An Interaction-based Software-Defined Security Model and Platform to secure cloud resources

A dissertation submitted to

Faculty of Engineering and Information Technology

University of Technology Sydney

In fulfilment of the requirements for the award of

Doctor of Philosophy

By

Sara Farahmandian

Supervised by

Professor Doan B. Hoang

2021

Dedicated To

My Divine Source

My parents, and my siblings

My primary supervisor

Thank for your great support and love

Certificate of Original Authorship

I, Sara Farahmandian declare that this thesis is submitted in fulfilment of the requirements for the award of Doctor of Philosophy, in the Faculty of Engineering and Technology at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise reference or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This document has not been submitted for qualifications at any other academic institution.

This research is supported by the Australian Government Research Training Program.

Signature of Student: Sara Farahmandian

Date: <04/01/2021>

Acknowledgement

During my doctoral candidature, I have received a myriad of lessons, support, and encouragement. I wish to express my sincere gratitude to my supervisor, Professor Doan B. Hoang, for his support and for sharing this journey alongside me. I cannot express in words how grateful I am and how much his supervision and mentorship meant to me. He has taught me significantly valuable lessons that enlightened my academic and personal life. He has been outstanding in providing insightful feedback and creating the balance between working and living. Following his lessons, I gradually become an independent researcher. I would like to thanks my co-supervisor, Dr. Priyadarsi Nanda, for his support during my candidature.

I am thankful for all SEDE staff who supports me in every steps of my journey. I would like to express my special thanks to all my teammates and friends in UTS Women in Engineering and IT (WiEIT) for their remarkable help and support. I will never forget the encouragement and assistance from all my colleagues and friends, including Tham Nguyen, Chau Nguyen, Ngoc Le, Tuan Vinh Ha, Hasti Hayati, Behnam Maleki, Ashish Nanda, Madhumita Abhijeet Takalkar, Deepak Puthal, Marie Joshua Tapas, and Azita Zoughi.

Finally, I would like to show my appreciation to my Family. To my mother, Parvin, the most incredible light in my life who taught me to be kind and strong and supported me in every hard steps of my life, to my father who always encourage me to overcome difficulties in my life, to my sister, Sepideh, who is my role model and symbol of pure light and love in my life, to my wonderful brother, Ehsan, whom talking to always keep me motivated, to my sister-in-law, Azadeh, who always kindly supported and encouraged me during this journey, to my little niece, Elina, whom her existence brings me pure joy and happiness, to my brother, Amin, whom always encouraged me in every step of life. Their support and love encouraged me in every step of my journey. This dissertation is dedicated to them. Without all of you, this research journey would not have been possible.

Author's Publications

Journal Article:

1. Hoang, D.B. and **S. Farahmandian**, *"Security of Software-Defined Infrastructures with SDN, NFV, and Cloud Computing Technologies",* in Guide to Security in SDN and NFV. 2017. Springer. p. 3-32.

2. **Farahmandian, S**. and D.B. Hoang, *"Policy-based Interaction Model for Cloud Security Breaches Detection and Prediction"*. Journal of Telecommunications and the Digital Economy, 2020.

3. **Farahmandian, S.** and D.B. Hoang, *"Software-defined Security Service Platform for Securing Cloud Infrastructure"*. IEEE Transactions on Cloud Computing (TCC) Journal, 2021 (under-review).

Conference Papers:

4. **Farahmandian, S**. and D.B. Hoang. *Security for software-defined (cloud, SDN and NFV) infrastructures–issues and challenges.* in Eight international conference on network and communications security. 2016.

5. **Farahmandian, S**. and D.B. Hoang. *SDS 2: A novel software-defined security service for protecting cloud computing infrastructure*. in 2017 IEEE 16th International Symposium on Network Computing and Applications (NCA). 2017. IEEE.

6. **Farahmandian, S**. and D.B. Hoang. *A Policy-based Interaction Protocol between Software Defined Security Controller and Virtual Security Functions.* in 2020 4th Cyber Security in Networking Conference (CSNet). 2020. IEEE.

Table of Contents

Figures

Tables

Algorithms

Abbreviations and Acronyms

NIST	National Institute of Standards and Technology
IEEE	Institute of Electrical and Electronics Engineers
ONF	Open Networking Foundation
ETSI	European Telecommunications Standards Institute
NFV	Network Function Virtualization
SDN	Software-Defined Networking
SDS_2	Software Defined Security Service
OT	Operational technology
SDSec	Software-Defined Security
VNF	Virtual Network Function
EM	Element Management
VSF	Virtual Security Function
VN	Virtual Network
VM	Virtual Machine
SDI	Software-Defined Infrastructure
SDC	Software-Defined compute
SDS	Software-Defined Storage
SC	Security Controller

OS	Operating System
CSA	Cloud Security Alliance
VLAN	Virtual Local Area Network
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
SaaS	Software-as-a-Service
PaaS	Platform-as-a-Service
IaaS	Infrastructure-as-a-Service
CP	Cloud Provider
CS	Cloud Service
CSP	Cloud Service Provider
ISP	Infrastructure Service Provider
SLA	Service Level Agreement
API	Application Programming Interface
SSL	Secure sockets layer
TLS	Transport Layer Security
APT	Advanced Persistent Threats
DoS	Denial-of-service
DDoS	Distributed Denial-of-service
SBI	Southbound Interface
NBI	Northbound Interface
OvS	OpenvSwitch
NFVI	NFV Infrastructure
EPC	Evolved Packet Core
MANO	Management and Orchestration
VNFM	Virtual Network Function Manager
VSFM	Virtual Security Function Manager
CC	Cloud Controller
SC	Security Controller
SP	Security Policy
IP	Internet Protocol
MAC	Media Access Control
SPM	Security Policy Manager
ESPM	Entity security policy-driven manager
PIM	Policy-based interaction manager
DPI	Deep Packet Inspections
SNM	Security Network Manager
ID	Identification

Abstract

Cloud computing has transformed a large portion of the IT industry through its ability to provision infrastructure resources – computing, networking, storage, and software – as services. Transferring to such an infrastructure relies on virtualization and its dynamic construction ability to spread over a geographical area. The challenge is in finding effective mechanisms for isolating security issues in cloud infrastructure. Isolation implies creating security boundaries for protecting cloud assets at different levels of a cloud security architecture. Building security boundaries is critical not only for recognizing security violations but also for creating security solutions. However, it is challenging as virtual boundaries are not as clear-cut as physical boundaries in traditional infrastructure. The difficulty rises as virtual boundaries among components are not well defined and often undefined, and hence they are not visible/controllable by the providers.

Additionally, defining object boundaries is extremely difficult because virtual objects are dynamic in both characteristics and functionality. Many efforts have been made to address security isolation challenges, but no attempt has been made to consider an overall solution to a dynamic, intelligent, programable, and on-demand security isolation system. Moreover, there is no platform/framework to deliver programmable and on-demand construction of security boundaries to protect cloud resources.

We develop a new method to protect cloud infrastructure with new intelligent isolation mechanisms to detect and predict security breaks. This research applies promising new technologies, including software-defined networking and network function virtualization, in providing on-demand security services over large-scale cloud infrastructure and overcoming challenges in constructing dynamic security boundaries. To protect cloud resources, we propose a Policy-based Interaction Model

and develop the Software-Defined Security Service. We develop a novel intelligent security isolation interaction algorithm to model security boundaries. To do so, we proposed a Policy-driven Interaction Model to construct dynamic security boundaries intelligently. A Software-Defined Security Service (SDS_2) model was developed with three novel components, including security controller, Sec-Manage protocol, and the virtual security function. The SDS_2 carries the concepts of a logically centralized security controller to provision on-demand security services.

The research novelty lies in its innovative and intelligent security isolation interaction model, novel approach in detecting and predicting security violations, and constructing dynamic, programmable, and on-demand VSFs. It enables i) overall visibility on security boundaries within the cloud infrastructure, ii) the automation of provisioning security services on-demand, iii) a proactive security technique against security interaction violations, iv) separation of security services for both cloud providers and tenants.

Chapter 1

Introduction

1.1 Introduction

Over the last decade, cloud computing has established itself as a useful technology for sharing and provisioning resources among tenants in a pay-as-you-go service fashion. It has transformed a large portion of the IT industry through its ability to provision infrastructure resources – computing, networking, storage, and software – as services. The concept of everything-as-a-service was developed to utilize virtualization technology that allows underlying physical resources to be virtualized into virtual resources and services [1].

Cloud computing relies on its aggregation and centralization of virtual resources and flexible provision and orchestration to provide services to its customers. The cloud is a large scalable environment that consists of a vast number of physical and virtual resources operating and communicating over the cloud network. The cloud resources are shared in an extensive distributed infrastructure where they can be allocated in various locations worldwide [2].

Meanwhile, the cloud has developed as a complex and large-scale infrastructure. It turns out to be more vulnerable to traditional and new security threats related to its structure and components. NIST declares security, portability, and interoperability as the main obstacles to ultimately adopting the cloud environment

[3]. Some of the traditional security issues found in the cloud infrastructure are data access control (illegal access to confidential data), loss and data leakage, trust, isolation.

Moreover, the ever-increasing number and gravity of cyberattacks against cloud assets together with the introduction of new software-defined technologies such as Network Function Virtualization (NFV), Software-Defined Networking (SDN), and on-demand IoT devices/services have brought with them many severe cloud security issues.

Accordingly, the standing of cybersecurity in our current society or any organization is undeniable. With modern infrastructures that support ever-increasingly complex and pervasive applications, such as social networks, the Internet of everything, mobile applications, cloud services, new security models, and innovative security technologies must be invented to match emerging applications sophistication as well as the complexity of their attacks.

As stated in the official AustCyber's website, "*The Internet of Things, Cloud Computing and the convergence of IT and operational technology (OT), are some of the current important disruptive technological trends that will contribute to the future demand of cyber security solutions.*"[4]. Exclusively the global connectivity and drastic growth of cloud services are increasing cybersecurity risks.

The virtualized and resource sharing nature of cloud infrastructures and the deployment of new technologies such as Software-Defined Networking (SDN) and Network Function Virtualization (NFV) has created a surge in the number of potential targets and the complexity of security threats and their defence mechanisms. As a consequence, numerous major security issues have been acknowledged related to current cloud infrastructure:

The cumulative number of virtual resources/functions and their connectivity. As anticipated, with the growing requests for cloud services, demand for virtual resources has drastically increased within the infrastructure. However, the major challenge is how to manage the security complexity of the interactions of these resources via an effective approach within the cloud infrastructure, which increases the reliability of cloud resources.

The massive number of virtual resources and their segregation. Cloud resources are shared among various customers in different centres all over the world. The challenge here is how to construct security boundaries to achieve sound isolation and how to isolate the resources on-demand and dynamically within an environment where its resource states/characteristics dynamically change. The main challenge is in finding effective isolation mechanisms in cloud infrastructure. Isolation implies the creation of security boundaries for protecting cloud assets at different levels of cloud security architecture.

Furthermore, cloud providers and tenant administrators' primary concern is to establish security in the virtual environment where each virtual function's workload, resources, and internal and external communication can be securely isolated when necessary on-demand. Nasseri et al. [5] mentioned a lack of proper consideration of isolation related to security in the research and academic community regardless of the fact that isolation is crucial and has a critical impact on confidentiality, integrity, and infrastructure availability. Security issues in a virtual cloud environment are more complex and challenging than those in traditional infrastructures since resources are both virtualized and shared among numerous users.

As a result, virtual boundaries among components/participants are not well defined and often undefined, and hence they are not visible/controllable by the providers. Multi-tenancy is a specific cloud characteristic that allows the sharing of applications, services, resources (compute, network, storage) among tenants.

Cloud multi-tenancy introduces critical security challenges related to the concept of isolation of tenants and shared virtual resources. In multi-tenant cloud architecture, such isolations are a crucial concept for both security and infrastructure management, and they ought to be considered at functional entity levels and appropriate abstraction levels of the infrastructure. In traditional environments, physical isolation is relatively simple as the boundaries between physical elements are well defined and visible [6, 7]. The situation is not clear cut in virtual environments unless one can keep track of all perimeters of all virtual objects created. Defining object boundaries is extremely difficult because virtual objects are dynamic in both characteristics and functionality. The task is resource-expensive due to the sheer number of virtual objects and their complexity.

A number of studies have attempted to address security issues related to security isolation in cloud infrastructure, such as [8], [9], [10], [11], and so on. The existing proposed isolation mechanisms are mostly focused on end-to-end isolation and are mainly classified as network isolation, performance isolation, space isolation, domain isolation, and tenant isolation. The current isolation mechanisms are not so useful in a dynamic and automated infrastructure which requires a dynamic, agile, and automatic construction of security boundaries in the provision of on-demand security functions. Another main security challenge in current mechanisms is a lack of a centralized security orchestrator with an overall view of underlying security infrastructure.

A security breach occurs when a security policy is violated over an interaction. Practically, a security breach is defined in terms of the policies that define the interaction breaches. An event is considered a security breach either when it violates a defined security policy or violates the Confidentiality, Integrity, and Availability of security principles that could have been avoided if a relevant security policy has been in place. The construction of security boundaries in a cloud system is related to the characteristics of the interacting objects in the environment and the policies and constraints that govern their interaction. **However, to the best of our knowledge, there is no previous work to provide isolation in relation to object interaction which can provide dynamic and on-demand security isolation.**

This research addresses these challenges by proposing an intelligent solution to detect and predict security breaches allowed by its policy-based interaction model. The study investigates an innovative algorithm to model the boundaries that can cause violations and create a system to provide on-demand security service. For this purpose, a *Software-defined Security Service (SDS$_2$) model* has been presented to provide on-demand and dynamic security services. We propose SDS$_2$ as *a Software-Defined Security (SDSec) Service* that uses virtual cloud resources and can be deployed by a cloud provider to protect its integrated infrastructure.

The SDS$_2$ introduces an innovative mechanism for the detection and prediction of security breaches in cloud infrastructure. The SDS$_2$ model fashions intelligent, dynamic, automated, and on-demand security functions to protect cloud entities and resources. Our design focuses on building a robust, dynamic, and automated security boundary to protect cloud assets relying on a solid and innovative interaction model and security policy expressions that govern the interactions.

The construction of security boundaries in a cloud system is related to the characteristics of the cloud entities interactions in the environments. The model will introduce a novel policy-driven interaction model as a new security protection trend against cloud infrastructure threats. The SDS$_2$ proposes the policy-based interaction model as a security defence against interaction violations in the cloud system. A security architecture will be designed in alignment with the security model, including three main layers: the security application layer, the security control layer, and the security data layer. The security application layer contains security applications and interfaces. The security control layer accommodates the security controller and its components. The security data layer hosts virtual security functions. To the best of our knowledge, there is a lack of an intelligent, proactive, and on-demand security service in isolation of cloud assets regarding their interaction over a large-scale cloud infrastructure using SDN and NFV techniques.

To realize the cloud infrastructure security model, we address a variety of facing security challenges. We design the security service model, construct intelligent security boundaries according to policies and constraints, build a specific interaction virtual security function, and develop a new communication protocol between the security controller and VSFs. We advance algorithms and mechanisms for security violation detection and prediction.

This research contribution is the proposed software-defined security service framework, architecture for security isolation of cloud's assets, on-demand security service, and a novel policy-driven interaction model to predict and detect security breaches, orchestration provisioning the virtual security functions, and protocol for orchestrating and managing virtual security functions. The model allows the security service to be easily integrated into a large-scale software-defined infrastructure while preserving the simplicity and independency of VSFs from underlying physical and virtual functions. Most importantly, the benefits of software-defined security control, on-demand security interaction monitoring, automation of management and configuration of the virtual security function, dynamic programmability of virtual security functions, and real-time interaction security violation detection and prediction are gained through the merging software-defined networking, network function virtualization, and cloud infrastructure.

Moreover, the significance of this research is its new vision for an efficient and capable cloud security orchestration in the fight against security violations and its novel solution in terms of detection and prediction of security violations. The proposed model can be applied for both cloud providers and their tenants regardless of their environment scale, complexity, and structural sensitivity.

The model is applicable for orchestrating automatic and on-demand virtual security functions by globally connecting security functions via specific security communication protocol and connectivity network by SDN technology. Furthermore, it enables i) security developers to initiate on-demand VSFs without limitation, ii) cloud infrastructure owners to gain more reliability in terms of security complexity, iii) providers and their end-users to benefit from advance agile security detection and prediction iv) cloud tenants to independently secure their organization v) provision of more QoS options for cloud customers.

This chapter is organized as follows. Section 1.2 provides a brief description of the terminologies involved in this thesis. Section 1.3 specifies the research problems addressed by this study. Section 1.4 states the research aim and objectives. Section 1.5 recaps the key contributions of this dissertation. Section 1.6 defines the research model and methodology. Section 1.7 presents the thesis structure.

1.2 Brief Background

Security has been recognized as a critical issue that must be resolved at each domain/level of a cyber-infrastructure. Recently, the integration of new technologies such as cloud and virtualization in modern IT infrastructures makes it more difficult for security experts to protect their systems against numerous security threats due to the virtual resource-sharing nature among tenants and the larger attack surface of clouds. The virtualization and virtualized infrastructure introduce new security challenges related to virtualized resources.

Traditional security measures and endpoint security are no longer adequate due to invisible boundaries created among shared logical and virtual entities among numerous users. To protect cloud resources against modern threats, there is an absolute need for a new isolation approach that can construct dynamic, automated, and on-demand security boundaries according to the object interaction rather than end-point devices. To allow the construction of security boundaries, we need a model with a centralizing overview of cloud resources and their interaction and the ability to construct security boundaries according to the object properties and interactions. Moreover, it can create dynamic, automated, and on-demand security functions to monitor the interactions to detect and predict security interaction violations.

This study focuses on the automation of security solutions for protecting cloud infrastructure by designing and implementing a software-defined security service. This research study's prime motivation is to provide a security software-based platform focusing on delivering dynamic and on-demand security isolation, detecting and predicting security violations in relation to a novel interaction model. The model enables an efficient orchestration of security services on-demand within the cloud infrastructure. This section presents a brief overview of the software-defined infrastructure paradigm, software-defined security (SDSec), software-defined security service, and provision of software-based security functions on-demand.

1.2.1 Software-Defined Infrastructure paradigm

Software-Defined Infrastructure has merged as a promising approach to transfer the operation and control of IT infrastructure entirely as software services. According to [12], Software-Defined refers to "*providing open interfaces to manage and control various sharing resources in different types of infrastructure for software programmability as well as providing access to infrastructure resources like usage data, topology, storage, and compute.*" The SDI is considered an abstraction layer of resource-sharing infrastructure, such as compute, storage, and networking, managed, controlled, and governed using the software. The SDI provides infrastructure capability to function as self-aware, self-scaling, and self-optimizing to enable agile business services and processes [13]. The software-defined infrastructures are presented to pave the way for a faster and dynamic (re)configuration and flexibility of infrastructure resources through software-based functions [14]. The software-defined infrastructure allows companies to deliver IT services with more dynamicity and agility governed by everything-as-a-software concepts. The SDI provides interoperability that enables companies to quickly implement their solutions regardless of types and hardware and their manufacturer. Besides, The SDI significantly improves speed and reduces the complexity of provisioning, deploying, and maintaining resources [15].

The SDI embraces two main concepts that have been presented in the current IT environment, the separation of the network control plane from the data plane, SDN [16], and separation of underlying network functions through software-based virtual functions, NFV [17]. The software-defined infrastructure requires virtual networks from SDN, virtual network functions from NFV, and computing, storage, and orchestration resources from the cloud, but there has not been a standard integrated architecture for SDI, and this presents a considerable challenge in designing a sound framework for an SDI security architecture. The software-defined infrastructure provides an integrated software-based infrastructure consisting of software-defined networking, network function virtualization, and cloud environment where virtualization is adopted as a foundation technology.

The virtualization technology empowers NFV and SDN to create scalable, dynamic, and automatically programmable virtual network functions and virtual network infrastructures in integrated cloud platforms such as telecom clouds. However, virtualization and creation of software-based virtual components within the cloud introduce new security challenges and exacerbate those existing ones in each domain. The primary security problem arises from creating numerous security boundaries that are often hidden or invisible within virtual environments. The proposed model is a software-based security service running on top of cloud software-defined infrastructure. The security services are software-based virtual security functions allocating in different locations according to triggered interaction.

1.2.2 Software-Defined Security (SDSec)

Software-defined security (SDSec) has been introduced to assist security experts in handling automatic security enforcement in distributed environments [18]. The SDSec architecture decouples the security control plane from the security forwarding plane (including software-based security instances like firewall, intrusion detection systems (IDS), deep packet inspection (DPI)) in the same way SDN isolates the control plane from the network data plane. It offers security functions as software instances independent from traditional physical security appliances. Security measures can be deployed effectively and rapidly based on the changing levels of system/business requirements. SDSec has been developed as a result of the inability of traditional security methods to cope with a mixture of virtual and physical elements in modern infrastructures.

In [19], the authors mentioned six main outstanding features and attributes that distinguished the SDSec approach from traditional security approaches: i) abstraction: abstracting security policies from the underlying hardware plane which runs in an independent software layer; ii) automation: automated programmable creation of software-based functions and configuring security functions without the need of manual configuration; iii) elasticity: software-based security resources can be provisioned elastically; iv) concurrency of control: providing a higher level of security control over the virtual environment; v) visibility; and vii) portability.

Cloud Security Alliance (CSA) introduced the idea of SDSec with Software-Defined Perimeter (SDP) to facilitate a new security architecture that is resilient against network attacks and achieves security with higher visibility and lower costs. The CSA described SDP as follows: SDP is a framework of security controls that mitigate network-based attacks on Internet-accessible applications by eliminating connectivity to them until devices and users are authenticated and authorized. The SDP reduced security attacks on network applications by disconnecting applications until a proper authentication of both users and devices [20, 21]. In addition, there are several commercial products which consider the SDSec approach like Catbird [22], vArmor [23], vShield [24], and OneControl [25].

