.

[2] H.B. Acharya and M.G. Gouda. Firewall Verification and Redundancy Checking are Equivalent. In *INFOCOM, 2011 Proceedings IEEE*, pages 2123 –2128, April 2011.

[3] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan. Conflict Classification and Analysis of Distributed Firewall Policies. *Selected Areas in Communications, IEEE Journal on*, 23(10):2069 – 2084, Oct. 2005.

[4] E. Al-Shaer, W. Marrero, A. El-Atawy, and K. ElBadawi. Network Configuration in a Box: Towards End-to-End Verification of Network Reachability and Security. In *Network Protocols, 2009. ICNP 2009. 17th IEEE International Conference on*, pages 123 –132, Oct. 2009.

[5] J. G. Alfaro, N. Boulahia-Cuppens, and F. Cuppens. Complete Analysis of Configuration Rules to Guarantee Reliable Network Security Policies. *International Journal of Information Securiry*, 7(2):103–122, March 2008.

[6] Turki Alharkan and Patrick Martin. IDSaaS: Intrusion Detection System as a Service in Public Clouds. In *CCGRID*, pages 686–687, 2012.

[7] Cloud Security Alliance. SecaaS Implementation Guidance, Category 6: Intrusion Management. https://downloads.cloudsecurityalliance.org/initiatives/secaas/SecaaS_Cat_6_Intrusion_Management_Implementation_Guidance.pdf, September 2012. Last visited: December 2012.

[8] Amazon.com. Amazon web services: Overview of security processes. http://awsmedia.s3.amazonaws.com/pdf/AWS_Security_Whitepaper.pdf, May 2011.

[9] Fatemeh Azmandian, Micha Moffie, Malak Alshawabkeh, Jennifer Dy, Javed Aslam, and David Kaeli. Virtual Machine Monitor-based Lightweight Intrusion Detection. *SIGOPS Oper. Syst. Rev.*, 45(2):38–53, July 2011.

[10] N. Ben Youssef and A. Bouhoula. Automatic Conformance Verification of Distributed Firewalls to Security Requirements. In *Social Computing (SocialCom), 2010 IEEE Second International Conference on*, pages 834 –841, Aug. 2010.

[11] N. Ben Youssef, A. Bouhoula, and F. Jacquemard. Automatic Verification of Conformance of Firewall Configurations to Security Policies. In *Computers and Communications, 2009. ISCC 2009. IEEE Symposium on*, pages 526 –531, July 2009.

[12] P. Bera, S. K. Ghosh, and Pallab Dasgupta. Formal Verification of Security Policy Implementations in Enterprise Networks. In *Proceedings of the 5th International Conference on Information Systems Security*, ICISS '09, pages 117–131, Berlin, Heidelberg, 2009. Springer-Verlag.

[13] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Yunshan Zhu. Bounded Model Checking. volume 58 of *Advances in Computers*, pages 117 – 148. Elsevier, 2003.

[14] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic Model Checking without BDDs. In *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, TACAS '99, pages 193–207, London, UK, UK, 1999. Springer-Verlag.

[15] Sören Bleikertz, Thomas Groß, and Sebastian Mödersheim. Automated Verification of Virtualized Infrastructures. In *the Proceedings of the 3rd ACM workshop on Cloud Computing Security (CCSW)*, pages 47–58, 2011.

[16] Achim D. Brucker, Lukas Brügger, Paul Kearney, and Burkhart Wolff. Verified Firewall Policy Transformations for Test Case Generation. In *Proceedings of the 2010 Third International Conference on Software Testing, Verification and Validation*, ICST '10, pages 345–354, Washington, DC, USA, 2010. IEEE Computer Society.

[17] R.E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *Computers, IEEE Transactions on*, C-35(8):677–691, 1986.

[18] Michele Bugliesi, Silvia Crafa, Massimo Merro, and Vladimiro Sassone. Communication and Mobility Control in Boxed Ambients. *Inf. Comput.*, 202(1):39–86, 2005.