Catbird implements several features and attributes that distinguish the SDSec approach from traditional security approaches. Catbird consists of two main elements: Catbird control center and a set of virtual machine appliances (VMAs) implemented as VMs. The system configures a mesh topology where the Catbird control center is located at the center of the network as the policy enforcement point to manage and distribute the security controls across the connected VMAs. There is a Linux-based VMA (virtual memory address) implemented inside it for every virtual switch, executing different security tasks through a hypervisor interface [22].

vArmour is another SDSec solution that exploits the benefits of virtualized environments. The architecture of vArmour is like any software-defined system architecture, where the control plane is decoupled from the forwarding plane. The vArmour Distributed Security System consists of a logically centralized controller and multiple autonomous enforcement point appliances connected by an intelligent fabric. It constitutes a security (SDSec) service layer to enforce a security rule to a whole data centre [23].

vShield is another solution for VMware vCloud. vShield provides the customer the ability to build policy-based groups and establish a logical boundary between them. vShield integrates several components: vShield App and Zones protect the virtual data center applications by creating segmentation between enclaves or silos of workloads. vShield Edge secures the edge of the virtual data center boundary and defends the communication between segmentations. vShield Endpoint offloads antivirus processing. vShieldManger provides a centralized control point to manage all vShield components [24].

1.2.3 Software-Defined Security Service (SDS$_2$)

The software-defined security service provides control, management, orchestration of security services on-demand based on its specific virtual security functions (VSFs). The SDS$_2$ offers a security model as well as a security service that relies on the object-oriented entities of a cloud environment, the interaction among them, and security policies that govern the interaction.

The proposed model consists of three main layers and essential components: security controller, interaction monitoring functions, virtual security networking, and intelligent algorithm. The security controller is the centralized security intelligence with overall visibility on cloud object interactions. Interaction monitoring functions are an essential feature of the security model in detecting and predicting security violations according to the intelligent algorithm. The virtual security networking function is in charge of transferring policies and interaction values between security controller-to-monitoring functions, monitoring function-to-monitoring function, and monitoring function-security controller. The proposed model enables the dynamic construction of security boundaries concerning the initiation of an object's interactions.

The SDS_2 enables automation and programmability of interaction virtual security functions to monitor a targeted interaction. It can monitor interactions based on two different states: 1) monitor based on a requested interaction; 2) monitor a random interaction/event within the system.

1.2.4 Software-Defined Network of Virtual Security Functions

Software-defined networking separates the control plane from the underlying network data plane for efficient data transport and fine-grained control of network management and services. SDN allows network virtualization and provision of virtual networks on-demand. SDS_2 decouples security functions and security networks from the underlying infrastructure. A communication protocol has been introduced to transfer security messages and interaction parameters between the security controller and VSFs.

The network connectivity inherits the fundamentals of SDN architecture using a solid policy-based interaction protocol. It works as a bridge between the security controller (SC) and the VSFs to transfer interaction values according to the SDS_2 Interaction Model. The main aims of designing the Sec-Manage protocol are 1) to provide direct communication between the SDS_2 security controller and its VSFs and 2) to transfer the parameters pertinent to the security aspects of objects' interaction between a VSF and the SC to monitor parameters of interaction to detect and predict security violations.

1.2.5 Provision of software-based security functions on-demand

Virtual network functions (VNFs) are defined as software-based functions decoupled from underlying physical network functions. There are various types of VNFs in various contexts. A virtual security function is a type of VNF with security functionality rather than networking. In the proposed SDS_2 model, one of the main functions includes our specific interaction virtual security function.

A VSF inherits some properties of VNF, which precisely emphasize detection and prediction of security interaction violations. A VSF in our usage is created to perform a specific security function and deployed at strategic locations in the cloud infrastructure that requires protection. It is a software-based function constructed to protect cloud infrastructure against any type of interaction violations. The VSF is a simple but efficient and intelligent security function, monitoring cloud entities interaction to detect and predict security violations. The SDS_2 dynamically triggers VSF to monitor an interaction on-demand.

1.2.6 Security Issues and Challenges in an Integrated Cloud/SDN/NFV Infrastructure Platform

Cloud computing demonstrated how best computing and storage resources could be virtualized and provisioned on-demand and offered as IT services. More importantly, its effective orchestration of services provides an excellent model for resources and service management. SDN and NFV demonstrated the most effective way network resources and services (network infrastructures, network functions, and connectivity services) can be created and managed. Cloud needs SDN and NFV to be integrated seamlessly to offer truly any resource as a service. SDN and NFV need to include cloud management infrastructure to provide network services and functionality. For example, existing telecommunications network infrastructures and service models are too rigid. They have to evolve into a telecom cloud to offer emerging and flexible services to their customers. An integrated software-defined infrastructure that seamlessly integrates cloud, SDN, and NFV will create a robust service model that incorporates all the best features of these technologies.

Two significant issues concerning cloud, SDN, NFV, and the integrated software-defined infrastructures are the security of the virtualization technology itself and the complexity of the virtualized interconnecting infrastructure. Cloud and SDN networks face an increasing intricacy of emerging social networks, applications, and services and their associated security problems. The whole range of issues includes scalability of cloud networks, the complexity of the way network functions communicate to each other, the lack of a centralized infrastructure control component, policy enforcement, dynamic workloads, multi-tenancy, isolation of tenants, services, resources (virtual networks, virtual machines, virtual storage). SDN and NFV allow tenants to share the underlying physical network to create their virtual networks, network functions, and services with their policy in a cloud environment. Integrating cloud, SDN, and NFV into a software-defined infrastructure provides a truly scalable, dynamic, and automatic programmable platform for creating everything as a service on demand.

All these infrastructures rely on virtualization as the core technology. Virtualization is pervasive in almost all components of the service infrastructures: virtual machines, virtual networks, virtual storage, virtual network functions, and virtual services. However, virtualization brings with it new security challenges in the way virtual elements are created and maintained. For the security of the infrastructure, all virtual elements have to be secure for their whole lifecycle; their creators (hypervisors) must be trusted and secure; appropriate isolation among servers, among services, and tenants must be preserved.

Although integration of cloud, SDN, and NFV into a service infrastructure provides benefit to both service providers and service users, the complexity of security of each technology, of virtual components, of individual infrastructures present a significant obstacle for comprehensive integration. One important aspect of virtualization is that it introduces invisible boundaries to traditional security mechanisms at various levels. To deal with this integrated software-defined infrastructure, one should use the very virtualization technology to provide security of the overall infrastructure; one should deploy the logically centralized paradigm of

SDN and NFV to separate security control from the functionality of security network functions. Software-Defined Security Service (SDS_2) proposed in that spirit to create a centralized security service model for the cloud-SDN-NFV infrastructure platform. The SDS_2 provides a centralized security controller over the infrastructure. The SDS_2 controller will possess the ability to create its flexible interconnecting infrastructure for connecting its security function elements. It will have the ability to program and manage its security function elements autonomously.

1.3 Research Questions

The major obstacle for organizations on complete migration through the cloud is considered as a security. As described, there are numerous security challenges pointed to the cloud infrastructure, especially on security isolations. Some of the significant security challenges are listed as a lack of proper visibility on security functions within a cloud infrastructure, lack of provisioning dynamic security monitoring orchestrator, absence of an efficient security violation prediction mechanism, and deprived efficient dynamic security isolation mechanism.

In our research, we seek answers to the following questions "How can security services dynamically construct security boundaries?", "How to automatically predict the next security breach within the system?", "How to deliver programmable virtual security functions?", "How to provide a centralized security controller with visibility on underlying security functions?" and "How to integrate the virtual security functions within the cloud infrastructure?". Three significant challenges have been identified. Initially, a virtual security function requires dynamic, automated, programmable, and on-demand capabilities to be integrated into cloud infrastructure with advanced detection and prediction techniques for providing cloud security. Secondly, it is essential to offer a centralized orchestrator to manage, control, and configure security functions over a large-scale distributed environment like a cloud. Lastly, an approach should be introduced to deliver dynamic security isolations considering costeffectiveness and efficiency. In brief, the research question addressed in this thesis can be specified as follows:

"How to secure and protect cloud infrastructure against security isolation breaches using new technologies based on SDN/NFV, and can the proposed model be realized in a practical environment?"

To address the issue, we investigate the following research questions.

- **Can SDN-NFV technologies be ported to the cloud security domain where they demonstrate their proficiency in the programmability and automation of on-demand security services?**
- The concepts of SDN and NFV in managing and controlling network function can be fully applied to our proposed model. However, to adapt these technologies to a security model, there are quite formidable changes to apply since these technologies cannot directly be applied to security. Virtual security functions are defined for specific security activities. Unlike network functions, virtual security functions have restricted functionalities in terms of resources and communication techniques. They are required to communicate with the security controller and other VSFs within the system. They do not need to handle massive volumes of messages transferred in an SDN network. The number of security messages is limited as well as required computational resources. The VSFs do not require complex service life chaining and properties compared to virtual network functions. It is challenging to apply the SDN/NFV entirely over a security platform in practice.
- **How can we discover an efficient security isolation technique to detect and predict security breaches in virtual environment?**

The research conducted in this thesis seeks to answer this question by proposing an intelligent algorithm for detecting and predicting security violations. Security breaches primarily result from some violation of the rule of interaction (or policy that governs the interaction) between objects when they interact. Unless one has a formal model of an interaction between objects, it is difficult to detect, predict or prevent security incidents. In this research, we propose a novel policy-driven interaction model that provides an innovative interaction structure foundation. The proposed model responds to this question by representing the interaction model characteristics and techniques to secure cloud objects according to their initiated interaction.

- **How to deliver dynamicity, programmability, and automation in relation to on-demand security services?**

To respond to this question, we proposed a Software-Defined Security Service (SDS_2) model to secure cloud infrastructure. The proposed model contains main components, including a logically centralized security controller, virtual security functions, policy-driven interaction protocol. The SDS_2 offers many key benefits to enterprise cybersecurity, including simplified security management and orchestration, visibility on logical security boundaries, dynamics and intelligent security detection and prediction mechanism, and agile security response to security breaches.

- **How to validate and evaluate a security model in a real deployment?**

The proposed model is required to be validated and evaluated in practice. Currently, there is no open-source software-defined security simulator, emulator, or integrated security platform to enables the implementation of our model on top of their systems. Moreover, there are no previous works on security following our interaction model to construct security isolation within the cloud infrastructure. To validate our proposed model, it is essential to design and develop a software-based environment to implement all required components, including our security controller, a virtual security function, and communication protocol. To validate the model and its security algorithm, we run various tests.

1.4 Research Aim and Objectives

The main aim of this thesis is to develop an intelligent security solution to protect cloud infrastructure that enables definable security boundaries based on an intelligent model to dynamically define realizable security boundaries aligned with new technologies like SDN/NFV. The security solution practically models security boundaries to detect and predict security violations by introducing a software-defined security architecture in relation to an innovative interaction model to secure the cloud's resources. So, the main focus is on developing a security platform to establish dynamic and intelligent security boundaries, new techniques to detect and predict security breaches, and control structures for provisioning/coordinating resources for counterattacks.

To accomplish the aim, we define following objectives and investigate practical solutions for their achievements.

Objective 1: Detecting and establishing dynamic security boundaries in the cloud environment. To achieve this objective, we divide it according to six sub-objectives as follows:

- Investigating existing defined physical and virtual boundaries, their issues, and solutions associated with providing on-demand and dynamic security boundaries
- Defining characteristics and functionality of cloud's objects in relation to defining security boundaries
- Inspecting policy-based structure in designing object interactions in cloud
- Proposing a Policy-based Interaction model intended for the construction of dynamic, secure boundaries according to defined interaction parameters
- Defining an intelligent interaction algorithm for cloud security breaches detection and prediction
- Establishing dynamic security boundaries according to the policy-based interaction model

Objective 2: Deploying an innovative security model to secure and protect cloud infrastructure and provision on-demand security services. The sub-objectives for this step are explained as:

- Designing a software-defined security model that can be integrated into cloud infrastructure to provide dynamic and agile security service to both cloud and its customers
- Proposing a software-defined security service (SDS_2) architecture to orchestrate, manage and establish defined dynamic security boundaries which align with the policy-based interaction model
- Creating security controller and its components with the purpose of establishing dynamic security boundaries, detection and prediction of security breaches
- Fashioning on-demand, intelligent, and agile Virtual Security Functions (VSFs) using Network Function Virtualization (NFV) techniques
- Realizing a policy-based interaction protocol between the software-defined security controller and virtual security functions based on SDN technology
- delivering a dynamic and on-demand security framework that allows orchestration, control, and management of interaction virtual security functions to protect an extensive distributed cloud infrastructure

Objective 3: Evaluating the feasibility and proficiency of the proposed model and its functions.

- Validating and evaluating our proposed security technique by implementing our Software-Defined Security architecture and Services (SDS_2) and demonstrating the platform performance evaluation.

1.5 Research Contributions and Significance

This research concentrates on securing cloud infrastructure in the provision of cloud security services on-demand. This thesis researches a novel interaction model for detecting and predicting interaction violations to initiate a new technique in constructing security isolation in cloud infrastructure. To our knowledge, there is no previous study or framework that proactively secures a large-scale distributed infrastructure like a cloud using an interaction model between resources at a high-level. Expected outcomes in relation to the above objectives are as follows:

- Software-defined security framework introduces a new knowledge on constructing dynamic and on-demand security boundaries in cloud infrastructure. The construction of security boundaries in a cloud system is related to the characteristics of the interacting objects in the environment and the policies and constraints that govern their interaction. The novelty of this approach is that it is a paradigm to build a robust, dynamic, and automated security boundary to protect cloud assets relying on a solid and innovative interaction model and security policy expressions that govern the interactions. The framework exploits four main concepts: logical centralization of security control, virtualization of secure connectivity, security functions virtualization, and orchestration of virtual resources.
- The proposed interaction model represents a new technique for detecting and predicting security breaches. The model governs the interactions among entities in a cloud environment. The proposed model and its introduced algorithms protect cloud resources against security threats via defined security boundaries constructed from the model and the system security policies that govern the interaction model.
- The proposed novel protocol opens up a new research area on communication security protocol between the security controller and virtual security controller using SDN/NFV technologies. The Sec-Manage protocol transfers security messages and interaction parameters between a security controller and its VSFs. The protocol is a novel approach in programming behaviour and configurational management of VSFs according to the proposed interaction model.
- The proposed architecture of dynamic and programmable software-defined security service (SDS_2) will be integrated within the cloud infrastructure. The

security service creates its specific virtual security function with the capability of agile response to security threats. SDS_2 offers many critical benefits to enterprise cybersecurity, including simplified security management and orchestration, visibility on logical security boundaries in the physical and virtual environment, dynamics and intelligent security configuration, and agile security response to security breaches.

- The proposed model is implemented in a cloud/SDN/NFV integrated software platform for practical realization. A software platform has been developed to evaluate and validate the proposed model and its introduced components. The innovation is a security software platform demonstrating the ability of SDS_2 in protecting cloud infrastructure using SDN/NFV concepts.

The research contributions are of Significance not only for cloud stakeholders such as end-users, tenants, security admins, security developers but also, they provide a significant impact on cybersecurity society.

For cloud providers, the proposed SDS_2 and its novel interaction model provide an intelligent proactive centralized security controller with a management and orchestration mechanism for providing on-demand virtual security functions. Moreover, the proposed security service enables cloud developers to develop their applications without concern over security protection. Tenants of cloud can use the SDS_2 in their environment to protect against internal and external attacks independently and without concerns on deployment cost and difficulty in programming and controlling the virtual security functions.

1.6 Research Methodology

There are several main phases in this research approach that should be considered in order. The selected methodology is adapted from [26]. This research is divided into five phases, as shown in Figure 1.1.

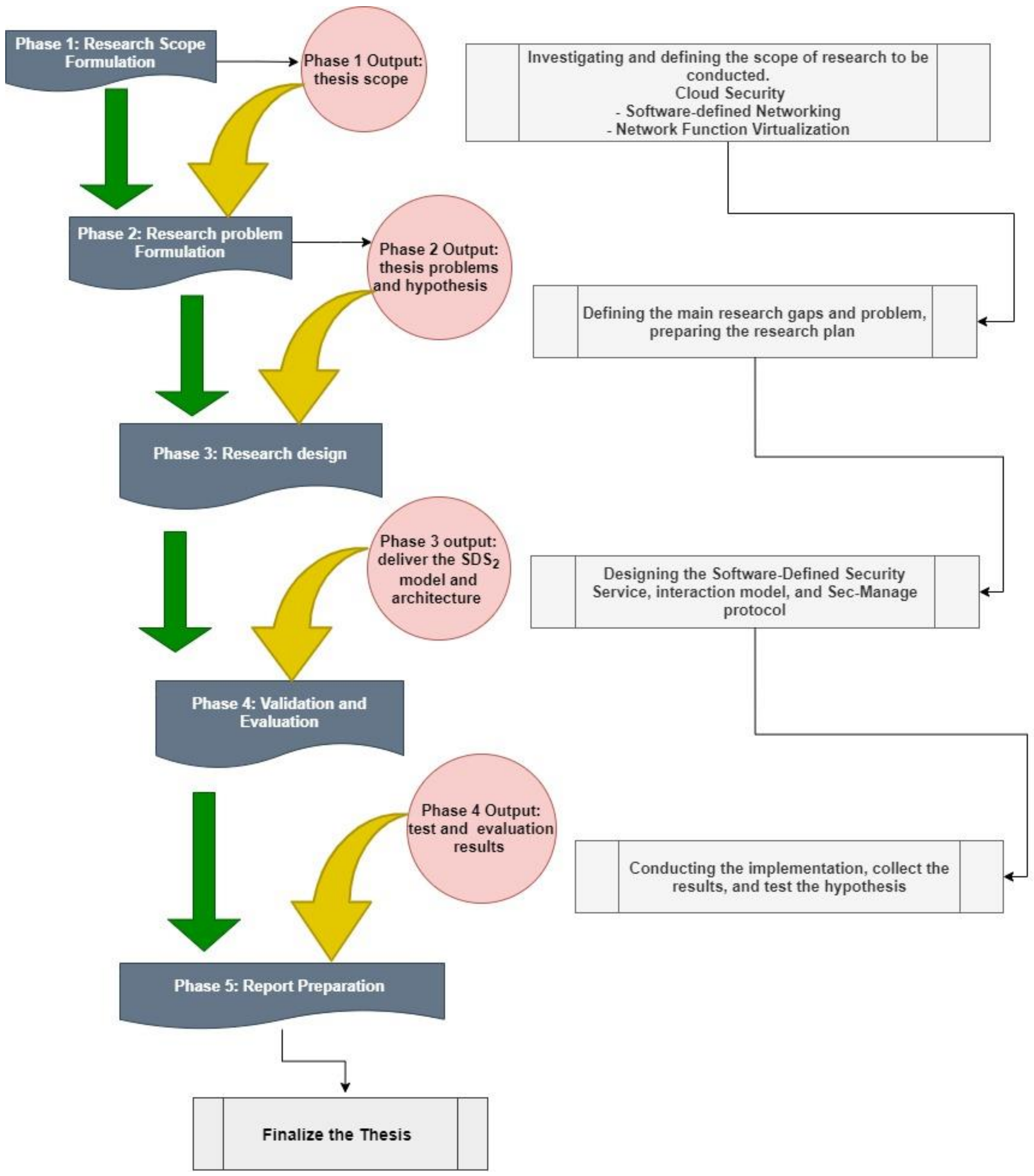

Phase 1: Research Scope Formulation
Phase 1 Output: thesis scope
Investigating and defining the scope of research to be conducted.
Cloud Security
- Software-defined Networking
- Network Function Virtualization
Phase 2: Research problem Formulation
Phase 2 Output: thesis problems and hypothesis
Defining the main research gaps and problem, preparing the research plan
Phase 3: Research design
Phase 3 output: deliver the SDS2 model and architecture
Designing the Software-Defined Security Service, interaction model, and Sec-Manage protocol
Phase 4: Validation and Evaluation
Phase 4 Output: test and evaluation results
Conducting the implementation, collect the results, and test the hypothesis
Phase 5: Report Preparation
Finalize the Thesis

Figure 1.1 Research Phases

The first phase is defining the scope of the research. This phase identifies the exact scope of the research, which includes several technologies. The second phase, is to recognize the research problem. In this phase, we study the current state-of-art on diverse previous works according to defined research scope. It includes involved technologies for building our model. In this phase we identify the research problem as well as current existing solution to the issue. After this phase, we gain a knowledge on the discovered problem, existing solution to the problem, and gaps in the proposed solutions. We classify the hypothesis concerning security problems where we prepare the research aim, research objectives, research questions, and significance. In the next phase, we design our proposed service model. We propose our new approaches and mechanisms for the research problem. In this phase, we create the model and new approaches to achieve our research objectives. The output defines our new solution to issue. The proposed solution requires to be validated and evaluated. In phase four, we validate and gather the data and results to evaluate the proposed solution and determine whether we need to refine the proposed model or not. In the last phase, we collect all the information and write a thesis to describe our work and future work. Figure 1.2 demonstrates all phases and their details.