[19] J. R. Burch, E.M. Clarke, K. L. McMillan, D.L. Dill, and L. J. Hwang. Symbolic Model Checking: 10 to the power of 20 States and Beyond. In *Logic in Computer Science, 1990. LICS '90, Proceedings., Fifth Annual IEEE Symposium on e*, pages 428–439, 1990.

[20] Levente Buttyan, Gabor Pek, and Ta Vinh Thong. Consistency Verification of Stateful Firewalls is Not Harder Than the Stateless Case. `http://www.techrepublic.com/whitepapers/`, August 2009.

[21] Rajkumar Buyya, James Broberg, and Andrzej Goscinski, editors. *Cloud Computing Principles and Paradigms*. John Wiley and Sons, Inc., 2011.

[22] Venanzio Capretta, Bernard Stepien, Amy Felty, and Stan Matwin. Formal Correctness of Conflict Detection for Firewalls. In *Proceedings of the 2007 ACM workshop on Formal methods in security engineering*, FMSE '07, pages 22–30, New York, NY, USA, 2007. ACM.

[23] Luca Cardelli and Andrew D. Gordon. Mobile Ambients. *Theoretical Computer Science*, 240(1):177 – 213, 2000.

[24] Rupali Chaure. An Implementation of Anomaly Detection Mechanism for Centralized and Distributed Firewalls. *International Journal of Computer Applications*, 7(4):5 – 8, September 2010.

[25] Zhe Chen, Shize Guo, and Rong Duan. Research on the Anomaly Discovering Algorithm of the Packet Filtering Rule Sets. In *Pervasive Computing Signal Processing and Applications (PCSPA), 2010 First International Conference on*, pages 362 –366, Sept. 2010.

[26] Cisco. CISCO ASA 1000V Cloud Firewall Data Sheet. `http://www.cisco.com/en/US/prod/collateral/vpndevc/ps6032/ps6094/ps12233/data_sheet_c78-687960.pdf`. [Online; accessed 19-March-2013].

[27] Edmund Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded Model Checking Using Satisfiability Solving. *Form. Methods Syst. Des.*, 19(1):7–34, July 2001.

[28] Mathieu Couture, Béchir Ktari, Mohamed Mejri, and Frédéric Massicotte. A Declarative Approach to Stateful Intrusion Detection and Network Monitoring. In *PST*, pages 175–179, 2004.

[29] Frédéric Cuppens, Nora Cuppens-Boulahia, Joaquin Garcia-Alfaro, Tarik Moataz, and Xavier Rimasson. Handling Stateful Firewall Anomalies. In Dimitris Gritzalis, Steven Furnell, and Marianthi Theoharidou, editors, *Information Security and Privacy Research*, volume 376 of *IFIP Advances in Information and Communication Technology*, pages 174–186. Springer Berlin Heidelberg, 2012.

[30] Sudhir N. Dhage and B. B. Meshram. Intrusion Detection System in Cloud Computing Environment. *International Journal of Cloud Computing*, 1(2):261–282, 2012.

[31] V. Lifschitz F. van Harmelen and B. Porter. *Handbook of Knowledge Representation*. Elsevier B.V., 2008.

[32] S. Ferraresi, E. Francocci, A. Quaglini, and A. Baiocchi. Algorithm to Automatically Solve Security Policy Conflicts Among IP Devices Configurations. In *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*, pages 923 –926, April 2008.

[33] S. Ferraresi, S. Pesic, L. Trazza, and A. Baiocchi. Automatic Conflict Analysis and Resolution of Traffic Filtering Policy for Firewall and Security Gateway. In *Communications, 2007. ICC '07. IEEE International Conference on*, pages 1304 –1310, June 2007.

[34] Zhi Fu, S. Felix Wu, He Huang, Kung Loh, Fengmin Gong, Ilia Baldine, and Chong Xu. IPSec/VPN Security Policy: Correctness, Conflict Detection and Resolution. In *In Proc. 2nd IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY), volume 1995 of LNCS*, pages 39–56. Springer, 2001.

[35] Borko Furht and Armando Escalante. *Handbook of Cloud Computing*. Springer, 2010.

[36] V. Fusenig and A. Sharma. Security Architecture for Cloud Networking. In *Computing, Networking and Communications (ICNC), 2012 International Conference on*, pages 45–49, 2012.

[37] Joaquin Garcia-Alfaro, Frédéric Cuppens, Nora Cuppens-Boulahia, and Stere Preda. MIRAGE: A Management Tool for the Analysis and Deployment of Network Security Policies. In *Proceedings of the 5th international Workshop on data privacy management, and 3rd international conference on Autonomous spontaneous security*, DPM'10/SETOP'10, pages 203–215. Springer-Verlag, 2011.

[38] Tal Garfinkel and Mendel Rosenblum. A Virtual Machine Introspection Based Architecture for Intrusion Detection. In *In Proc. Network and Distributed Systems Security Symposium*, pages 191–206, 2003.

[39] Tal Garfinkel and Mendel Rosenblum. When Virtual is Harder than Real: Security Challenges in Virtual Machine Based Computing Environments. In *Proceedings of the 10th conference on Hot Topics in Operating Systems - Volume 10*, HOTOS'05, pages 20–20, Berkeley, CA, USA, 2005. USENIX Association.

[40] Amjad Gawanmeh and Sofiène Tahar. Modeling and Verification of Firewall Configurations Using Domain Restriction Method. In *6th International Conference on Internet Technology and Secured Transactions*, December 2011.

[41] K. Golnabi, R.K. Min, L. Khan, and E. Al-Shaer. Analysis of Firewall Policy Rules Using Data Mining Techniques. In *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, pages 305 – 315, April 2006.

[42] Google. Google App Engine. `https://appengine.google.com/`. [Online; accessed 19-March-2013].

[43] M.G. Gouda and A.X. Liu. A Model of Stateful Firewalls and its Properties. In *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*, pages 128 – 137, June-1 July 2005.

[44] M.G. Gouda, A.X. Liu, and M. Jafry. Verification of Distributed Firewalls. In *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*, pages 1 –5, Dec. 2008.

[45] Y. Gu, C. Li, K. Li, Z. Zhuo, and D. Zhang. State Migration, March 2012. Work in Progress.

[46] Y. Gu, C. Li, K. Li, Z. Zhuo, and D. Zhang. State Migration. `http://tools.ietf.org/id/draft-gu-opsawg-policies-migration-02.txt`, March 2012. Internet Draft, Last Accessed: October 2012 ,Expires: September 6, 2012.

[47] Y. Gu, M. Shore, and S. Sivakumar. A Framework and Problem Statement for Flow-associated Middlebox State Migration. `http://www.ietf.org/id/draft-gu-statemigration-framework-03.txt`, October 2012. Internet Draft, draft-gu-statemigration-framework-03, Last Accessed: December 2012, Expires: April 24, 2013.

[48] Mohammad Hajjat, Xin Sun, Yu-wei Eric Sung, David Maltz, and Sanjay Rao. Cloudward Bound : Planning for Beneficial Migration of Enterprise Applications to the Cloud. In *the Proceedings of the ACM SIGCOMM 2010*, pages 243–254, New York, NY, USA, 2010. ACM.

[49] Mohammad Hajjat, Xin Sun, Yu wei Eric Sung, David Maltz, Sanjay Rao, Kunwadee Sripanidkulchai, and Mohit Tawarmalani. Cloudward Bound: Planning for Beneficial Migration of Enterprise Applications to the Cloud. In *In the Proceedings of the ACM SIGCOMM 2010 conference*, pages 243–254, New York, NY, USA, 2010. ACM.