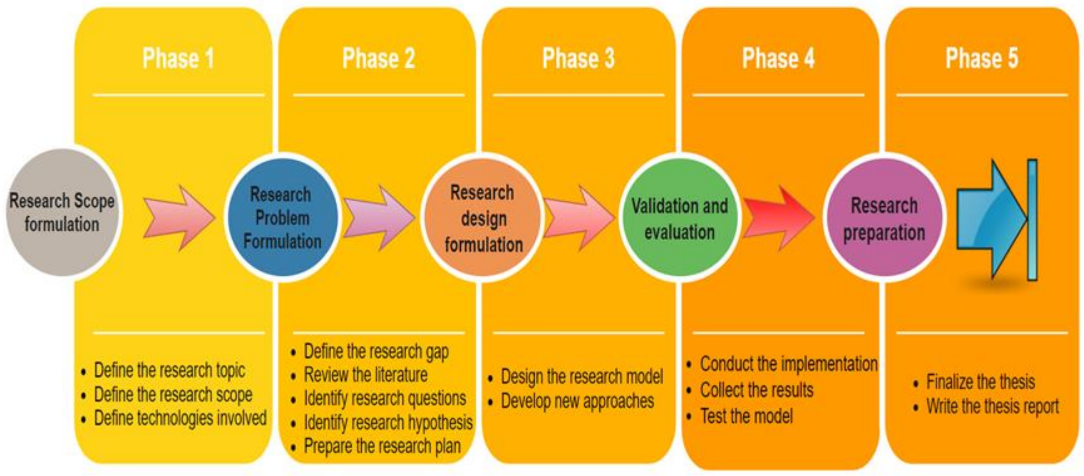

Phase 1
Phase 2
Phase 3
Phase 4
Phase 5
Research Scope formulation
Research Problem Formulation
Research design formulation
Validation and evaluation
Research preparation
• Define the research topic
• Define the research scope
• Define technologies involved
• Define the research gap
• Review the literature
• Identify research questions
• Identify research hypothesis
• Prepare the research plan
• Design the research model
• Develop new approaches
• Conduct the implementation
• Collect the results
• Test the model
• Finalize the thesis
• Write the thesis report

Figure 1.2 Research Methodology

1.7 Thesis Structure

This research has produced several papers, including three conferences, two journal papers, and one book chapter published in Springer. Figure 1.3 demonstrates the structure of the thesis. This thesis is organized into eight chapters as follow:

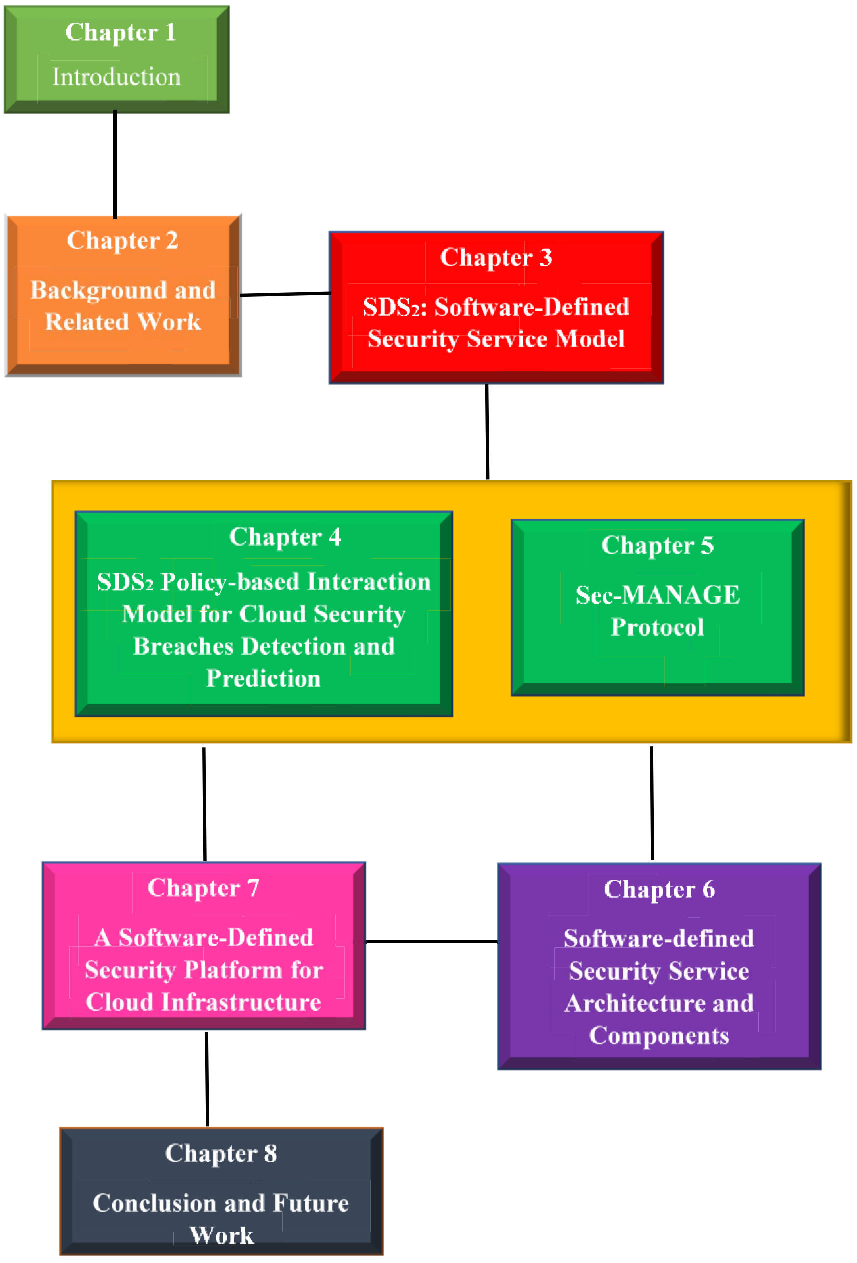

Chapter 1
Introduction
Chapter 2
Background and Related Work
Chapter 3
SDS2: Software-Defined Security Service Model
Chapter 4
SDS2 Policy-based Interaction Model for Cloud Security Breaches Detection and Prediction
Chapter 5
Sec-MANAGE Protocol
Chapter 7
A Software-Defined Security Platform for Cloud Infrastructure
Chapter 6
Software-defined Security Service Architecture and Components
Chapter 8
Conclusion and Future Work

Figure 1.3 Research Structure

- **Chapter 1: Introduction**

This chapter presents an overview of this research study. It introduces the importance of a new security approach for protecting cloud infrastructure using a novel SDS$_2$ security platform concerning the provision of agile security services on-demand. This chapter presents the research problem, research aim, objectives, research contribution, research methodology, and thesis structure.

- **Chapter 2: Background and Related Work**

This chapter provides a background on Software-Defined Infrastructure (SDI) and revolutions in terms of security challenges and technologies, security challenges, enabling technologies for a software-based security platform, virtual enabling, and resources for deployment of on-demand security services, and SDI architecture and models. As a leading and fundamental technology, a brief explanation has been presented in relation to virtualization and cloud infrastructure and its deployment models and security challenges.

This chapter also provides a brief background on the involved open-sources platform required for the practical implementation of the proposed SDS_2 model. The chapter describes SDN and NFV architecture and techniques used for this research study. It reviews both security challenges and solutions applying in SDN and NFV. This chapter discusses the integrated Cloud/SDN/NFV solution to the provision of a programmable and automated security system.

- **Chapter 3: SDS_2: Software-Defined Security Service Model**

This chapter presents a broad overview of the proposed software-defined security service for provisioning on-demand security services. It represents the significant contribution of the proposed model in relation to protecting cloud infrastructure.

- **Chapter 4: SDS_2 Policy-based Interaction Model for Cloud Security Breaches Detection and Prediction**

This chapter presents the proposed novel, policy-driven interaction model. We demonstrate the structure of the interaction model and its specific parameters. The algorithms for detecting and predicting security breaches in relation to the interaction model are presented in this chapter.

- **Chapter 5: Sec-MANAGE Protocol**

This chapter demonstrates the design and specification of the proposed Sec-Manage protocol to configure and manage VSFs to provide on-demand security services. The protocol is designed to deal with security interaction constraints in cloud infrastructure. In this chapter, the detailed design and operation of the Sec-Manage protocol are described. Furthermore, this chapter also presented implementation results related to transferring the interaction parameters between VSFs and controller and an on-demand allocation of VSFs within the system.

- **Chapter 6: Policy-based Software-defined Security Service Architecture and Components**

This chapter presents an overall architecture of SDS_2 in provisioning security services on-demand. It presents the detailed structure of SDS_2 components, including the SDS_2 controller, and virtual security functions.

- **Chapter 7: A Software-Defined Security Platform for Cloud Infrastructure and Evaluation**

This chapter presents the developed software-defined security platform in provisioning VSFs on-demand for detection and prediction of security violations. This chapter demonstrates the practical implementation of the SDS_2 model and discusses the results.

- **Chapter 8: Conclusion and Future Works**

Chapter 8 summarizes the ideas presented in this thesis, the major research contribution, and outlines future research work.

Chapter 2

Background and Related Work

2.1 Introduction

Software-defined networks, network functions, virtualization platforms, and clouds have established themselves as modern IT service infrastructures. They rely on virtualization technology to virtualize and aggregate physical resources into pools of virtual resources (virtual machines, virtual networks, virtual storage, virtual functions, and virtual services) and provision them to users on demand. Security has been recognized as an essential and integral part of the design of systems, infrastructures, organizations, and services; yet, the current state of security research and practice is at best fragmented, local, or case-specific.

This chapter presents background on software-defined infrastructure and the provision of on-demand security services within the cloud infrastructure. We explore concepts of security isolation and related challenges in maintaining isolation in large-scale infrastructure. We review various technologies, including virtualization, software-defined networking, network function virtualization, and cloud computing, while exploring security challenges entangled with these technologies.

Designing the proposed SDS_2 requires in-depth knowledge of involved technologies and their deployment models. We provide an overview of the open-source platform to be used, such as the OpenFlow protocol [27], OpenStack [28], CloudSimSDN-NFV framework [29], NFV platforms, and SDN controllers.

The rest of this chapter is organized as follows. Section 2.2 gives a brief overview of software-defined infrastructure and virtualization as the primary fundamental technology. Section 2.3 presents cloud computing and its related security challenges. Section 2.4 provides a background of the SDN technique and presents major security issues. Section 2.5 describes the NFV technique and its security challenges. Section 2.6 provides an overview of policy and security policy mechanisms in the cloud. Section 2.7 includes literature on works based on isolation and describes security isolation in cloud computing. Section 2.8 briefly introduces open sources for developing a cloud SDN-NFV-based security system. Section 2.9 summarizes this chapter.

2.2 Software-Defined Infrastructure

Software-Defined Infrastructure (SDI) is a resource-sharing infrastructure that embraces the concept of separation of the network control plane from its data plane and software realization of network functions from the underlying hardware appliances. It established itself as an efficient approach to designing and deploying modern IT service infrastructure by integrating Software-Defined Networks, Network Functions Virtualization platforms, and Clouds.

The SDI combines Software-Defined compute (SDC), Software-Defined Networking (SDN), and Software-Defined Storage (SDS) into a fully software-defined data center for providing simplified and standardized IT consumption models, with automatic configuration, ease of management, and centralized visibility over the infrastructure functions and resources. The integrated SDI infrastructure adopts virtualization as a fundamental and vital technology for its SDN, NFV, and cloud constituents. They rely on virtualization to create virtual resources, including virtual networks (VNs), virtual network functions, virtual storage, virtual machines (VMs), and virtual services. This section explores virtualization as a key technology in SDI and investigates security challenges related to this technology.

2.2.1 Virtualization

Virtualization is the technology that simulates the interface to a physical object by multiplexing, aggregation, or emulation. With multiplexing, it creates multiple objects from one instance of a physical object. By aggregation, it creates one virtual object from multiple physical objects. Through emulation, it constructs a virtual object from a different type of physical object [30]. On another level, virtualization can be defined as the logical abstraction of assets, such as the hardware platform, operating system (OS), storage devices, network, services, or programming interfaces. Virtualization technology plays an essential role in the development and management of services offered by a provider.

More commonly, virtualization is introduced as a software abstraction layer placed between an operating system and the underlying hardware (computing, network, and storage) in the form of a hypervisor. A hypervisor is a small and specialized operating system that runs on a physical server (host machine), allowing physical resources to be partitioned and provisioned as virtual resources (virtual CPU, virtual memory, virtual storage, and virtual networks).

A hypervisor creates and manages virtual machines on computing resources, which are isolated instances of the application software and guest OS that run like separate computers. A virtual machine (VM) encapsulates the virtual hardware, the virtual disks, and the application metadata. Since the hypervisor manages the hardware resources in cloud data centers, multiple virtual machines, each with its operating system and applications and network services, can run parallel in a single hardware device [31]. Figure 2.1 illustrates the virtualization of virtual machines.

Virtualization technology has been deployed by enterprises in data centers storage virtualization (NAS, SAN, database), OS virtualization (VMware, Xen), software or application virtualization (Apache Tomcat, JBoss, Oracle App Server, Web Sphere), and Network Virtualization [32]. Virtualization technology enables each cloud tenant to perform its own services, applications, operating systems, and even network configuration in a logical environment without considering underlying physical infrastructure [33].

Virtualization is a key technology for cloud computing, SDN, and NFV. The technology enables network functions virtualization and software-defined network to create scalable, dynamic, and automated programmable virtual network functions and virtual network infrastructures in integrated cloud platforms such as telecom clouds. Virtualization is the foundation of integrated software-defined infrastructure (cloud/SDN/NFV), which allows the abstraction of the underlying resources for sharing with other tenants, isolation of users in the same cloud/network, and isolation of services functions running on the same hardware.

Virtualization allows elastic and scalable resource provisioning and sharing among multiple users. The technology allows multi-tenancy in clouds through an isolation mechanism. It enables each cloud tenant to perform its own services, applications, operating systems, and even network configuration in a logical environment without concerns over the same underlying physical infrastructure. Virtualization results in better server utilization and server/data center consolidation (multiple VMs run within a physical server) and workload isolation (each application on a physical server has its own separate VM).

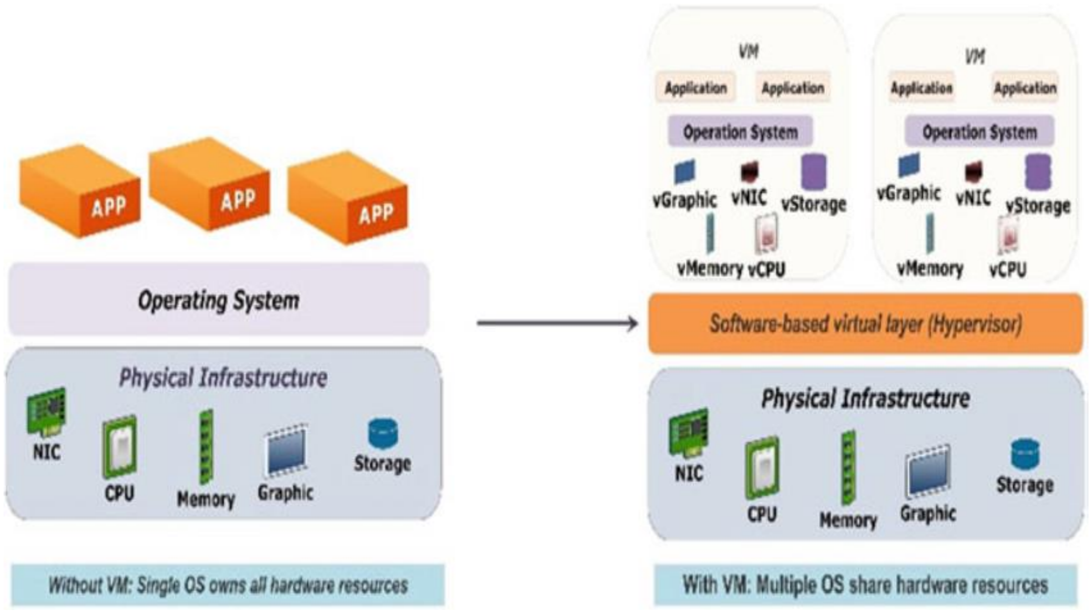


Figure 2.1 Virtual Machines Virtualization

❖ **Security Issues - Virtualization**

With virtualization, the complete state of an operating system and the instances of the application software together with their associated virtual hardware, disks, and metadata are captured by the VM. This state can be saved in a file, and the file can be copied and shared. Creating a VM reduces ultimately to copying a file. VM is an essential component of the cloud, SDN, and NFV. In SDN, a virtual network is created (virtualized) from the underlying network resources, and its virtual image can be captured by a file. Within this file, VMs exist as network elements (switches, routers, and communication links) of the virtual network. In NFV, a single VM or multiple VMs capture the complete state of a VNF instance which can be recorded as a file. In the architecture of these infrastructures, a hypervisor is a centerpiece that performs the task of virtualizing resources.

Virtualization thus brings with it all the security concerns of the guest operating system, along with new virtualization specific threats, including hypervisor attacks, inter-VM attacks, inter-virtual network attacks, and inter-virtual function attacks [34].

This part describes a number of fundamental security issues pertaining to virtualization and virtual environments.

Software Life Cycle of Virtual Image Object The traditional assumption is that the software life cycle is sequential on a single line, so management processes progress monotonically along the sequence. However, the virtual execution object model maps to a tree structure rather than a line. At any point in time, multiple instances of the virtualized entity (e.g., VM, VNF) can be created, and then each of them can be updated, different patches installed, and so on. This problem has profound implications for security [30].

The Indefinite Attack in a Virtual Environment Some of the infected VMs, VNs (Virtual Network), and VNFs may be dormant at the system clean up time, and later, they could surface and infect other systems. This scenario can repeat itself and guarantee that infection will perpetuate indefinitely. In the non-virtual environment, once an infection is detected, the infected systems are quarantined and then cleaned up.

Rollback VM Attack Rollback is a feature that reverts all changes made by a user to a virtual machine when the user logs off from the virtual machine. As the complete state of a VM can be recorded, the feature opens the door for a new type of vulnerability caused by events recorded in an attacker's memory. The first scenario is that one-time passwords are transmitted in the clear, and the protection is not guaranteed if an attacker can replay rolled-back versions and access past sniffed passwords.

The second scenario is related to the requirement of some cryptographic protocols regarding the freshness of the random-number source used for session keys and nonce. When a VM is rolled back to a state where a random number has been generated but not yet used, the door is left open for protocol hijacking [30].

Security Risks Posed by Shared Images A user of a public cloud such as Amazon Web Service (AWS) has the option to create an image (Amazon Machine Image, AMI) from a running system, from another image in the image store, or from the image of a VM and copy the contents of the file system to the bundle. Three types of security risks were identified and analyzed: (1) backdoors and leftover credentials, (2) unsolicited connections, and (3) malware. The software vulnerability audit revealed that 98% of the Windows AMIs and 58% of Linux AMIs had critical vulnerabilities [30].

Hypervisor Security Another critical security issue in virtualized environments is hypervisor vulnerabilities. A hypervisor creates virtual resources (VMs, VNs, and VNFs) inside the SDI and has the ability to monitor each of them. This feature introduces a high-security risk in terms of confidentially, integrity, availability, authenticity, and accountability. It may allow an attacker to view, inject, or modify operational state information connected with the SDI through a direct/indirect method. As a result, the attacker is able to read/write the contents of resources such as memory, storage, and other components of the SDI.

Hypervisor hijacking is a type of attack that allows an adversary to take control of a hypervisor and access all VMs created by that particular hypervisor or other less secure hypervisors in the infrastructure. In the worst case, it may even introduce misconfigurations in SDN controllers when integrated with NFV technology. Furthermore, existing errors or bugs inside a virtual function or a hypervisor may allow an attacker to compromise other virtualized network functions for more serious attacks.

A research was conducted on virtualization security issues to address the security of virtual system models. It discusses various security issues posed by virtualization according to a different classification. They perform a security analysis according to a reference architecture shown in Figure 2.2. They determined the threats affecting the architecture and analyzed the vulnerabilities and possible attacks [35].

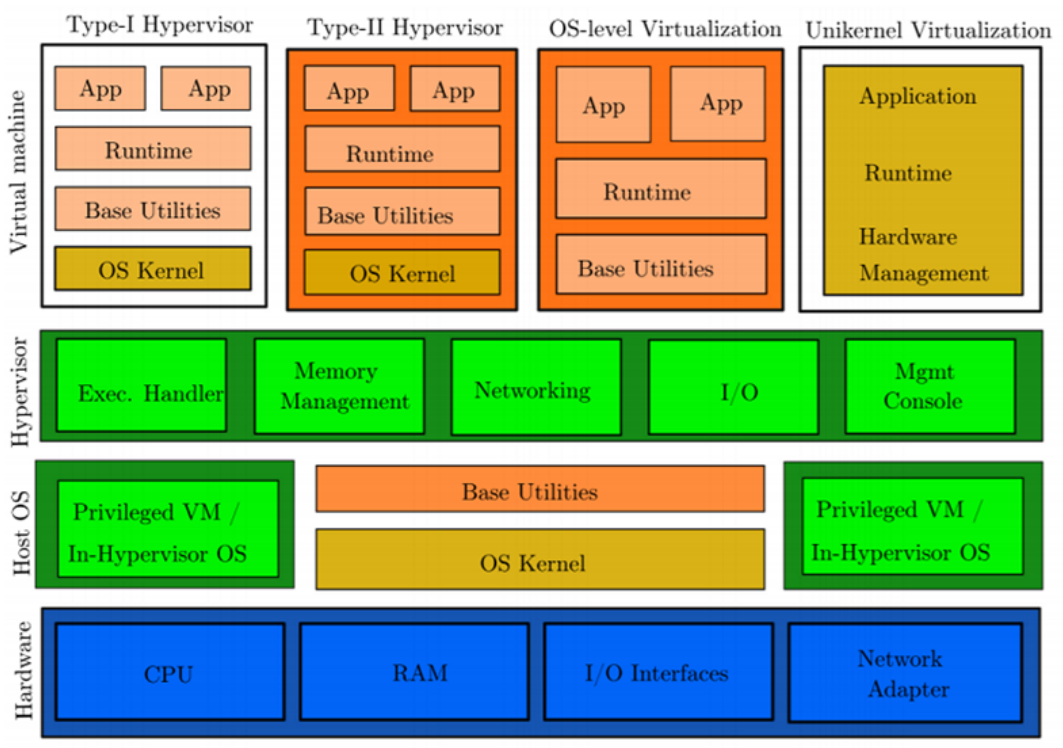


Figure 2.2 Virtualization Reference Model [35]

The research detailed vulnerabilities in four categories with respect to the virtualization model: VM application, the VM guest OS, the hypervisor, and the execution environment of VMM [35].