[50] H. Hamed and E. Al-Shaer. Taxonomy of Conflicts in Network Security Policies. *Communications Magazine, IEEE*, 44(3):134 – 141, March 2006.

[51] H. Hamed, E. Al-Shaer, and W. Marrero. Modeling and Verification of IPSec and VPN Security Policies. In *Network Protocols, 2005. ICNP 2005. 13th IEEE International Conference on*, page 10 pp., Nov. 2005.

[52] A. A. Hassan. Algorithms for Verifying Firewall and Router Access Lists. In *Circuits and Systems, 2003 IEEE 46th Midwest Symposium on*, volume 1, pages 512 – 515 Vol. 1, Dec. 2003.

[53] A. A. Hassan and L. Hudec. Management and Verification of Firewall an Router Access Lists. *COMPUTING AND INFORMATICS*, 23(1):77–100, February 2004.

[54] H. Hu, G. Ahn, and K. Kulkarni. FAME: A Firewall Anomaly Management Environment. In *SafeConfig10 Proceedings of the 3rd ACM workshop on Assurable and usable security configuration*, 2010.

[55] Hongxin Hu, Gail-Joon Ahn, and Ketan Kulkarni. Detecting and Resolving Firewall Policy Anomalies. *IEEE Transactions on Dependable and Secure Computing, TDSC*, 9(3):318–331, 2012.

[56] G. V. Hulme. Cloudpassage Aims to Ease Cloud Server Security Management. `http://www.csoonline.com/article/658121/`, January 2011.

[57] A.S. Ibrahim, J. Hamlyn-Harris, John Grundy, and M. Almorsy. CloudSec: A Security Monitoring Appliance for Virtual Machines in the IaaS Cloud Model. In *Network and System Security (NSS), 2011 5th International Conference on*, pages 113–120, 2011.

[58] A. Jasti, P. Shah, R. Nagaraj, and R. Pendse. Security in Multi-Tenancy Cloud. In *Security Technology (ICCST), 2010 IEEE International Carnahan Conference on*, pages 35–41, 2010.

[59] A. Jeffrey and T. Samak. Model Checking Firewall Policy Configurations. In *Policies for Distributed Systems and Networks, 2009. POLICY 2009. IEEE International Symposium on*, pages 60 – 67, July 2009.

[60] Hicham El Khoury, Romain Laborde, François Barrère, Maroun Chamoun, and Abdelmalek Benzekr. A Formal Data Flow-Oriented Model For Distributed Network Security Conflicts Detection. In *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*, pages 20–27, March 2012.

[61] I. Kotenko and O. Polubelova. Verification of Security Policy Filtering Rules by Model Checking. In *the Proceedings of the IEEE 6th International Conference on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS)*, volume 2, pages 706 –710, Sept. 2011.

[62] KVM. Kvm. `http://www.linux-kvm.org`.

[63] Huan Liu. A New Form of DOS Attack in a Cloud and its Avoidance Mechanism. In *Proceedings of the 2010 ACM workshop on Cloud computing security workshop*, CCSW '10, pages 65–76, New York, NY, USA, 2010. ACM.

[64] L. Lu, R. Safavi-Naini, J. Horton, and W. Susilo. Comparing and Debugging Firewall Rule Tables. *IET Information Security*, 1(4):143–151, 2007.

[65] Dave Malcolm. The Five Pillars of Cloud Computing. `http://soa.sys-con.com/node/904780`, 2009. SOA & WOA magazine, Cloud Expo article, Last Visited April 2012.

[66] Tim Mather, Subra Kumaraswamy, and Shahed Latif. *Cloud Security and Privacy An Enterprise Perspective on Risks and Compliance*. O'RIELLY, 2009.

[67] Jeanna Matthews, Tal Garfinkel, Christofer Hoff, and Jeff Wheeler. Virtual Machine Contracts for Datacenter and Cloud Computing Environments. In *the Proceedings of the first Workshop on Automated Control for Datacenters and Clouds (ACDC)*, pages 25–30, New York, NY, USA, 2009. ACM.

[68] Kenneth Lauchlin McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1992. UMI Order No. GAX92-24209.

[69] M. Mejri, K. Adi, and H. Fujita. Formal Specification and Analysis of Firewalls. In *Proceedings of the 2009 conference on New Trends in Software Methodologies, Tools and Techniques*, Amsterdam, The Netherlands, The Netherlands, 2009.

[70] Microsoft. Microsoft Office 365. office.microsoft.com/en-ca/. [Online; accessed 19-March-2013].

[71] 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.

[72] Antonio Muñoz, Javier Gonzalez, and Antonio Maña. A Performance-Oriented Monitoring System for Security Properties in Cloud Computing Applications. *The Computer Journal*, 55(8):979–994, 2012.

[73] Maria Napierala, Luyuan Fang, and Dennis Cai. IP-VPN Data Center Problem Statement and Requirements, June 2012. Work in Progress.

[74] Flemming Nielson, Hanne Nielson, René Hansen, and Jacob Jensen. Validating Firewalls in Mobile Ambients. In Jos Baeten and Sjouke Mauw, editors, *CONCUŔ99 Concurrency Theory*, volume 1664 of *Lecture Notes in Computer Science*, pages 78–78. Springer Berlin / Heidelberg, 1999.

[75] Salman Niksefat and Masoud Sabaei. Efficient Algorithms for Dynamic Detection and Resolution of IPSec/VPN Security Policy Conflicts. In *Proceedings of the 2010 24th IEEE International Conference on Advanced Information Networking and Applications*, AINA '10, pages 737–744, Washington, DC, USA, 2010. IEEE Computer Society.

[76] NIST. The NIST Definition of Cloud Computing. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf. [Online; accessed 19-March-2013].

[77] NIST. NIST SP 800-77 Guide to IPsec VPNs. http://csrc.nist.gov/publications/nistpubs/800-77/sp800-77.pdf, December 2005. Last Accessed: August 2013.

[78] NIST. NIST SP 800-94 Guide to Intrusion Detection and Prevention Systems (IDPS). http://csrc.nist.gov/publications/nistpubs/800-94/SP800-94.pdf, February 2007. Last Accessed: August 2013.

[79] V. Nunes Leal Franqueira, P. A. T. van Eck, R. J. Wieringa, and R. H. C. Lopes. A Mobile Ambients-based Approach for Network Attack Modelling and Simulation. In *the Proceedings of the 4th Int. Workshop on Dependability Aspects on Data Warehousing and Mining applications, (DAWAM)*, pages 546–553, Los Alamitos, March 2009. IEEE Computer Society.

[80] Jon Oberheide, Evan Cooke, and Farnam Jahanian. Exploiting Live Virtual Machine Migration. In *BlackHat DC Briefings*, Washington DC, February 2008.

[81] Liang. Ou, Huamin. Jin, and M. Chen. Content Delivery Network Based Virtual Machine Migration Scheme. http://tools.ietf.org/html/draft-ou-cdn-vm-migration-scheme-00, October 2012. Internet Draft, draft-ou-cdn-vm-migration-scheme-00, Last Accessed: December 2012, Expires: April 16, 2013.

[82] Liviu Pene and Kamel Adi. A Calculus for Distributed Firewall Specification and Verification. In *the Proceedings of the the fifth SoMeT'06 on New Trends in Software Methodologies, Tools and Techniques*, pages 301–315, Amsterdam, The Netherlands, The Netherlands, 2006. IOS Press.

[83] S. Pozo, R. Ceballos, and R. M. Gasca. CSP-Based Firewall Rule Set Diagnosis using Security Policies. In *Availability, Reliability and Security, 2007. ARES 2007. The Second International Conference on*, pages 723 –729, April 2007.

[84] Promela. Promela Language. `http://spinroot.com/spin/Man/promela.html`. [Online; accessed 12-August-2013].