The VM applications are related to memory management and software interfaces. The memory management includes runtime variable type checking, kernel interface in user space, deallocation of memory, and development of software flows. Software interface vulnerabilities are connected to improper operation and configuration of access control mechanisms, code injections, and concurrency vulnerabilities (related to improper synchronization mechanisms) [35].

VM guest OS is related to possible vulnerabilities caused by software management and OS kernel oversight. Software management can trigger vulnerabilities related to solving dependency, degradation of services, and configuration issues. Each vulnerability raises the risk of sophisticated attacks in a virtual environemnt [35].

The hypervisor is prone to different vulnerabilities related to VM-hypervisor crosstalks, inter-VM crosstalk, and management console. The VM-hypervisor vulnerabilities refer to resource isolation issues, resource sharing with host, hypervisor oversight issues, and implementation issues). Inter-VM vulnerabilities are considered to be based mainly on the isolation of resources shared among various VM which are sharing the same hypervisor on the same hardware [35].

The hypervisor's execution environment is mainly related to the host OS and hardware that the hypervisor is running on. It occurs when the hypervisor depends on running on top of an OS that inherits the vulnerabilities related to OS software and applications [35]. Figure 2.3 shows a classification of considered threats and attacks according to virtualization reference in the research.

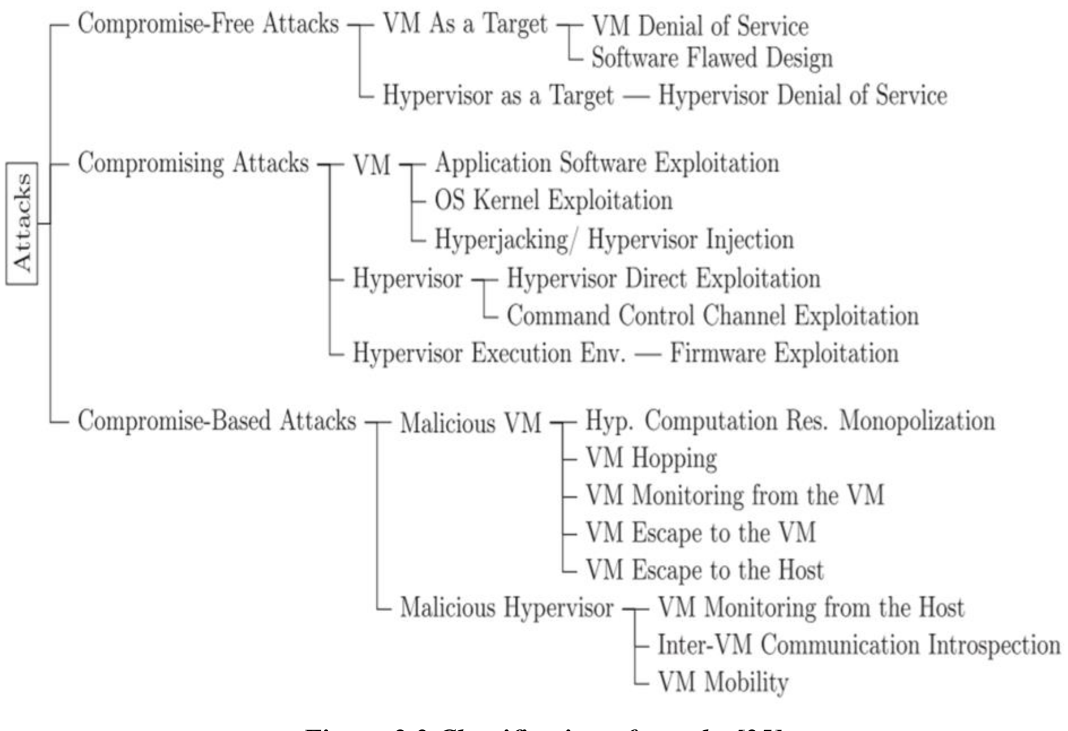

Attacks
Compromise-Free Attacks
VM As a Target
VM Denial of Service
Software Flawed Design
Hypervisor as a Target
Hypervisor Denial of Service
Compromising Attacks
VM
Application Software Exploitation
OS Kernel Exploitation
Hyperjacking/ Hypervisor Injection
Hypervisor
Hypervisor Direct Exploitation
Command Control Channel Exploitation
Hypervisor Execution Env.
Firmware Exploitation
Compromise-Based Attacks
Malicious VM
Hyp. Computation Res. Monopolization
VM Hopping
VM Monitoring from the VM
VM Escape to the VM
VM Escape to the Host
Malicious Hypervisor
VM Monitoring from the Host
Inter-VM Communication Introspection
VM Mobility

Figure 2.3 Classification of attacks [35]

Currently, there are various vulnerabilities and risks that can jeopardize the virtualization environment within a cloud infrastructure. An attacker can exploit or penetrate any of the vulnerabilities and access sensitive data and resources. [36] presented another classification of challenges related to virtualizations (Figure 2.4).

The research categorized the virtualization challenges into six primary classes and different sub-classes. The major classes are consist of challenges related to *virtualization characteristics* (in respect of virtualization characteristics technology in cloud infrastructures such as mobility, isolation, and scalability); infrastructure challenges (concerning any software and hardware components required by the virtual environment and its components); challenges on access and communication security (related to any types of access to virtual resources and their way of communication); data security challenges (in respect of how to securely share sensitive data in an insecure virtual environment); challenges related to controlling and monitoring (related to lack of visibility and monitoring on virtual functions within the system); security policies and rules issues (considered as a dynamic enforcement policies in a different layer of the virtual environment, integration of dynamic and static policies).

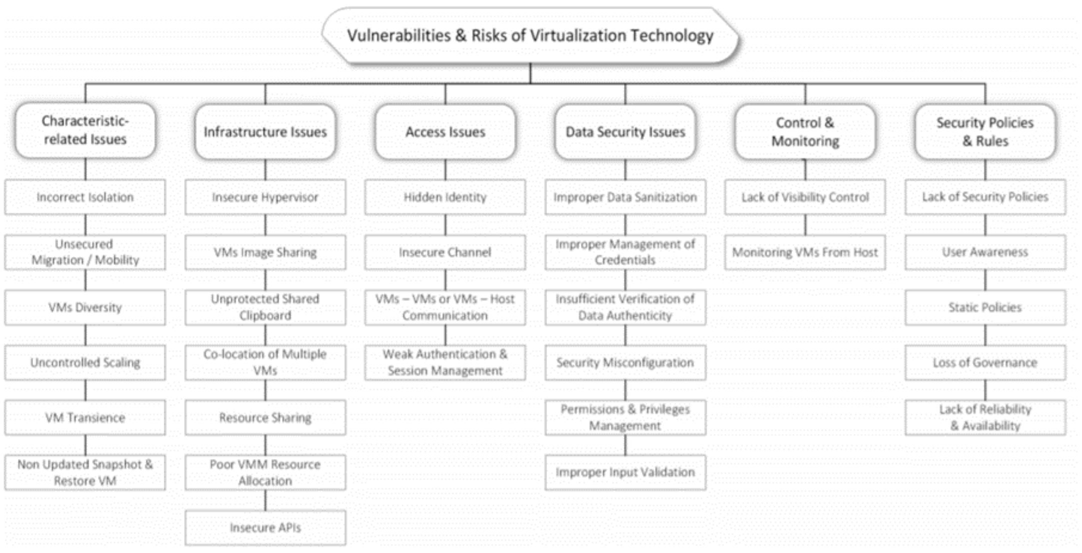

Vulnerabilities & Risks of Virtualization Technology
Characteristic-related Issues
Infrastructure Issues
Access Issues
Data Security Issues
Control & Monitoring
Security Policies & Rules
Incorrect Isolation
Unsecured Migration / Mobility
VMs Diversity
Uncontrolled Scaling
VM Transience
Non Updated Snapshot & Restore VM
Insecure Hypervisor
VMs Image Sharing
Unprotected Shared Clipboard
Co-location of Multiple VMs
Resource Sharing
Poor VMM Resource Allocation
Insecure APIs
Hidden Identity
Insecure Channel
VMs – VMs or VMs – Host Communication
Weak Authentication & Session Management
Improper Data Sanitization
Improper Management of Credentials
Insufficient Verification of Data Authenticity
Security Misconfiguration
Permissions & Privileges Management
Improper Input Validation
Lack of Visibility Control
Monitoring VMs From Host
Lack of Security Policies
User Awareness
Static Policies
Loss of Governance
Lack of Reliability & Availability

Figure 2.4 Security Vulnerabilities and Risk in a Virtual Environment [36]

❖ Solution and Guidance

Cloud Security Alliance (CSA Security Guidance V3.0) has produced guidance for critical areas of focus in cloud computing and has offered recommendations on the following issues:

Virtual machine guest hardening: Proper hardening and protection of a VM instance can be delivered via software to each guest.

Hypervisor security: The hypervisor needs to be locked and hardened using best practices. The primary concerns should be the proper management of configuration and operation and physical security of the server hosting the hypervisor.

Inter-VM attacks and blind spots: VMs may communicate with each other over a hardware backplane rather than a network. As a result, standard-network based security controls are blind to this traffic and cannot perform monitoring or in-line blocking. In-line virtual appliances help to solve this problem.

Migration of VMs: An attack scenario could be the migration of a malicious VM in a trusted zone, and with traditional network-based security control, its misbehavior will not be detected. Installing a full set of security tools on each machine is another approach to adding a layer of protection.

Performance concerns: Installing security software for physical servers onto a virtualized server can result in severe degradation in performance. Security software needs to be virtualization-aware.

Operational complexity from VM sprawl: The ease at which VM's can be provisioned has led to an increase in the number of requests for VM's in typical enterprises. This creates a larger attack surface and increases the odds of misconfiguration or operator error opening a security hole. Policy-based management and the use of a virtualization management framework are critical.

Instant-on gaps: A VM can be started and stopped with ease, and this creates a situation where threats can be introduced into the gap when a VM is turned off and when it is restarted, leaving the VM vulnerable. Best practices include network-based security and virtual patching that inspects known traffic attacks before they can get to a newly provisioned or newly started VM.

Virtual machine encryption: VMs are vulnerable to theft or modification when they are dormant or running. The solution to this problem is to encrypt VM images at all times, but there are performance concerns.

Data comingling: There is concern that different classes of data (or VM's hosting different classes of data) may be intermixed on the same physical machine. VLAN, firewalls, and IDS/IPS should be used to ensure VM isolation as a mechanism for supporting mixed model deployments. Data classification and policy-based management can also prevent this.

Virtual machine data destruction: When a VM is moved from one physical server to another, enterprises need the assurance that no bits are left behind on the disk that could be recovered by another user or when the disk is de-provisioned. Zeroing memory/storage encryption of all data is a solution to this problem. Encryption keys should be stored on a policy-based key server away from the virtual environment.

Virtual machine image tampering: Pre-configured virtual appliances and machine images may be misconfigured or may have been tampered with before you start them.

In-motion virtual machines: The unique ability to move VMs from one physical server to another creates complexity for audits and security monitoring. In many cases, VMs can be relocated to another physical server (regardless of geographical location) without creating an alert or trackable audit trail.

2.3 Cloud Computing

Cloud computing has become an alternative IT infrastructure where users, infrastructure providers, and service providers all share and deploy resources for their business processes and applications. Business customers are shifting their services and applications to cloud computing since they do not need to invest in their own costly IT infrastructure but can delegate and deploy their services effectively to cloud vendors and cloud service providers [37].

Cloud computing offers a low-cost effective solution to provision on-demand cloud resources using its capability of pooling and resource virtualization. Cloud clients can store their data, share their informations, and consume and run their services with a low-cost and fast access over a remote and accessible server rather than on physical resources with limited capacity [38].

The most relevant definition is probably the one provided by the National Institute of Standards and Technology (NIST) [39]: "*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 five essential characteristics, three service models, and four deployment models. The five characteristics are on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service. Software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS) constitute the three service models [40].

SaaS directly offers cloud services such as Google Docs, Google Map, and Google Health, online to users. With PaaS, developers can order a required development platform, which may consist of SDK (software development kit), documentation, and test environment, to develop their own applications. IaaS is more about packaging and provisioning underlying virtual resources to customers, who then build, orchestrate, provision, and sell tailored infrastructure resources to organizations to support their own businesses.

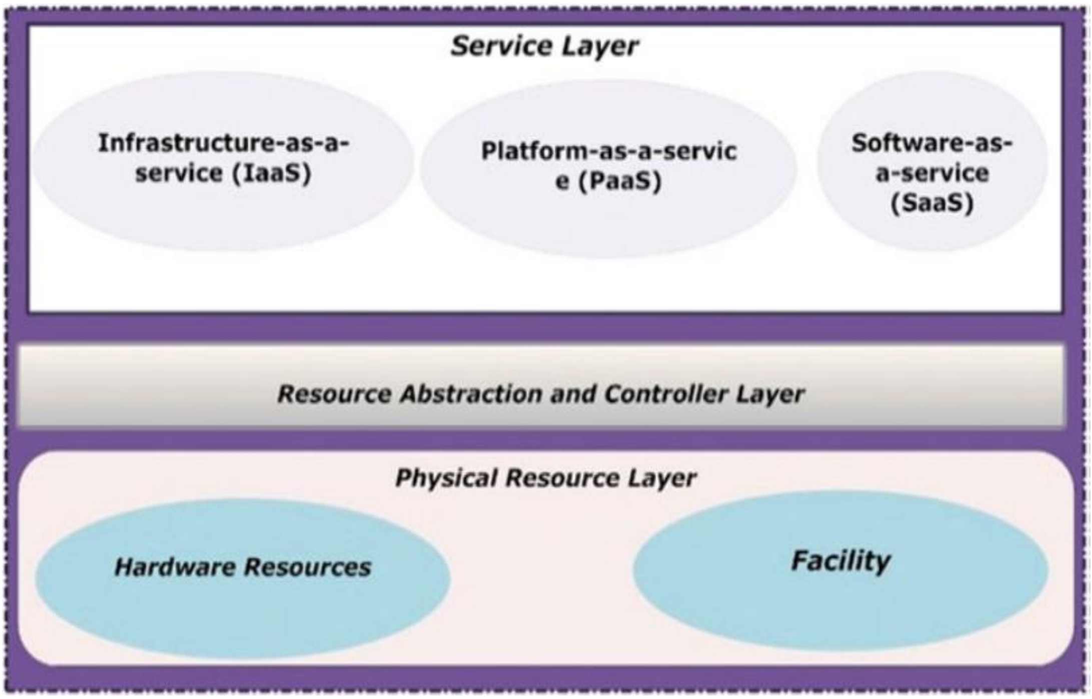

Service Layer
Infrastructure-as-a-service (IaaS)
Platform-as-a-servic e (PaaS)
Software-as-a-service (SaaS)
Resource Abstraction and Controller Layer
Physical Resource Layer
Hardware Resources
Facility

Figure 2.5 Cloud provider—three-layer service orchestration model

NIST provides a three-layer service orchestration model, as shown in Figure 2.5. The physical resource layer includes all the physical computing resources: computers (CPU and memory), networks (routers, firewalls, switches, network links, and interfaces), storage components (hard disks), and other physical computing infrastructure elements.

The resource abstraction and control layer contains the system components that cloud providers use to provide and manage access to the physical computing resources through software abstraction (virtualization layer). The resource abstraction components include software elements such as hypervisors, virtual machines, virtual data storage, and other computing resource abstractions. The control aspect of this layer refers to the software components responsible for resource allocation, access control, and usage monitoring. The service layer contains interfaces for cloud consumers to access the computing services.

In [41], the cloud was described according to two main ends consisting of the front end and back end. The cloud tenants and users can communicate with the cloud and access required and available services like OpenStack dashboard. The back end includes underlying physical and virtual resources that are responsible for delivering cloud services (refer to Figure 2.6).

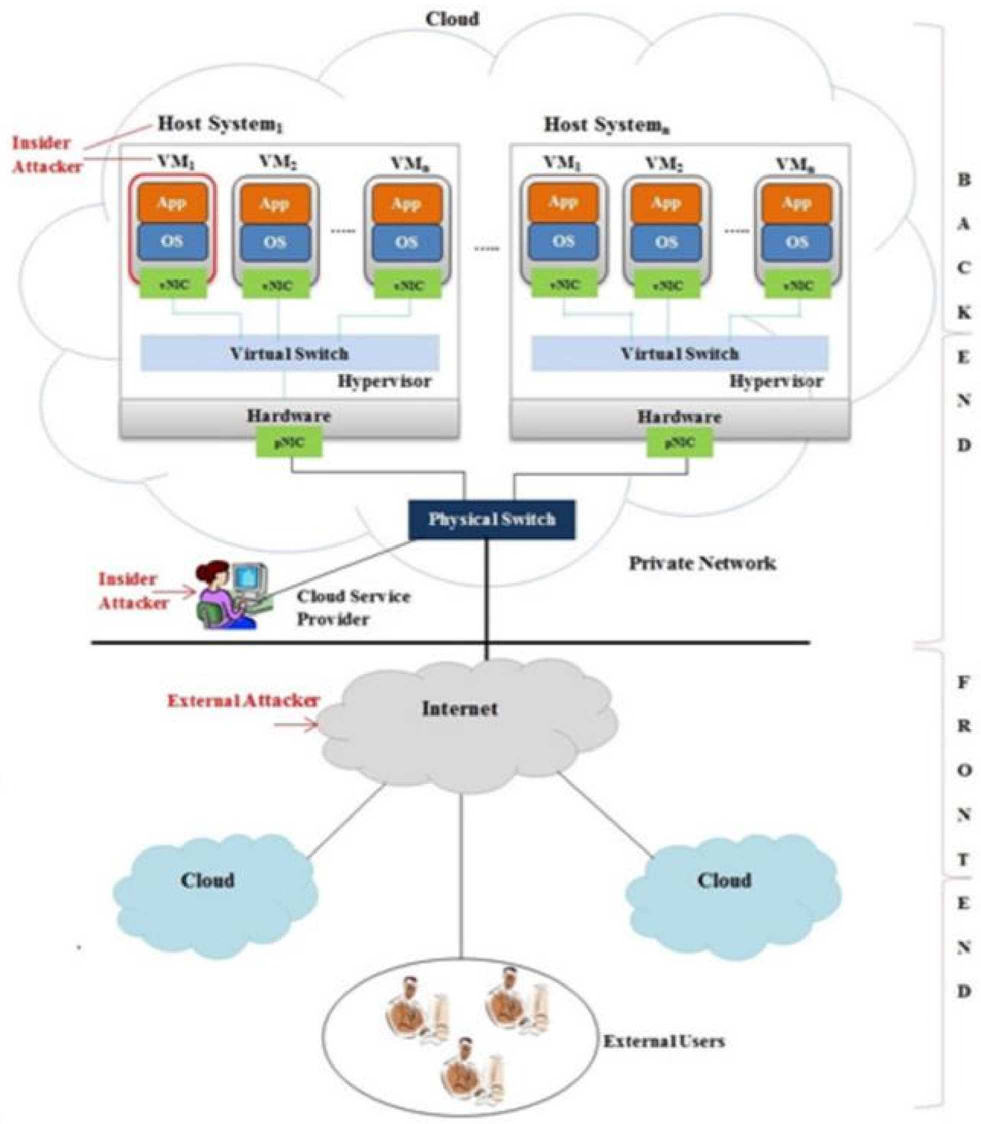


Figure 2.6 Cloud computing front and back end [41]

2.3.1 Cloud terminology – roles and boundaries

Cloud introduced various predefined roles and classifications to each organization migrating to cloud infrastructure. This section defines some of these roles and how they interact with each other within the cloud system.

Cloud Provider (CP) refers to an individual, association or third party that delivers cloud computing services via on-demand, pay-as-you-go systems as a service to businesses. The major responsibility is to deliver reliable and available cloud services to its tenants according to their signed Service Level Agreements (SLA). It provides a cloud-based platform, services, infrastructure, application, and storage.

Cloud Consumer/Tenant is an individual/organization who consumes services provided by cloud providers.

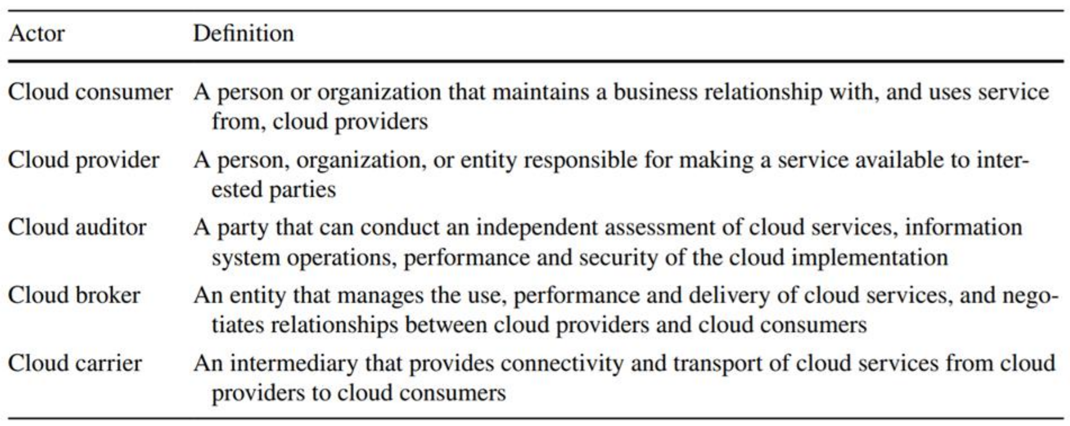
Cloud Resources Administrator is an individual or third party with higher privilege that performs administerial tasks for cloud resources. The cloud resource admin is considered a specific role with high accessibility to specific resources and can configure/change the resources.