[85] R. Ptak. CloudPassage: First Security/Compliance Assurance Solutions for the Elastic Cloud. `http://ptaknoel.com/`, Feb. 2011.

[86] Rackspace. The Rackspace Cloud. `http://www.rackspace.com/cloud/`. [Online; accessed 19-March-2013].

[87] Srinivasa Rao, Boddi Reddy Rama, and K.Naga Mani. Firewall Policy Management Through Sliding Window Filtering Method Using Data Mining Techniques. *International Journal of Computer Science and Engineering Survey*, 2(2):39–55, May 2011.

[88] Mark Reitblatt, Nate Foster, Jennifer Rexford, Cole Schlesinger, and David Walker. Abstractions for Network Update. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '12, pages 323–334, New York, NY, USA, 2012. ACM.

[89] M. Rezvani and R. Aryan. Analyzing and Resolving Anomalies in Firewall Security Policies based on Propositional Logic. In *Multitopic Conference, 2009. INMIC 2009. IEEE 13th International*, pages 1 – 7, Dec. 2009.

[90] M. Rezvani and R. Aryan. Specification, Analysis and Resolution of Anomalies in Firewall Security Policies. *World Applied Sciences Journal 7 (Special Issue of Computer and IT)*, pages 188 – 198, 2009.

[91] S. Roschke, Feng Cheng, and C. Meinel. Intrusion Detection in the Cloud. In *Dependable, Autonomic and Secure Computing, 2009. DASC '09. Eighth IEEE International Conference on*, pages 729 –734, Dec. 2009.

[92] Mohsen Rouached, Hassen Sallay, Ouissem Ben Fredj, Adel Ammar, Khaled Al-Shalfan, and Majdi Ben. Formal Analysis of Intrusion Detection Systems for High Speed Networks. In *Proceedings of the 9th WSEAS international conference on Advances in e-activities, information security and privacy*, ISPACT'10, pages 109–114, Stevens Point, Wisconsin, USA, 2010. World Scientific and Engineering Academy and Society (WSEAS).

[93] SAIL. Scalable and Adaptive Internet Solutions (SAIL). `http://www.sail-project.eu/`. [Online; accessed 19-March-2013].

[94] Sebastian Schmerl, Michael Vogel, and Hartmut König. Using Model Checking to Identify Errors in Intrusion Detection Signatures. *Int. J. Softw. Tools Technol. Transf.*, 13(1):89–106, January 2011.

[95] S. Shin and G. Gu. CloudWatcher: Network Security Monitoring using OpenFlow in Dynamic Cloud Networks (or: How to Provide Security Monitoring as a Service in Clouds?). In *Network Protocols (ICNP), 2012 20th IEEE International Conference on*, pages 1–6. IEEE, 2012.

[96] Snort. What is Snort. `http://www.snort.org/`. [Online; accessed 12-August-2013].

[97] Tao Song. *Formal Reasoning about Intrusion Detection Systems*. PhD thesis, University of California, Davis, 2007.

[98] SPIN. Verifying Multi-threaded Software with Spin. `http://spinroot.com/spin/whatispin.html`. [Online; accessed 12-August-2013].

[99] Natalia Stakhanova, Yao Li, and Ali A. Ghorbani. Classification and Discovery of Rule Misconfigurations in Intrusion Detection and Response Devices. In *Proceedings of the 2009 World Congress on Privacy, Security, Trust and the Management of e-Business*, CONGRESS '09, pages 29–37, Washington, DC, USA, 2009. IEEE Computer Society.

[100] S. Subashini and V. Kavitha. A Survey on Security Issues in Service Delivery Models of Cloud Computing. *Journal of Network and Computer Applications*, 34(1):1 – 11, 2011.

[101] Sugar. Syntax of Sugar CSP Description. `http://bach.istc.kobe-u.ac.jp/sugar/package/current/docs/syntax.html`, 2010. Last modified: Tue Jun 29 13:09:26 2010 JST.