Cloud Broker is generally acting as an interface between the cloud provider and cloud consumer. It is an application or an individual that manages the performance, usage, and delivery of cloud resources.

Cloud-Oriented Architecture (COA) is a conceptual model encircling all elements within the cloud infrastructure. It includes all entities and elements networked to form a cloud environment.

Cloud Object is an individual object that can be directly or indirectly identified via an identifier like ID number, location, name, and precisely defined characteristics within the cloud environment. A cloud object can be defined as a static or dynamic object according to its role. A cloud tenant has access to the static/dynamic objects shared by cloud providers.

Additionally, [42] defined different types of actors in cloud computing, displayed in Figure 2.7.



Actor	Definition
Cloud consumer	A person or organization that maintains a business relationship with, and uses service from, cloud providers
Cloud provider	A person, organization, or entity responsible for making a service available to interested parties
Cloud auditor	A party that can conduct an independent assessment of cloud services, information system operations, performance and security of the cloud implementation
Cloud broker	An entity that manages the use, performance and delivery of cloud services, and negotiates relationships between cloud providers and cloud consumers
Cloud carrier	An intermediary that provides connectivity and transport of cloud services from cloud providers to cloud consumers

Figure 2.7 different actors in cloud infrastructure [42]

2.3.2 Cloud Security

Since the cloud has become a large-scale and complex infrastructural environment, it becomes more vulnerable to traditional and new security threats related to its structure and elements. Cloud security is a critical part of computer security, which describes policies, technologies, control, and monitoring applications to protect virtualized data, and shared resources (services, applications, cloud objects). According to NIST, the major obstacle to adapting cloud environment and services in most organizations is security, portability, and interoperability. Cloud security covers numerous security issues and challenges. [43] considered cloud vulnerabilities and security requirements and highlighted security challenges in related areas like NFV, SDN, IoT, and cloud applications. It proposed appropriate countermeasures to mitigate security threats. However, the paper lacked security issues related to cloud data and resource isolation. [44] provided a survey on security challenges related to cloud infrastructure. The paper covered security countermeasures related to cloud nature and its virtualized, distributed, and resource sharing environments. It suggested an integrated security solution. [45] provided a security service that enables cloud tenants to protect and monitor their systems by deploying an intrusion detection system. Their method mainly focused on the tenant level. [46] published a taxonomy of intrusion detection and prevention literature. They analyzed various attacks related to service platforms and presented potential attacks and mitigation strategies against such attacks in the cloud. The main notable security issues refer to two major concepts: virtualization and multi-tenancy.

Virtualization is defined as the conceptual process of creating virtual version instances extracted from underlying physical resources and delivering them as software-based virtual components. Virtualization is pervasive in almost all components of the service infrastructures: virtual machines, virtual networks, virtual storage, virtual network functions, and virtual services. Cloud computing relies on virtualization technology to virtualize and aggregate physical resources into pools of virtual resources (virtual machines, virtual networks, virtual storage, virtual functions, and virtual services) and provision them to users on demand.

A hypervisor and virtual machines are components of a virtual environment. A VM is an image operating system (OS) that contains memory and storage. A hypervisor is responsible for constructing, managing, and controlling VMs within the system. The hypervisor virtualizes the hardware resources like CPU, memory, storage, and network and allocates them to each VM [47]. As discussed in section 2.2.1, various virtual environment vulnerabilities can be exploited by attackers and endanger the whole system. It is worth noting that the impact of security challenges in a virtual cloud environment is more critical than traditional infrastructures since resources are virtualized and shared among numerous users.

Since virtualization is a crucial technology in cloud infrastructure, any vulnerability can endanger the whole system in a high-security breach. For example, any error and vulnerability inside the hypervisor can allow an attacker to launch VMs attacks (shutting down VMs) or monitor other VMs and their shared resources. A compromised VM can inform an attacker of the underlying network operation to exploit existing network vulnerabilities. It can also enable an adversary to compromise the hypervisor and achieve control over the whole system. Local users and malicious codes can bypass security boundaries or even gain privileges to cause damage to the infrastructure and its users through vulnerabilities found in virtualization software.

Multi-tenancy is a specific cloud computing characteristic that allows sharing applications, services, resources (compute, network, storage) among tenants. Each virtual instance can be shared by one or more tenants. A hypervisor supports multi-tenancy by providing access to a pool of shared resources in cloud environment. Cloud multi-tenancy enables multiple users can access and use an application/resource in a same infrastructure. Three methods are recognized to achieve multi-tenancy in cloud which are physical separation, database, and using virtualization. The virtualization is a method to enable multi-tenancy especially in infrastructure as a service (IaaS). In a platform as a service (PaaS) provider, users can execute various applications/services in a multi-tenancy environment using virtual platform. It is worth noting that multi-tenancy can be exploited in co-tenancy, co-residency- and co-location attacks since the valued tenant's data might be placed in the same physical location or hardware. An adversary can access side VMs, perform illegal scripts, or even run unauthorized applications [44, 48].

Some of the traditional security issues found in the cloud infrastructure are data access control, loss and data leakage, trust, and isolation. Cloud-specific security issues include insecure interfaces and APIs, malicious insiders, account or service hijacking, virtualization security, and service interruption. In the following, we discuss these critical and significant security challenges that affect cloud security.

While there are many security concerns in cloud computing, Cloud Security Alliance (CSA) released twelve critical security threats specifically related to the shared, on-demand nature of cloud computing for cloud computing with the highest impact on enterprise business [49]:

Data Breaches A data breach is an incident in which sensitive, protected, or confidential information is released, viewed, stolen, or used by an individual who is not authorized to do so.

Weak Identity, Credential, and Access Management Data breaches and enabling attacks can occur because of a lack of scalable identity access management systems, failure to use multifactor authentication, weak password use, and a lack of continuous automated rotation of cryptographic keys, passwords, and certificates.

Insecure APIs (Application Programming Interface) Provisioning, management, orchestration, and monitoring are all performed using a set of software user interfaces (UIs) or application programming interfaces. These interfaces must be designed with adequate controls to protect against accidental and malicious attempts to circumvent policy. Cloud providers deliver services to their customers through software interfaces mostly integrated with the web application layer. The stability of cloud components is dependent upon the security level of these APIs within the cloud infrastructure. Insecure cloud APIs can cause various threats related to confidentiality, availability, integrity, and accountability.

These API functions and web applications share a number of vulnerabilities, resulting in high-level security problems. Consequences of any malfunction in APIs may allow malicious codes to be imported inside the cloud and expose user confidential data. Although robust authentication methods, proper access controls, and encryption methods may solve some of the above problems, still, there are serious gaps especially related to the inability of massive auditing and logs. Any APIs that will interact with sensitive data within cloud infrastructure must be protected with a secure channel such as SSL/TLS.

System and Application Vulnerabilities System vulnerabilities are exploitable bugs in programs that attackers can use to infiltrate a computer system to steal data, take control of the system, or disrupt service operations.

Account Hijacking This is a significant threat, and cloud users must be aware of and guard against all methods such as phishing, fraud, and exploitation of software vulnerabilities to steal credentials. It is a kind of identity theft that aims to deceive end-users to obtain their sensitive data. If an attacker gains control of a user account, it can snoop on all customer's activities, manipulate and steal their data, or redirect the customer into inappropriate sites. These kinds of threats can be accomplished through phishing email, faux pop-up windows, spoofed emails, buffer overflow attacks, which result in the loss of control of the user's account.

Malicious Insiders A malicious insider threat to an organization is a current or former employee, contractor, or another business partner who has authorized access to an organization's network, system, or data and intentionally misuses that access in a manner that negatively affects the CIAAA of the organization's information system. According to the Cloud Security Alliance (CSA) cloud security threat list, this type of threat is one of the most serious cloud-specific security challenges. It happens when an employee of cloud service providers (CSPs) abuses his/her level of access to gain confidential information of cloud customers for any nefarious purposes. The worst case is when a malicious system administrator can access client resources hosted on virtual machines and data stores. So, detecting such indirect accesses to client data is one of the challenging tasks in cloud infrastructure.

Advanced Persistent Threats (APTs) These are a parasitical form of cyber-attack that infiltrates systems to establish a foothold in the computing infrastructure of target companies from which they smuggle data and intellectual property.

Data Loss Data stored in the cloud can be lost for reasons other than malicious attacks. Accidental deletion by the cloud service provider or a physical catastrophe such as a fire or earthquake can lead to permanent customer data loss.

Insufficient Due Diligence An organization that rushes to adopt cloud technologies and chooses cloud service providers (CSPs) without performing due diligence exposes itself to a myriad of commercial, financial, technical, legal, and compliance risks.

Abuse and Nefarious Use of Cloud Services Poorly secured cloud service deployments, free cloud service trials, and fraudulent account sign-ups via payment instrument fraud expose cloud computing models such as IaaS, PaaS, and SaaS to malicious attacks. Malicious actors may leverage cloud computing resources to target users, organizations, or other cloud providers.

Denial of Service (DoS) Denial-of-service attacks are attacks meant to prevent users of a service from accessing their data or their applications by forcing the targeted cloud service to consume inordinate amounts of finite system resources so that the service cannot respond to legitimate users. *Service interruption* caused by DoS/DDoS attacks are usually attempted against Internet services with a large population of users, and it is more so against the cloud as a center of a high number of cloud services and users. These attacks may render services and computing resources unavailable. A DDoS attack may occur when an attacker gains access to a tenant's VMs credentials due to their vulnerabilities.

Shared Technology Issues Cloud service providers deliver their services by sharing infrastructure, platforms, or applications. The infrastructure supporting cloud services deployment may not have been designed to offer strong isolation properties for a multi-tenant architecture (IaaS), re-deployable platforms (PaaS), or multi-customer applications (SaaS). This can lead to shared technology vulnerabilities that can potentially be exploited in all delivery models.

2.4 Software-Defined Networking

Referring to research conducted by [50], the Software-Defined Networking (SDN) term was initially made to express the OpenFlow protocol idea and functionality at Standford University, CA, USA. The new technology emerged as a networking paradigm that separates the data forwarding plane from the control plane. It centralizes the network state and the decision-making capability in the control plane (SDN controller), leaving simple forwarding operation at the data plane (SDN network devices) and abstracting the application plane underlying network infrastructure. The separation of the control plane and the data forwarding plane is through a programming interface between the SDN network devices and the SDN controller [51].

The Open Networking Foundation (ONF) defines a high-level architecture for SDN [52], with three main layers as shown in Figure 2.8: the application layer for expressing and orchestrating application and network service requirements; the control layer for network control, services provisioning, and management; and the infrastructure layer for the abstraction of physical network resources. The infrastructure layer can be expanded into two planes: the physical plane and the virtual plane. The physical resources plane consists of the underlying physical infrastructure, and the virtual resources plane which represents the virtual resources abstracted from the physical resources through virtualization [53, 54].

SDN network devices are all placed at the infrastructure layer. According to instructions programmed by their SDN controller, the SDN network devices make a simple decision of what to do with incoming traffic (frames or packets). The SDN controller (or group of controllers) is located in the control layer. The technology programs and controls the forwarding behavior of the network devices and presents an abstraction of the underlying network infrastructure to the SDN applications. Applications and network services are on the application layer. The controller allows applications to define traffic flows and paths, with the support of a comprehensive information database of all underlying network infrastructure operations, in terms of common characteristics of packets to satisfy the needs of the applications and to respond to dynamic requirements by users and traffic/network conditions [53].

The SDN controller uses interfaces for communicating with other layers. To communicate with the data/infrastructure layer, a southbound interface (SBI) is used for programming and configuring network devices. A northbound interface (NBI) is provided for the interaction between the SDN controller and applications to communicate with the application layer. The NBI is to describe the application needs and pass along the commands to orchestrate the network. East/west interfaces are for information exchange between multiple or federated controllers. The OpenFlow protocol has been developed and widely adopted as one of the SBIs between SDN controllers and SDN switches. OpenFlow uses a secure channel for message transmission over the Transport Layer Security (TLS) connection.

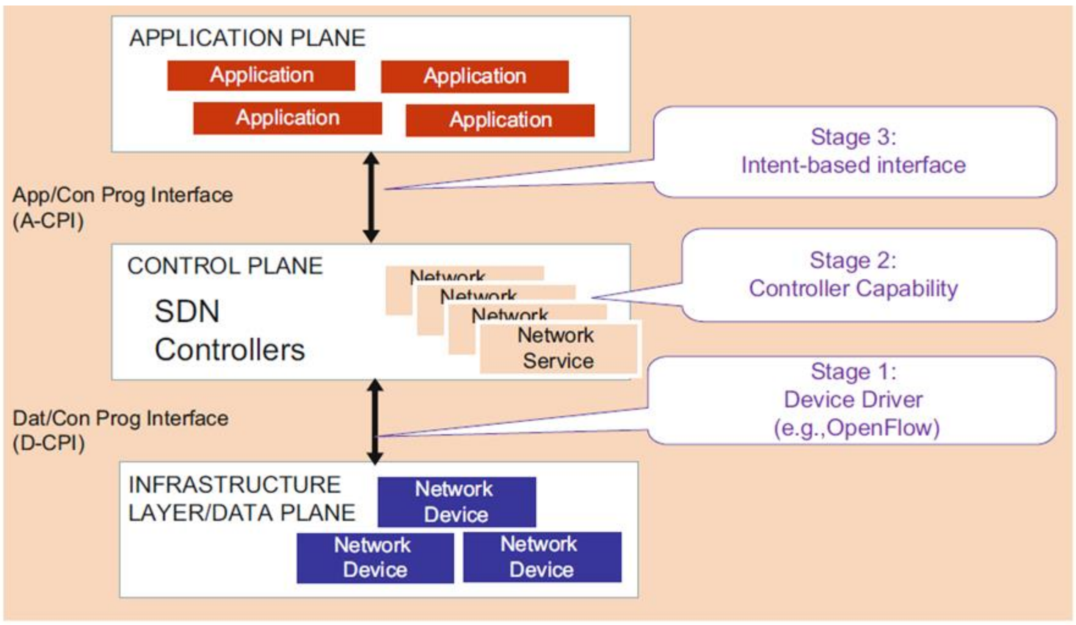

APPLICATION PLANE
Application
Application
Application
Application
App/Con Prog Interface (A-CPI)
Stage 3: Intent-based interface
CONTROL PLANE
SDN Controllers
Network
Network
Network
Network Service
Stage 2: Controller Capability
Stage 1: Device Driver (e.g.,OpenFlow)
Dat/Con Prog Interface (D-CPI)
INFRASTRUCTURE LAYER/DATA PLANE
Network Device
Network Device
Network Device

Figure 2.8 SDN Components

A centralized SDN controller dictates the network policies for underlying network fuctions[55]. There are varieties of SDN controller platforms such as FloodLight [56] , OpenDayLight [57], ONOS [52], POX [58], and Ryu [59].

2.4.1 SDN Interfaces

The crucial components of SDN are communicating through Application Programming interfaces (APIs). The APIs are considered as architectural components of SDN that push configuration, rules, and information to underlying forwarding function [60]. The SDN network is surrounded by three main APIs: Southbound API, Northbound API, and East and westbound API.

❖ **Southbound API**

The Southbound API acts as a connecting bridge between the SDN controller and underlying forwarding functions, which plays a crucial role in separating functionality between control and data plane. The required configurations and information between controller and forwarding elements travel through this API [61]. This interface's main objective is to transfer the SDN controller decisions/rules to underlying SDN devices and provide information related to SDN devices back to the SDN controller. The OpenFlow is one of the most widely accepted and deployed southbound standards for SDN. However, it is not the only one, there are other southbound API like ForCES [62], Open vSwitch Database (OVSDB) [63], POF [64], OpFlex [65], and OpenState [66].

IETF proposed forwarding and Control Element Separation (ForCES). Like OpenFlow, it decouples the control plane from the data plane but can still be kept in the same network function. The OVSDB is considered to offer advanced management capability for OVSs. The OVSDB is a complementary protocol to OpenFlow and can allow the SDN control functions to create multiple virtual switches.

- **OpenFlow**

OpenFlow is a standardized and well-known southbound protocol that defines the communication between SDN controller and OpenFlow switches. The messages are transmitted over a secure channel implemented via a Transport Layer Security (TLS) connection over TCP. The SDN controller defines and programs the underlying OpenFlow switches packet forwarding behavior using the exchanged messages. According to SDN controller rules, OpenFlow switches perform packet forwarding and report back its configuration status and traffic conditions to the SDN controller over exchanged messages [53]. The main functionality of OpenFlow switches is to support the SDN controller and forward packets through the SDN network [67]. The SDN switches are mainly responsible for forwarding and handling traffic according to rules set by SDN controller rules, gathering network states, and transferring them to the SDN controller through the southbound interface [68-70]. The SDN switches are required to understand the OpenFlow header for forwarding the packets. The OpenFlow protocol has advanced through different versions: version 1.0 consisting 12 fixed matching fields and one single flow to version 1.5, including 41 matching fields and quite a few new functionalities [60, 71].

The OpenFlow devices must include three essential components: one/multiple flow tables, secure channel, and OpenFlow protocol. The OpenFlow is recognized as a flow-oriented protocol that contains switches and port abstractions to handle and control the SDN network's traffic flows [72-74]. Each OpenFlow device may consist of one or more flow table, a protocol for communication between devices and the external controller, and a secure channel that connect them to the SDN controller. The table/s in OpenFlow switches consist of flow entries in a match, actions, and statistics format. Each packet should match with one of the matching fields presented by the OpenFlow protocol. According to [53, 75], "*a flow is a set of packets transferred from one endpoint to another endpoint*." So, a flow table in each OpenFlow device consists of various flow entries. Figure 2.9 shows an OpenFlow essential operation, its flow table, and entries.

The transferred messages between SDN controller and OpenFlow switches are categorized into three types: controller-to-switch messages, asynchronous messages, and symmetric messages. The controller-to-switch messages are referred to types of messages that manage and program the OpenFlow switches. The asynchronous messages are used to notify the security controller of any changes on the state of OpenFlow switches. The last type of message is considered as a hello message for both SDN controller and devices to ascertaining the liveness of the connection [53]

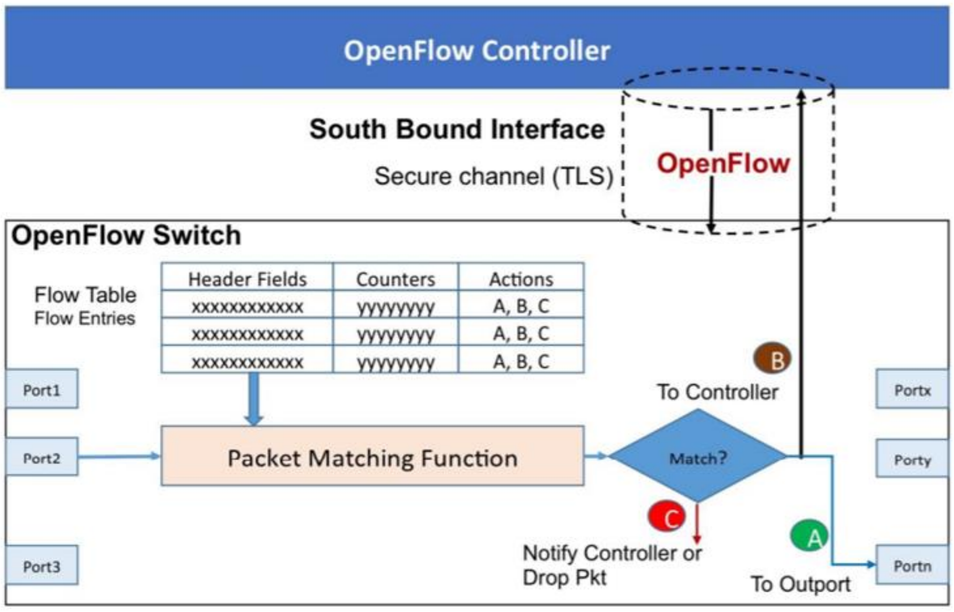


Figure 2.9 OpenFlow Switch- operation [53]

❖ **Northbound API**

The northbound interface is still an open issue compared to the southbound interface and does not follow a common standard like OpenFlow. The northbound interface is a software ecosystem and plays a critical role for application developers. It connects the control plane to the application plane and provides information on underlying SDN devices to application developers. Regarding northbound interfaces, each existing SDN controller such as OpenDaylight, Floodlight, and NOX proposed and defined its interface according to their specific definitions [76] [77] [16]. SDN controller has chosen different programming languages to provide an abstraction of SDN controller functions and underlying forwarding behavior from the application developers.

❖ **East and westbound API**

Centralized control over the large scale of networks is the main feature presented by SDN, but the number of switches controlled by a single SDN controller is limited. The east/westbound interfaces are required in the case of distributed SDN controllers. The interface requires the functional ability to import/export data between distributed SDN controllers. Currently, each SDN controller introduces its version of the east/westbound interface. There are various proposal of that proposed interfaces between SDN controllers such as Onix data import/export functions [78], ForCES CE–CE interface [62], [79], and distributed data stores [80].

2.4.2 SDN Security Challenges

SDN introduces a new networking paradigm, and its impact is in the form of a new framework, new components, structural layers, and interfaces. SDN brings with it new security challenges beyond those that existed in traditional networks.

As SDN decouples the control plane from the data plane, the technology brings with it new sets of components, interfaces, as well as many new security issues. Security challenges in SDN can be divided based on its three layers: the data plane, the control plane, and the application plane. The data plane can suffer from various security threats such as malicious OpenFlow switches, flow rule discovery, flooding attacks (e.g., switch flow table flooding), forged or faked traffic flows, credential management, and insider malicious host. The application plane inherits security challenges such as unauthorized or unauthenticated applications, fraudulent role insertion, lack of authentication methods, and lack of secure provisioning.

The control plane faces several security issues related to the centralized SDN controller, communication interfaces, policy enforcement, flow rule modification for modifying packets, controller-switch communication flood, system-level SDN security challenges (related to lack of auditing accountability mechanisms), and lack of trust between the SDN controller and third-party applications [81]. Since the control plane in the SDN architecture acts as the heart of this virtual network infrastructure, security vulnerabilities on this layer can cause failure to the entire virtual network architecture.

According to a study published by [82], various SDN security challenges have been analyzed and classified in relation to SDN structure. Security challenges associated with the SDN framework by the affected layer/interface are categorized as follows:

- Application Layer Unauthorized access is through the unauthenticated application. Malicious applications may introduce fraudulent rule insertion. Configuration issues arise from a lack of policy enforcement.
- Control Layer Unauthorized access can be introduced through unauthorized controller access and unauthenticated application. Data modification is presented in the form of flow rule modification to modify packets. Malicious applications can introduce fraudulent rule insertion and controller hijacking. Denial of service (DoS) may occur due to controller-switch communication flood. Configuration issues may arise because of the lack of TLS (or other authentication techniques) adoption or lack of policy enforcement.

- Data Layer Unauthorized access may occur with unauthorized controller access. Data leakage may result from flow rule discovery (side-channel attack on input buffer) or forwarding policy discovery (packet processing timing analysis). Data modification is a result of flow rule modifications. Malicious applications may introduce controller hijacking. Denial of service may occur due to controller switch communication flood or switch flow table flooding. Configuration issues may arise from lack of TLS (or other authentication techniques) adoption.
- Application-Control Interface (NBI—Northbound Interface) Unauthorized access may occur because of unauthenticated applications. The malicious application may introduce fraudulent rule insertion. Configuration issues may occur due to lack of policy enforcement.
- Control-Data Interface (SBI—Southbound Interface) Unauthorized access can be introduced through unauthorized controller access. Data modification is presented in the form of flow rule modifications. Malicious applications can introduce controller hijacking. Denial of service may occur due to controller switch communication flood. Configuration issues may arise from lack of TLS (or other authentication techniques) adoption.

Besides, since SDN uses virtualization technology to virtualize networks (VNs), it inherits traditional security problems related to the virtualization of virtual machines and new security issues related to the virtualization of network hypervisors and their isolation. It also suffers threats such as Dos/DDoS attacks, with higher impact because of SDN control centralized architecture. In another classification, the SDN introduces critical security challenges, including unauthorized access (control plane, data plane, application plane), routing policy collision, fraudulent flow rules insertion or tampering in switching level, insecure interfaces, and system-level SDN security challenges.

SDN Controller. Since SDN decouples the data plane from the control plane, it is the responsibility of the centralized controller to deal with all incoming network flows. As a consequence, the controller itself is a key bottleneck and is the target for various attacks such as flooding and DDoS attacks. An SDN controller can be implemented in a virtual or physical server with associated resources. An attacker can launch a kind of resource consumption attack on the controller to render it unavailable in response to flow rules coming from underlying switches and force it to respond extremely slowly to packet-in events or sending packet-out messages. A DoS/DDoS attack is one of the most serious security threats against the SDN controller when an attacker endlessly sends IP packets with different headers to the controller to put it in a nonresponsive state.

Routing policy collision. Policy collision is another specific security challenge in SDN architecture when various vendors and third-party applications use different configurations and programming models. This is critical since a malicious component can delete, insert, or modify existing and predefined policies of flows inside the SDN controller. Separate servers or applications with different policy rules may result in policy conflicts with each other.

Fraudulent flow rules insertion or tampering in switching level. A compromised or malicious application can generate fraudulent flow rules while communicating with the controller. An attacker can inject fake flow rules through the switches by exploiting vulnerabilities of southbound interfaces. It is possible for an attacker to tamper with network information by modifying flows in flow tables. These malicious flow rules can cause a network to behave abnormally. For instance, [83] introduced an attack in which an attacker generates forged link layer discovery protocol (LLDP) packets through an OpenFlow network to create vulnerabilities on internal links between two switches. An adversary can also insert malicious flow rules by monitoring the traffic from OpenFlow Switches.

Insecure interfaces. Another critical security challenge in SDN infrastructure is related to insecure Application Programming Interfaces (APIs): Northbound, Southbound, and East and West Interfaces. This security issue is critical since all communications between the SDN controller, the application layer, the underlying forwarding layer, or even the communication between multiple controllers go through these interfaces. For instance, vulnerabilities and the lack of standard protocol in the northbound interface may enable attackers to interfere with both the application and the controller's operation and send a malicious request through the controller or network elements or even generate flooding attacks with the purpose of disrupting its operation. An adversary is also capable of sending a large number of requests through the northbound interface to occupy the interface bandwidth. In a multi-domain multi-controller environment, the controller’s communication goes through the East/West APIs. These SDN controllers may be from different vendors and do not have a common secure channel between them. Messages among them may be sniffed by an attacker through vulnerabilities of East-West APIs, and sensitive information may be exposed.

System-level SDN security challenges. A specific SDN system-level security concerns auditing processes. As it is essential to keep comprehensive state information of network devices in the infrastructure to prevent unauthorized access, providing an auditing and accountability mechanism in SDN is a critical security challenge [84].

2.5 Network Function Virtualization

Network functions virtualization (NFV) is proposed aiming to virtualize an entire class of network component functions using virtualization technologies. The objective is to decouple the network functions from the network equipment. The NFV is proposed to pave the way for a new way to provision network services in comparison to current existing practices. According to [85], the NFV introduces three main differences: decoupling software from hardware as network functions are defined as software-based virtual network functions; flexibility on deploying network functions where enables faster creation of virtual network functions and their placement over any NFV-enables devices; dynamic scaling which allows flexibility in deploying VNFs over larger-scale infrastructure.

ETSI provides an NFV reference architecture for a virtualized infrastructure and points of reference to interconnect the different components of architecture. The NFV architecture has three critical components for building a practical network service: network functions virtualization infrastructure (NFVI), VNFs, and NFV management and orchestration (MANO) [85]. Figure 2.10 shows an overall view of NFV architecture adapted from the ETSI NFV model.

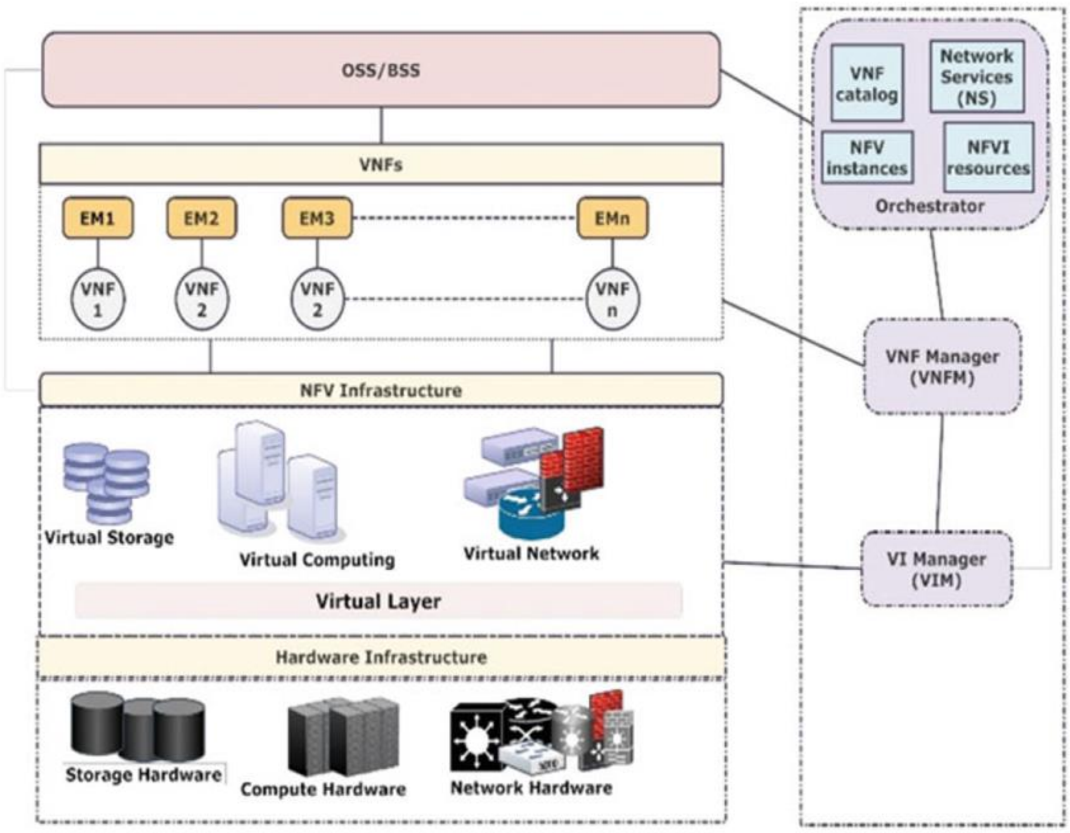


Figure 2.10 NFV architecture

- NFV Infrastructure (NFVI)

The NFVI includes hardware and a hypervisor that virtualizes and abstracts the underlying resources. The NFVI encompasses all underlying physical and software resources which are used to host the VNF. The physical resources consist of computing, storage, and network functions that provide required storage, compute, and connectivity for each VNF through a virtualization layer. The virtualization layer includes a hypervisor that enables the abstraction of physical resources for initiated VNFs.

- NFV-MANO

The NFV MANO is responsible for configuring, deploying, and managing the life cycle of VNFs. The NFV-MANO includes different functional blocks like Network Functions Virtualization Orchestrator (NFVO) which provides both resource and service orchestration; Virtualized Network Function Manager (VNFM) which manages the life cycle of single/multiple VNFs; Virtualized Infrastructure Manager (VIM) which provides management and control over NFVI hardware and software resources. An NFV-MANO architectural framework is shown in Figure 2.11 according to the ETSI description [86, 87].

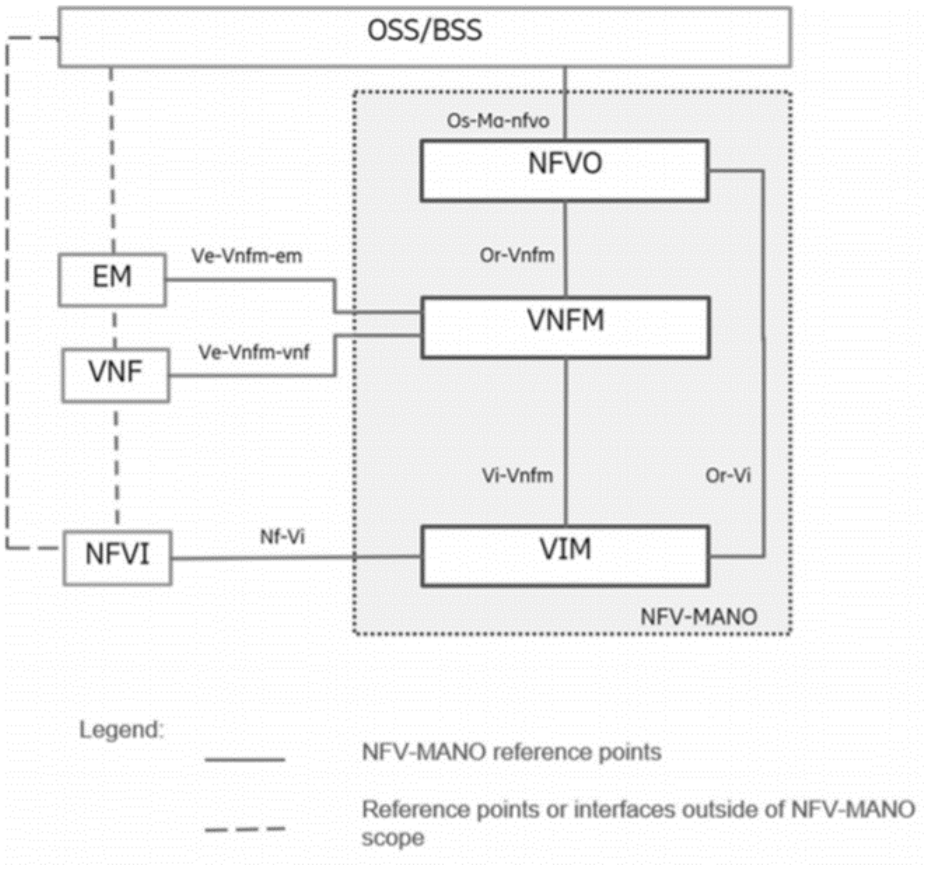


Figure 2.11 NFV-MANO architectural framework [86]

The main focus of NFV-MANO is its involvement in most virtualized-specific management tasks that are essential within the NFV architecture. It defines interfaces to provide communication between its different components. The NFV-MANO also coordinates with a traditional network management system, Operations Support System (OSS) and Business Support Systems (BSS), to enable management of both VNFs as well as functions running on legacy devices [17].

- VNF

A network function is now a virtual instance of a customized software program called a virtual network function (VNF). The VNF is the software implementation of a network function that runs over the NFVI. This object can be created on-demand, launched into operation wherever needed, without installing new equipment (on any virtual or physical servers at data centers, gateways, routers). It can be moved at will and terminated when its function is no longer needed [88].

The NFV enables network functions to be executed as software instances in a virtual machine (VM) on single or multiple hosts instead of customized hardware equipment. Network functions virtualization can be applied to both data and control planes in fixed or mobile infrastructures. The NFV allows operators to combine numerous network equipment types into high-volume switches, servers, and storage inside data centers, network nodes, and end-user premises. It offers a new means for creating, deploying, and managing networking services.

Examples of these classes of functions include switching elements; tunnel gateway elements: IPSec/SSL (secure sockets layer), VPN (a virtual private network) gateways; security functions: firewalls, virus scanner, and intrusion detection systems; traffic analysis services: load balancers, network monitoring, and deep packet inspection tools; service assurance: SLA (service-level agreement) monitoring, test, and diagnostics; mobile network elements: multifunction home router, set-top boxes, base stations, and the evolved packet core (EPC) network [89].

An essential key principle of NFV is service chaining: as each VNF provides limited functionality on its own, service chaining allows combining multiple VNFs to create useful new network functions and services.

2.5.1 NFV Security Challenges

In this section, we present security challenges related to NFV architecture. As network components are virtualized, NFV networks contain a level of abstraction that does not appear in traditional networks. Securing this complex and dynamic environment, that encompasses the virtual/physical resources, the controls/protocols, and the boundaries between the virtual and physical networks, is challenging for many reasons according to CSA[90]:

Hypervisor dependencies Hypervisors are available from many vendors. They must address security vulnerabilities in their software. Understanding the underlying architecture, deploying appropriate types of encryption, and applying patching diligently are all critical for the security of the hypervisors.

Elastic network boundaries In NFV, the network fabric accommodates multiple functions. Physical and virtual boundaries are blurred or non-existent in NFV architecture, making it challenging to design security systems.

Dynamic workloads The NFV is about agility and dynamic capabilities, but traditional security models are static and unable to evolve as network topology changes in response to demand.

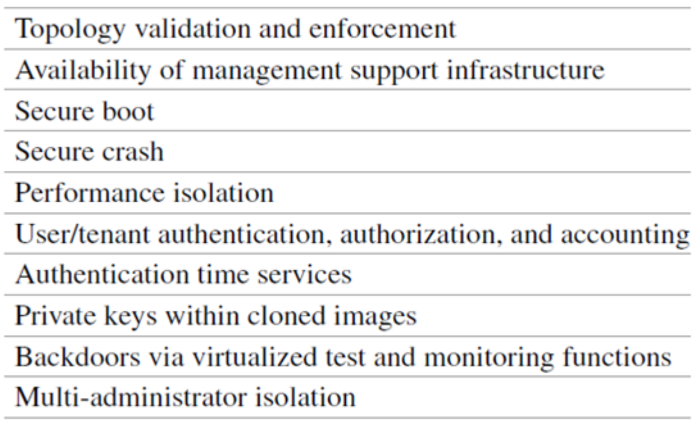
Service insertion NFV promises elastic, transparent networks since the fabric intelligently routes packets that meet configurable criteria. Traditional security controls are deployed logically and physically in-line. With NFV, there is often no simple insertion point for security services that are not already layered into the hypervisor.

Stateful versus stateless inspection Security operations during the last decade have been based on the premise that stateful inspection is more advanced and superior to stateless access controls. NFV may add complexity where security controls cannot deal with the asymmetry flows created by multiple, redundant network paths and devices.

Scalability of available resources Deeper inspection technologies- next generation firewalls and Transport Layer Security decryption, for example- are resource-intensive and do not always scale without offload capability.

The ETSI Security Expert Group focuses on the security of the software architecture. It identifies potential security vulnerabilities of NFV and establishes whether they are new problems or just existing problems in different guises [91]. The identified new security concerns resulting from NFV are as shown in Table 2.1.

Table 2.1 Security Vulnerabilities of NFV



Topology validation and enforcement
Availability of management support infrastructure
Secure boot
Secure crash
Performance isolation
User/tenant authentication, authorization, and accounting
Authentication time services
Private keys within cloned images
Backdoors via virtualized test and monitoring functions
Multi-administrator isolation

The ETSI confirmed that NFV and its components certainly create new security concerns as presented in table 2.1. However, they have only provided security guidance specifically for NFV development and its architecture [91], but this does not consist of any specific implementation details

Another study revealed,[92], the NFV architecture can be prone to various security risks as VNF mostly runs over virtual resources. The study listed several potential security risks related to NFVI according to different attack scenarios such as isolation failure risk, mainly focused on improper isolation methods related to VNF and hypervisors; network topology validation and implementation failure, which introduced new attacks in relation to dynamicity and automation of network implementations within NFV framework and improper network implementations between VNFs; regular compliance failure; this focuses on violation of security policies and laws related to VNF especially in terms of migration and location changes; denial of service protection failure, which introduces the possibility of resource exhaustion caused by compromised VNFs; malicious insider which points to internal security risks that arise from NFV framework and its components, and security logs troubleshooting failure, which highlights the security risks related to a considerable number of records on hypervisor due to a compromised VNF which causes hypervisor failure.

2.6 Policy

Considering today's technologies, security systems are required for more advanced mechanisms to protect resources against complicated daily threats. Complex connectivity of resources within the virtual infrastructure is very challenging to be managed efficiently against security violations. The policy is defined as rules that govern the system behaving. Policies can be driven from service level agreements (SLA), business goals, and enterprises relationships. However, refining these high-level policies into policies involving a specific service and then into understandable low-level policies to be implemented by specific devices to support the service is not an easy task to behold [93].

Enforcing security policies is more complicated and critical in the current state of systems and organizations caused by a massive number of constructed virtual network functions and their complicated structure. The difficulty rises as each security component follows its own policy and enforcement mechanism, where security policies have different types and are mostly hard to extract[94]. Despite the fact that massive benefits are introduced by cloud technologies in various studies, there are still critical concerns regarding information policies such as security, privacy and access control rules in dynamic and large-scale cloud environments.

In any system, an event is considered a security breach either when it violates a defined security policy or violates the Confidentiality, Integrity, and Availability of security principles that could have been avoided if a relevant security policy has been in place. According to Dave [95], a policy (or policy rule) is a simple declarative statement linking a policy object with a value and a policy rule. In general, a policy is not easy to work with, as at one extreme it applies to the overall behavior of a complex organization (or entity). At the other extreme, it applies to a particular action on an element of the organization or specific firewall rules on a network connection. [96] stressed that managing access control rules are quite a difficult task to be handled and requires organizational security policies to be unfolded to attain access control rules packages. According to the study, subjects acquire rules and permissions according to their role in each level of their organizations.

[97] defined the policy as a "*set of rules that contains conditions or criteria with proper validate action were defined the system's behavior if the represented condition is satisfying or successful*." The conditions depend on the nature and characteristics of the system resources. The actions are defined as tasks that require to be enforced or controlled by the system. In other words, each policy definite how the system resources should be properly accessed or used if their conditions are fulfilled. Policy rules are associated with a value to an object. According to [95], "*policy rules are defined either based on conditions or actions*." Each policy might consist of one or more conditions or actions.

Policy Conditions A policy condition is defined if the policy rule is applicable for any object within the system. The object can be consisting of any items like user, organization, a user in an organization, application, a network/subnet, time,

Policy Action A policy action describes what are the validated behavior/actions of an object like a network, device, and other similar entities in any system.

Policies in nature could be classified into two fundamental categories: complex or simple. Simple policies include a set of clear conditions and validated actions (Figure 2.12). Complex policies are constructed from simple policies but include more sophisticated interaction between objects. A nested policy can be considered a combination of various simple policies that create more complicated policies containing nested conditions and actions. Policies can be defined as a group of policies assigned to one/multiple entities/subjects. According to [98], a group policy consists of five types of policies *authentication, authorization, filtering, channel protection, and operational*. The authorization policies are defining rules to authenticate a subject in a network element. The mechanisms to be considered for authentication can be varied, such as biometrics, primary authentication method, Kerberos, certificates, physical credentials, and shared secrets. The authorization policies are referred to policies that specify privileges for each authenticated subject. The privileges are considered activities that grant/deny an action. The filtering policies are referred to as policies that create rules to define each network element filtering criteria. The channel protection policies are determined to provide channel protection requirements according to associated security such as IPSec, SSL/TLS, and SA. The operational policies described the behavior of a network in the face of any triggered event.