[102] N. Tamura and M. Banbara. Sugar: A CSP to SAT Translator Based on Order Encoding. In *the Proceedings of the Second International CSP Solver Competition*, pages 65–69, 2008.

[103] Zahra Tavakoli, Sebastian Meier, and Alexander Vensmer. A Framework for Security Context Migration in a Firewall Secured Virtual Machine Environment. In *EUNICE*, volume 7479 of *Lecture Notes in Computer Science*, pages 41–51. Springer, 2012.

[104] Ines Ben Tekaya, Mohamed Graiet, and Bechir Ayeb. Intrusion Detection with Symbolic Model Verifier. In *ICSEA 2011, The Sixth International Conference on Software Engineering Advances*, pages 183–189, 2011.

[105] Fotis Tsifountidis. Virtualization Security: Virtual Machine Monitoring and Introspection. Master's thesis, Royal Holloway, University of London, UK, 2010.

[106] Tomás E. Uribe and Steven Cheung. Automatic Analysis of Firewall and Network Intrusion Detection System Configurations. In *Proceedings of the 2004 ACM workshop on Formal methods in security engineering*, FMSE '04, pages 66–74, New York, NY, USA, 2004. ACM.

[107] Luis M. Vaquero, Luis Rodero-Merino, and Daniel Morán. Locking the Sky: A Survey on IaaS Cloud Security. *Computing*, 91(1):93–118, January 2011.

[108] R. Villemaire and S. Halle. Strong Temporal, Weak Spatial Logic for Rule Based Filters. In *Temporal Representation and Reasoning, 2009. TIME 2009. 16th International Symposium on*, pages 115 – 121, July 2009.

[109] VMware and Savvis. Securing the Cloud: A Review of Cloud Computing, Security Implications and Best Practices. Technical report, VMware and Savvis, 2009. white paper.

[110] Y. Wang, Y. Gu, and D. Zhang. vFW State Migration Problem Statement. https://datatracker.ietf.org/doc/draft-wang-state-migration-vfirewall/, December 2012. Internet Draft, draft-wang-state-migration-vfirewall-00, Last Accessed: February 2013, Expires: June 22, 2013.

[111] WMware. Vmware. http://www.VMware.org.

[112] T. Wood, K. K. Ramakrishnan, J. Van Der Merwe, and P. Shenoy. Cloudnet: A Platform for Optimized WAN Migration of Virtual Machines. Technical report, University of Massachusetts, 2010.

[113] Xen. Xen. http://www.xen.org.

[114] Chen Xianqin, Wan Han, Wang Sumei, and Long Xiang. Seamless Virtual Machine Live Migration on Network Security Enhanced Hypervisor. In *Broadband Network Multimedia Technology, 2009. IC-BNMT '09. 2nd IEEE International Conference on*, pages 847 –853, Oct. 2009.

[115] Yi Yin, Jiangdong Xu, and Naohisa Takahashi. Verifying Consistency Between Security Policy and Firewall Policy by Using a Constraint Satisfaction Problem Server. In Ying Zhang, editor, *Future Computing, Communication, Control and Management*, volume 144 of *Lecture Notes in Electrical Engineering*, pages 135–145. Springer Berlin Heidelberg, 2012.

[116] Shuyuan Zhang, A. Mahmoud, S. Malik, and S. Narain. Verification and Synthesis of Firewalls Using SAT and QBF. In *Network Protocols (ICNP), 2012 20th IEEE International Conference on*, pages 1–6, 30 2012-Nov. 2.

[117] Yinqian Zhang, Ari Juels, Alina Oprea, and Michael K. Reiter. HomeAlone: Co-residency Detection in the Cloud via Side-Channel Analysis. In *IEEE Symposium on Security and Privacy*, pages 313–328, 2011.

[118] 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*, CCS '12, pages 305–316, New York, NY, USA, 2012. ACM.