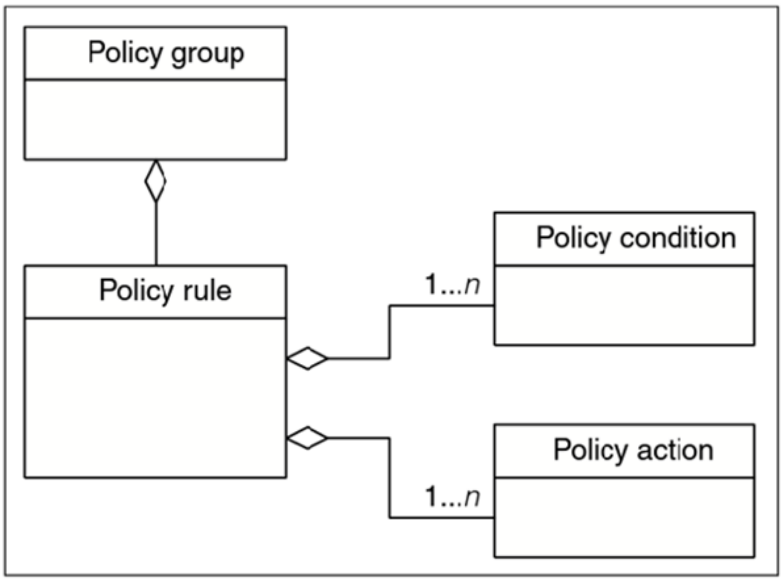


Figure 2.12 Simple Policy [95]

Policies can be divided into two main categories: management policies (using policies for managing resources) and security policies (using policies for security purposes like controlling traffic). [95] classified policies into five main policy types based on their intent:

Motivational Policies These types of policies focus on whether and how a policy goal is accomplished. The two specific related policies in this category are known as Configuration and Usage policies.

Installation Policies These policies represent specific administrative permissions as well as dependencies among different components. In other words, they define what can and cannot be put on a component and configuration of mechanisms on component installation.

Error and Event Policies These policies define what action should be taken in case of a component failure or malfunction.

Security Policies These are essential policies that deal with the security of the components and the whole system. Security policies define the desired behavior of the heterogenous application, systems, networks, and any type of object within the system. They mainly deal with accessibility, authorization, authentication, accountability, and auditing rules to system resources/components. They determine the validated actions to be performed on any resources within the system.

Service Policies These types of policies described available services in the network. Service policies characterize the network and other services within the system.

A security policy mostly includes two parts, the targeted traffic, and the matching action. Security devices determine the targeted traffic using a existing fields of a packet header like IP address, port, and protocol. The action can be considered any possible action within the system like a drop, or a forward. Each security mechanism enforces the policies according to its method and criteria [99].

2.6.1 Security Policy Mechanisms

In this section, we explore various security policy mechanisms.

The principal aim of security mechanisms is to analyze and enforce the policies. The existing mechanisms can be divided into two parts based on their operations: statically or dynamically. Static mechanisms enable security devices to analyze the program source code or binary code before the execution step to catch the security violations before allowing the program to be run. These methods are not efficient for large-scale virtual environments such as the cloud with many VNFs and services. On the other hand, dynamic mechanisms monitor the programs and services during their running time and interven as necessary. [100] enforce the security policies according to program/application code. The study presents a taxonomy of security policies based on the granularity of the code.

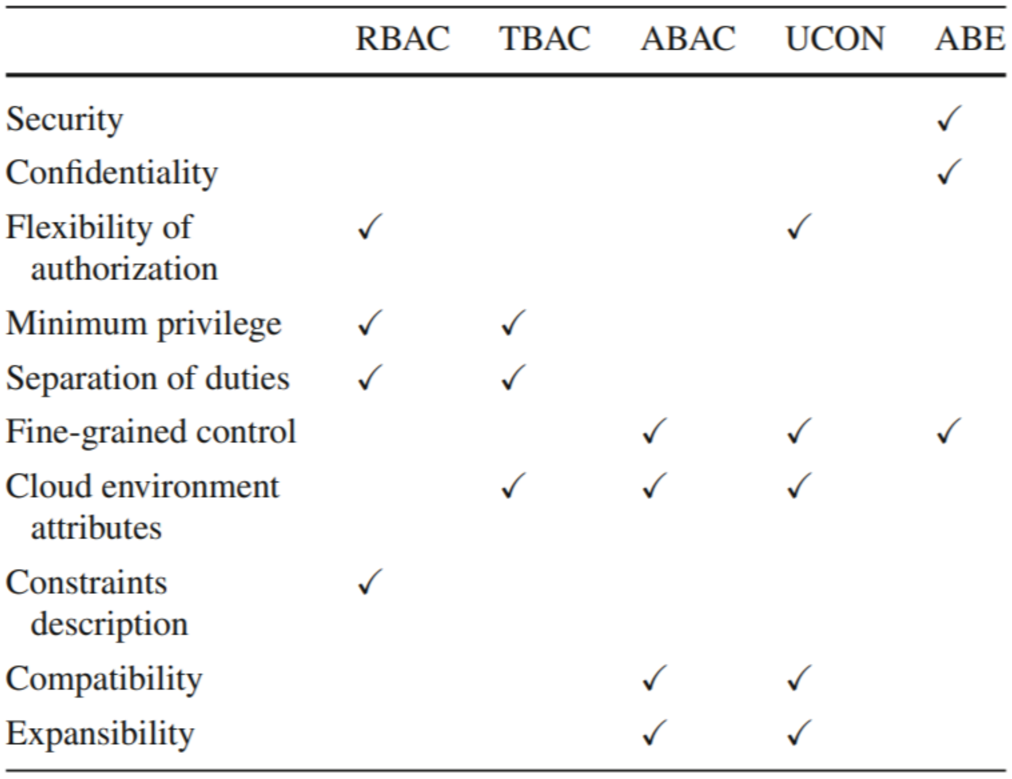
The multi-tenancy allows resources to be shared among multiple users and services. However, the challenge is on how to enforce the policies in such an environment. New technologies like virtualization, cloud, SDN, and NFV have revealed new challenges to enforce security policies. [101] demonstrated a security architecture based on virtualization and Trusted Computing technologies. The model addressed customer isolation in which isolation policies specified by customer policies that are automatically enforced. The model solution was presented according to Trusted Virtual Domain (TVD). [102] introduced CloudFlow as cloud-wide policy enforcement to deploy the policies on the cloud. The model enforced the information flow policies. The model is initially designed for an openstack cloud environment. Another study presented by [103] proposed a middleware to enforce security policies in the OpenStack cloud environment. It is a pluggable module within the OpenStack nova service. The model decides on user requests according to given security policies. The model is limited to the OpenStack environment. [104] proposed a policy-based security architecture to secure SDN domains. They defined different modules within their application to determine security policies related to packets and match for conflict discovery. It only detected policy conflicts concerning a request. The mechanism did not include prediction on security policy violations. A monitoring approach was presented by [105] as a security solution to detect malicious or compromised services according to their agent monitoring with the capability of enforcing policies. The policy enforcement mechanism supports service interaction authorization policies. The model used interaction authorization policies to enforce one/more policies. The model is limited to end-to-end service monitoring in a cloud-based environment. [106] introduced a policy space analysis and focused on addressing issues related to network security policy enforcement on middleboxes.

Using software-defined networks, [107] proposed a policy-based security architecture to secure inter and intra domain communication between different hosts across multiple domains with their Policy-based Security Application. A policy language was presented which specifies attributes associated with entities and flows in SDN. [108] introduced an approach for automatic enforcement of security policies in network function virtualization according to dynamic network changes. It deployed virtual security functions (VSFs) for security policy reinforcement and introduced a security awareness manager in the orchestrator. A cyberspace-oriented access control model (CoAC) was proposed to provide access to sensitive data [109]. The method considered operations a combination of many atomic processes and defined a CoAC policy that permits access only if a particular operation's security risk is below a specified threshold.

2.6.2 Access Control Policy Enforcement Methods

The major responsibility of access control policies is to restrict access between a subject (initiator who wants to access a resource) and an object (reactor which is the resource to be accessed by subject). Access controls are described as a significant part of security analysis on cloud computing. The majority of security mechanisms studies have focused their research on proposing various access control policy enforcement methods such as [110], [111], [112], [113], [114], [115], [116].

In this section, we describe access control mechanisms. [117] reviewed existing access models and policies along different application scenarios focusing on cloud and user requirements. The major components of an access model are described as subject, object, and access control policy. The research reviewed four types of access control: task-based access control, action-based access control, attribute-based access control, and usage-based access control. They described a comparison between different types of access controls shown in Figure 2.13.



	RBAC	TBAC	ABAC	UCON	ABE
Security					✓
Confidentiality					✓
Flexibility of authorization	✓			✓	
Minimum privilege	✓	✓			
Separation of duties	✓	✓			
Fine-grained control			✓	✓	✓
Cloud environment attributes		✓	✓	✓	
Constraints description	✓				
Compatibility			✓	✓	
Expansibility			✓	✓	

Figure 2.13 A comparison between different types of access control models[117]

The authors in [118] proposed a geographical Role-Based Access Control. It relied on role-based mechanisms and defined constraints according to user location and position. It relied on role-based mechanisms and described conditions according to user location and position. The method mostly focuses on determining an authorization control function according to users' role schema and position.

A cloud access control security model (CCACSM) was proposed considering different policy levels: authority level, action level, and behavior level. The model is categorized under the action-based access control model. The CCACSM architecture described the relation among the main component of an access control model (Figure 2.14) [119]. [120] proposed POSTER for enhancing administrative role-based access control. It has integrated obligations via an executive model by defining three main obligatory actions. However, the model only focused on administrative actions within the system. Although the model reduced potential security administrative risk, it mainly concentrated on a decentralized system rather than centralized mechanisms. [121]addressed the access control difficulties related to objects and linked object states.

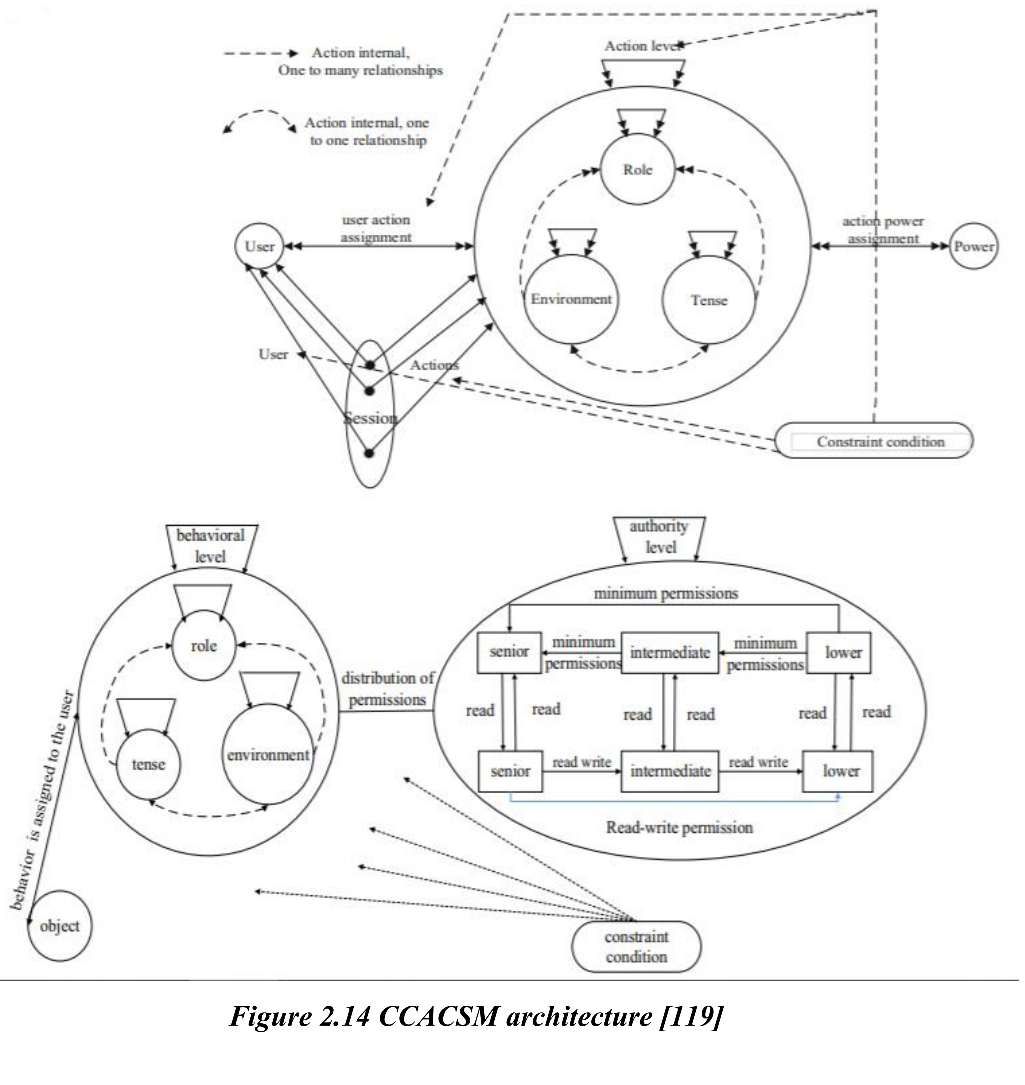


Figure 2.14 CCACSM architecture [119]

2.7 Security by Isolation

Isolation is the most critical part of every shared and multitasking computing system, which provides resilience against different forms of violation/attacks and is an essential measure in our design. Isolation is a technique for separating or partitioning different concerns that can be used for both resource management and security purposes. For example, process isolation in the time-sharing operating system is realized with virtual address space, and network isolation in the early network operating system is realized with a firewall. In network management, system management, and service management, isolation is used to identify, detect, and isolate faults, misconfiguration, and performance issues. Security isolation has been a critical approach to system and network security. The systems community has adopted virtualization as the technique of choice for providing isolation. [8] identifies some challenges related to virtualization, mainly focusing on Side-channel attacks (SCAs) vulnerabilities causing isolation violation.

The responsibility of the infrastructure service provider (ISP) is to provide a secure infrastructure that ensures tenant's virtual machines are isolated in a multitenancy environment, and the various networks within the infrastructure are isolated from one another. Virtual networks can be one or many networks over which virtual machine traffic flows. Isolation of virtual machines within this network can be enhanced using virtual firewall solutions that set firewall rules at the virtual network controller. Although virtual machines are often marketed as the ultimate security isolation tool, it has been shown that many existing hypervisors contain vulnerabilities that can be exploited.

In a multi-tenant environment, traffic isolation, address space isolation, performance isolation, and control isolation are often required for different purposes. Traffic isolation prevents any data packets from leaking between tenants. Address space isolation allows the tenants to isolate their network by choosing their end-host IP and media access control (MACs) addresses independently from each other.

Control isolation enables the tenants to control and configure their network without affecting other tenants [122].

The design of classical security devices cannot protect the components of virtualized environments since traditional security depends on physical network devices. These devices cannot see the significant security activities inside virtualized environments [123]. Isolation will become an essential technique for monitoring virtual security boundaries.

2.7.1 Isolation Classification

In this section, we classify different types of isolations and their potential usage:

Tenant Isolation In a cloud configuration, tenants share the same underlying physical infrastructure. Without network isolation, tenants could intentionally or unintentionally consume a large part of the network, intrusively see data on the network that does not belong to them or invoke breaches such as unauthorized connection monitoring, unmonitored application login attempts, malware propagation, and various man-in-the-middle attacks.

Domain Isolation In order to label packets and enforce the isolation policies, it is necessary to determine the domain for each data flow. Each domain is associated with a set of input ports of the edge switches. Since the architecture distinguishes intra-tenant, inter-tenant, and external communications, the controller needs to check to which IP range the destination IP address belongs. There is a separate database table for mapping public IP addresses to the tenants who have been allocated such addresses.

Data Isolation Customers in fields such as banking or medical records management often have extreme data isolation requirements and may not even consider an application that does not supply each tenant with its own individual database. VM Isolation A hypervisor divides the host hardware resources among multiple VMs. It coordinates all accesses by VMs to the underlying hardware resources and thus provides the necessary isolation between the virtual machines. In other words, VMs can share the physical resources of a single computer and remain completely isolated from each other as if they were in separated physical machines [124].

VM Isolation A hypervisor divides the host hardware resources among multiple VMs. It coordinates all accesses by VMs to the underlying hardware resources and thus provides the necessary isolation between the virtual machines. In other words, VMs can share the physical resources of a single computer and remain completely isolated from each other as if they were in separated physical machines [124].

Traffic Isolation in Hypervisor-Based Environments Network traffic isolation is through the creation of segmented networks. In physical network isolation, network interface cards will be dedicated to a specific application or group of applications, and thus physical segmentation is provided between networks. In logical/virtual network isolation, software such as VLAN or network interface virtualization is used. Each interface is assigned a unique IP and MAC address; thus, each is logically distinct. The VLAN tagging can be defined in the host server to isolate network traffic further. Traffic for multiple applications share the same physical interfaces, but each application sees only the network traffic and resources assigned to it and cannot see traffic or resources assigned to other applications.

Traffic Isolation in Zones-Based Environments Similar to hypervisor-based virtualization, when a zone is provisioned, one or more network interfaces are presented, and the IP stack is enabled. The IP and MAC addresses are configured on the logical interface. Routing policies and network security can be hardened in these zones when the zones are provisioned.

Network Isolation Any isolated virtual network can be made up of workloads distributed anywhere in the data center. Workloads in the same virtual network can reside on the same or separate hypervisors. Additionally, workloads in several multiple isolated virtual networks can reside on the same hypervisor. Virtual networks are also isolated from the underlying physical infrastructure. Because traffic between hypervisors is encapsulated, physical network devices operate in an entirely different address space than the workloads connected to the virtual networks.

Network Segmentation Network isolation is between discrete entities. Network segmentation applies to homogeneous entities, e.g., protection within a group. Traditionally, network segmentation is a function of a physical firewall or router designed to allow or deny traffic between network segments or tiers. For example, segmenting traffic between a web tier, application tier, and database tier. In a virtual network, network services that are provisioned with a workload are programmatically created and distributed to the hypervisor vSwitch. Network services, including L3 segmentation and firewalling, are enforced at the virtual interface.

2.7.2 Standard Network Security Solutions by Isolation

With compliance and regulatory requirements, network isolation and network security have become essential elements of any service infrastructure deployment. The technology used for network traffic isolation does not always cover issues with security breaches that stem from external networks, side-channel attacks, or regulatory concerns between tenants. Network security is built on top of network isolated traffic. Standard security solutions include:

Network Firewalls Firewalls are often situated at the edges of networks to filter potential security threats coming from untrusted sources. Network firewalls may be hardware devices, software such as soft switches, or a combination of both.

LAN Tagging Tagging allows multiple logically separated networks (VLANs) to use the same physical medium. Thus, two separate VLANs cannot communicate with each other. VLAN configurations are performed at the switch and define the mapping between VLANs and ports. Packets sent by a virtual network interface on a VLAN cannot be seen by virtual interfaces on other VLANs, and broadcast and multicast packets sent from a virtual network interface on a VLAN will be distributed only to the network interfaces on the same VLAN.

Role-Based Security On the client-side, the user devices must have hardened user authentication. On the database server side, role-based security, or role-based access control (RBAC), needs to be employed.

2.7.3 Cloud resource isolation mechanisms

In this section, we explore several existing isolation mechanisms in cloud computing. Researches focus on a various level to provide isolation within the system.

Security isolation analysis in the cloud is comparatively a domain with less work. [125] proposed a multi-tenant isolation solution using VMs as the boundary of security whereby applications run within containers on top of these virtual machines. To improve the security of running applications as a container in the cloud, it was suggested to run one container per VM. However, the drawback of such a system is recognized as its efficiency in performance. Silverline was proposed by [126] for enhancing data and network isolation for cloud tenant's services. The model concentrated on providing isolation via OS-level and virtual instances. It used labeling and information-flow tracking services. The method only focused on providing data and network isolation at the tenant level.

A mechanism known as SLIM (Secure Logical Isolation for Multi-tenancy) was introduced in [127] as an end-to-end approach to providing isolation among tenant's resources in a multi-tenant cloud storage environment. SLIM consisted of five privilege processes: security gateway, gatekeeper, guard and proxy, tenant authenticator, and request. The model only considered tenant-level isolation and two types of attacks: within and across the tenant. [128] proposed a method for strong tenant separation in cloud platforms by isolating components at the network level. It focused mainly on tenant separation via physical and cryptographic separation for large infrastructures. [129] conducted a research survey on network isolation solutions for multi-tenant data centers for isolating cloud services. It emphasized the main challenges of isolation in a multitenant environment and pointed out appropriate existing isolation solutions. [130]proposed a Highly Scalable Isolation Architecture for Virtualized Layer-2 Data Center Networks (SVDC). It used SDN technology to provide isolation for a layer-2 data center at the network level. The method separated global identifiers for virtual networks and enhances MAC-in-MAC encapsulation. The SVDC only provided isolation at the network level by decoupling identifiers while using the server-local identifier to differentiate virtual networks within a physical server.

2.8 Open-sources for Deploying a Cloud Security SDN/NFV platform

CloudSimSDN-NFV: It is a new simulation framework consisting of NFV, cloud, and SDN features. It is an extended version of CloudSim, a novel framework for modeling cloud computing. The framework supports NFV functionalities and is based on mapping the architecture and components of ETSI NFV Management and Orchestration (MANO), including NFV Orchestrator (NFVO), VNF Manager (VNFM), and Virtual Infrastructure Manager (VIM) [29].

OpenStack: OpenStack is an IaaS cloud platform based on shared storage, compute, and network resources. OpenStack is a collection of open-source technology projects with various functional components. OpenStack is an example of an integrated software-defined infrastructure involving ETSI NFV architecture framework, SDN network infrastructure, and cloud IaaS. It provides an automated infrastructure for cloud users. The OpenStack uses SDN technology to generate automated network infrastructure, NFV to create VNFs, and cloud to orchestrate and manage services. It is an IaaS cloud solution based on the integration of numerous benefits that interact through a set of OpenStack APIs, which is available to all cloud users.

OpenFlow: The protocol is considered as the first SDN communication standard protocol. The OpenFlow defines the southbound communication between SDN controller and OpenFlow Switch. It allows remote programmability and management of network devices through a network controller. The protocol provides a dynamic configuration of network functions via its controller.

Mininet: It enables constructing a realistic virtual-based network, switches, and application code on a single virtual machine. It can connect to various types of commercial SDN controllers like OpenDaylight, FloodLight, Pox, and so on. It provides an experimental testbed environment which consists of OpenFlow and SDN system.

OpenFlow Switches: It is an OpenFlow-based data switch that communicates over an OpenFlow channel to an SDN controller. It contains one or more flow tables and a group table that perform packet forwarding and lookup. It enables massive network automation and supports standard management, interfaces, and protocols.

2.9 Summary

This chapter provided an overview of software-defined infrastructure and its core technologies, including virtualization, cloud computing, software-defined networking, and network function virtualization. Moreover, this chapter discussed security challenges related to each technology as well as security isolation challenges. We provided a brief overview of various related topics utilized in software-defined security architecture, including policies and their current approaches. A brief introduction to open-source technologies used for developing and deploying the SDS_2 platform was given.

Chapter 3

Software-Defined Security Service Model

3.1 Introduction

Over the last decade, cloud computing has established itself as an effective technology for sharing and provisioning resources among tenants in a pay-as-you-go service fashion. The concept of everything-as-a-service was developed to utilize virtualization technology that allows underlying physical resources to be virtualized into virtual resources and services.

In parallel to cloud computing, the software-defined networking (SDN) paradigm has enabled the automation of virtual networks and network management with centralized control. Network function virtualization (NFV) pushes the concept even further by allowing virtualization (software implementation) of network functions, traditionally realized by hardware, and deploying them on computing devices.

Naturally, the concept of Cloud SDN/NFV integrated platform has been realized to take advantage of resource pooling and virtualization of cloud computing, programmability and automation of SDN, and network function virtualization performance, programmable, and dynamic systems and services. Along with these advances in resource and service virtualization, security issues have been explored as well. Security controls have to be developed to safeguard these platforms.

Many attempts have been made to address integrated resource-infrastructure platform security, providing security mechanisms and virtual security functions to counter numerous emerging security threats [43]. However, the challenge remains due to the scale and complexity of the virtual resource infrastructure and the difficulties in developing a matching security architecture that provides security and isolation of resources in a multi-tenant environment as well as provisioning dynamic security functions for security services on demand.

This thesis addresses those issues by proposing a logically centralized Software-Defined Security Service (SDS_2) Model for provisioning on-demand security functions capable of protecting cloud resources with the help of Software-defined Networking and Network Function Virtualization technologies.

The remainder of this chapter is organized into four sections. Section 3.2 justifies the proposed SDS_2 model. Section 3.3 presents SDS_2 security model. Section 3.4 describes the application of SDS_2 in a data centre. Section 3.5 discusses the features of the SDS_2 model. Section 3.6 provides a roadmap of this dissertation. Section 3.7 summarizes this chapter.

3.2 Why Programmable and automated Security Services on Demand?

Cloud computing has evolved into a key structure for IT industries for providing users on-demand services. Cloud architecture enables users to access cloud services over the Internet at any time regardless of their location through application software like web browsers. Cloud computing resources such as virtual servers, virtual storage, virtual networks, and virtual services are made available using virtualization technologies. In the current world, cloud computing has enabled many emerging technologies, resources, and services crucial to the current state of our lives in various areas, from industrial infrastructures and services to personal healthcare.

However, a broad movement to cloud infrastructure is limited by security challenges questioning cloud services reliability regarding protecting organizations' sensitive data and resources over intelligent security threats, especially over an integrated Cloud/SDN/NFV infrastructure. We have identified a number of significant issues of currently integrated cloud security platforms:

The vast numbers of virtual functions and their connectivity-service infrastructure. As anticipated, with billions of virtual functions within the infrastructure and their connectivity, the challenge here is how to manage the security and complexity of these functions and their connectivity over the broad area of cloud infrastructure while harnessing their main capability to protect cloud resources. The vast number of virtual connections among virtual functions makes it even more difficult.

The enormous number of virtual functions and their provisioning framework. The virtual functions are capable of interacting with others functions and performing their designated functions. However, the challenge lies in automating secure orchestration and provisioning these functions in critical areas as needed dynamically.

The massive number of virtual resources and their security isolation mechanism. As projected, with billions of virtual resources within the cloud infrastructure, the challenge is to provide an effective mechanism to provide inclusive visibility on undefined/invisible boundaries caused by the virtual functionality of resources within the cloud.

The vast number of virtual resources and their massive vector attacks. The augmented number of virtual functions and the complexity of their interaction within the virtual environment opens up massive security threats. The challenge is to provide proactive security mechanisms that dynamically predict security threats regardless of the number of virtual resources and functions within the infrastructure.

To address the challenging issues, we investigate various security models, isolation technologies, security function capabilities, protocols, and programmable security mechanisms for efficiently protecting the cloud and dynamic orchestration of virtual security functions.

On security orchestration and programmability of on-demand virtual functions. We explore security models and mechanisms for security service orchestration. We investigate a software-defined security architecture with a centralized security controller with overall visibility on the security functionality of underlying virtual functions to orchestrate a massive number of virtual security functions.

On automated secure connectivity and secure network architecture. The Software-Defined Networking (SDN) plays a critical role in SDI architecture by creating programmable virtual connectivity between components. Software-Defined Networking is developed as a technology to remove the current black box network infrastructure restrictions. SDN decouples the decision-making (control) plane from the data forwarding plane for adequate data transportation and fine-grained control of network management and services.

The SDN controller can configure networking devices automatically to deal with dynamic networks [131]. The SDN programmability network in security is still not so common and is still developing. We aim to adopt SDN to deploy our security network within our security architecture for efficient programmability of network security between our virtual security components.

On virtual security function. Network Function Virtualization (NFV) offers a new method for creating, deploying, and managing networking services by separating network/security functions from underlying hardware equipment. The technology introduces Virtual Network Functions (VNFs) as software-based virtual functions that can be created on-demand and launched into the system wherever required without installing new equipment [132]. NFV is a new emerging technology and did not completely develop within the infrastructure due to its architecture limitations and incomplete standards and protocols. However, we aimed to adopt the concept of VNFs to create specific programmable and dynamic Virtual Security Functions within our security architecture according to its designated functionalities.

On communication, control, and management protocol. Virtual security functions are not network routing devices, so heavy protocols for program network flows in network functions are not applicable to configure and manage VSFs in dynamic and programmable security infrastructure. No specific efforts have been made to address this issue within cloud security architectures. We decided to investigate the deployment of a new and simple protocol for transferring specific security parameters among components of our security model.

On visibility of security boundaries and construction of dynamic security boundaries. Security issues in a virtual cloud environment are more complex and challenging than those in traditional infrastructures since resources are both virtualized and shared among numerous users. As a result, virtual boundaries among components/participants are not well defined and often undefined, and hence they are not visible/controllable by the providers.

Isolation implies creating security boundaries for protecting cloud assets at different levels of a cloud security architecture. The main challenge is finding effective mechanisms for constructing dynamic security boundaries in cloud infrastructure. We investigate various security isolation mechanisms in different levels to provide an operative, intelligent and innovative technique to create security boundaries for protecting cloud resources.

On proactive security violation mechanism. A dynamic virtual environment of cloud infrastructure with a massive number of interactions among its virtual resources cannot rely only on traditional security mechanisms with limited capabilities to prevent and predict real-time security violations. We investigated mechanisms and algorithms to enable dynamic, intelligent, and effective proactive mechanisms to protect cloud resources regardless of their complexity.

This thesis addresses these significant challenges by introducing a logically centralized Software-Defined Security Service (SDS_2) Model. It provides a distributed architecture for orchestrating, managing a specific Interaction-based Virtual Security function to monitor cloud entities interactions. It proposed an innovative and intelligent Policy-based Interaction Model to manage, detect, and predict security violations in cloud infrastructure.

This chapter discusses the significance of the proposed Software-Defined Security Service (SDS_2) Model for provisioning on-demand virtual security functions and dynamic construction of security boundaries within a cloud infrastructure. We represent an overview of the proposed SDS_2 Model and its indispensable components. A roadmap of this research is sketched.

3.3 Software-Defined Security Service (SDS_2) Model

The ever-increasing number and gravity of cyberattacks against cloud assets, together with the introduction of new technologies, have brought many serious cloud security issues. Security issues in a virtual cloud environment are more complex and challenging than those in traditional infrastructures since resources are both virtualized and shared among numerous users. Traditional security mechanisms are not able to deal with virtualized environments.

The design of classical security devices cannot entirely protect the components of virtualized environments since traditional security depends on physical network devices. These devices cannot see the significant security activities inside virtualized environments [123]. To combat security attacks where attackers use software to exploit infrastructure vulnerabilities and virtualized agents to attack from anywhere and on multiple fronts instantaneously, we need to deploy the same tools and technologies of the attackers.

The concept of Software-Defined Security (SDSec) is a new approach in designing, deploying, and managing security by separating the forwarding and processing plane from the security control plane, similar to how SDN abstracts the forwarding plane from the control and management plane. Such separation provides a distributed security solution, which scales as VMs by virtualizing the security functions, and provides the ability to manage it as a logical, single system [133].

We propose the SDS_2 model as an SDSec Service that uses virtual cloud resources and can be deployed by the cloud provider to protect its integrated infrastructure. SDS_2 model exploits six main concepts: logical centralization of security control, virtualization of secure connectivity, security functions virtualization, and orchestration of virtual resources, dynamic construction of security boundaries, proactive technique encountering security violations according to cloud resource interactions.

This chapter describes our proposed Software-Defined Security Service model. The model integrates Software-Defined Networking (SDN) and Network Function Virtualization (NFV) techniques to enable dynamic programmability of virtual security functions within the introduced SDSec system. The system entails a novel model of a Software-based Virtual Security Function (VSF), a streamlined SDS_2 Security Controller (SC), an innovative Interaction model and practical Policy-based Interaction Model, and a novel and efficient protocol (Sec-Manage) between SC and VSFs for both management and communication.

The proposed model permits the programmability of heterogeneous virtual security functions for provisioning on-demand security services and their efficient management. A prototype is implemented with configurable virtual security functions representing specific interaction-based security functions, operating Sec-Manage. The implementation results demonstrate the feasibility and efficiency of the proposed model.

In the context of, we consider two main concepts: i) an underlying security device refers to specific virtual interaction monitoring known as VSFs in relation to the interaction model and its parameters; ii) an underlying virtual resource refers to virtual cloud resources such as Network, APP, Users, VMs, and Storage.

The SDS_2 deploys the very virtual resources of the cloud to provide its protection service. It behaves like a trusted tenant overseeing and providing the security service for the cloud infrastructure. SDS_2 belongs to the new software-defined approach that manages security by separating the security forwarding and processing plane from the security control plane. SDS_2 utilizes concepts and techniques of cloud, SDN, and NFV.

Applying the NFV concepts for security, virtualization technologies are used to implement virtual security functions (VSFs) on a VM or industry-standard commodity hardware. These virtual security functions can be created on-demand and moved to or instantiated in strategic locations in a software-defined dynamic virtual network environment.

Applying the SDN concepts for security, network virtualization is deployed to provision virtual security networks (VSNs) connecting virtual security functions. The SDN enables the SDS_2 model to create an automatic and dynamic specific communication link among VSFs and their connected security controller within the system.

A logically centralized SDS_2 controller forms a domain-wide view of the underlying network of virtual security functions. The SDS_2 controller can program, configure, and control the VSFs autonomously through a new and efficient proposed protocol known as the Sec-Manage protocol. The proposed SDS_2 model is shown in figure 3.1. It consists of three separate planes: the security application plane, the security control plane, and the security infrastructure plane or data plane.

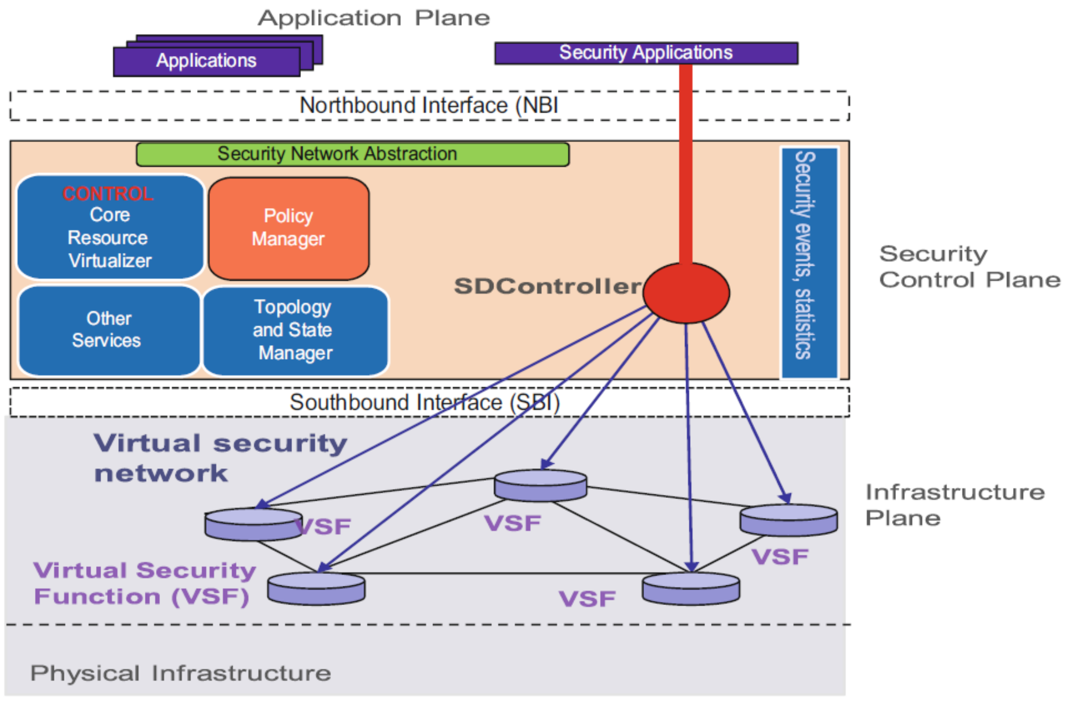


Figure 3.1 SDS$_2$ overall architecture

The SDS$_2$ security control plane, which includes one or more security controllers, provides an abstraction to build security services over virtual security elements. It is considered an SDSec network operating system that provides essential security services via interfaces: the southbound interface (SBI) to network devices and the northbound interface (NBI) to security applications.

3.3.1 SDS$_2$ Security Layers

The high-level architecture of the Software-Defined Security Service (SDS$_2$) model with three principal layers, comprising the security application layer, the security control layer, and the security data layer, is shown in figure 3.2.

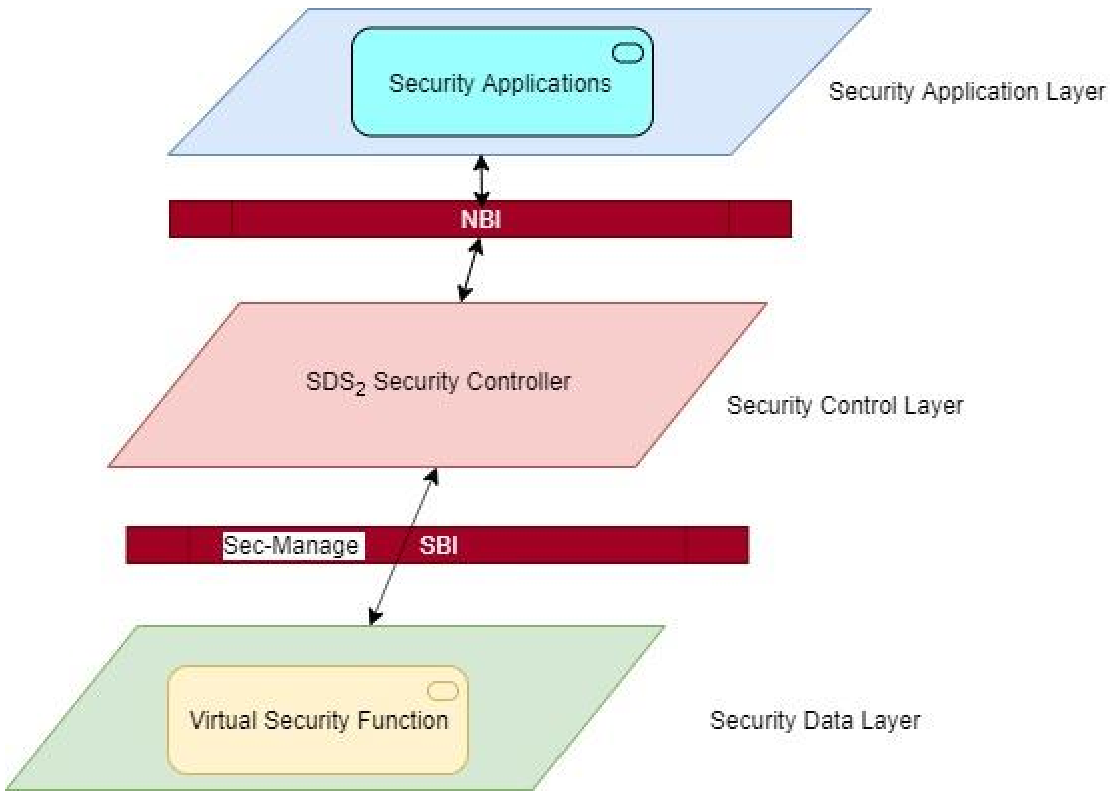

Security Applications
Security Application Layer
NBI
SDS$_2$ Security Controller
Security Control Layer
Sec-Manage
SBI
Virtual Security Function
Security Data Layer

Figure 3.2 SDS$_2$ Security Layers

- **The Security Application Layer**

The security application layer contains security applications and interfaces. The developers can deploy their security policies and applications regardless of the knowledge about the underlying security functions through a Northbound API.

- **The Security Controller Layer**

The security control layer accommodates the security controller and its components (Figure 3.3). The main component of this layer refers as SDS$_2$ Security Controller. Various components have been deployed within the security controller from analyzing interactions, interpreting security policies to defined modules to detect and predict security interaction violations. Security controller functions interpret security requirements, like security policies, and analyze interaction parameters based on the interaction model. The security controller directs security policy rules, interaction parameters, and instructions to VSFs through the Sec-Manage protocol.

The SDS_2 security controller is a software element written in java with additional components to protect cloud resources. These components allow the security controller to i) process the requested interaction from inside/outside, ii) control, orchestrate, and manage virtual security functions to monitor interactions, iii) intelligently detect and predict security violation according to interaction parameters. Details of the security controller are described in Chapter 6.

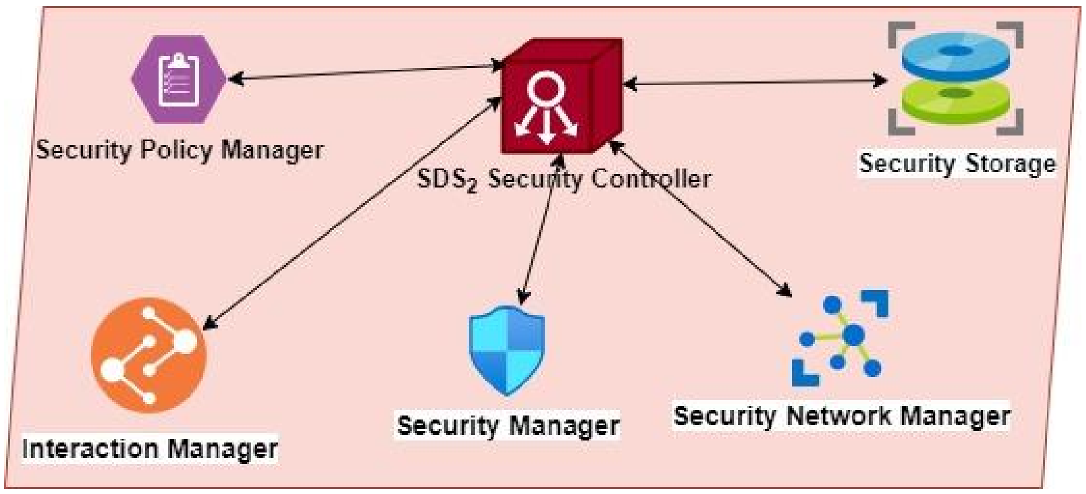


Figure 3.3 Security Control Layer

- **The Security Data Layer**

This layer is composed of virtual security functions and Sec-Manage protocol (Figure 3.4). The VSFs can form an individual or a cluster to monitor a specific or group of interactions triggered within the system. The interaction can be categorized into three classes: i) user request, ii) interaction triggered by security controller, iii) triggered interaction within the system between resources. VSF is a simple but efficient and intelligent security function, monitoring cloud entities interactions to detect and predict security violations. It should be noted that VSFs are not switches or routers; they only perform their defined security functions and relay their data/status to their controller and other VSFs when directed, such as in chaining operations.

The Sec-Manage protocol is a simple protocol designed to program, configure, and manage VSFs according to the interaction model and allow them to report their operational status to the controller.

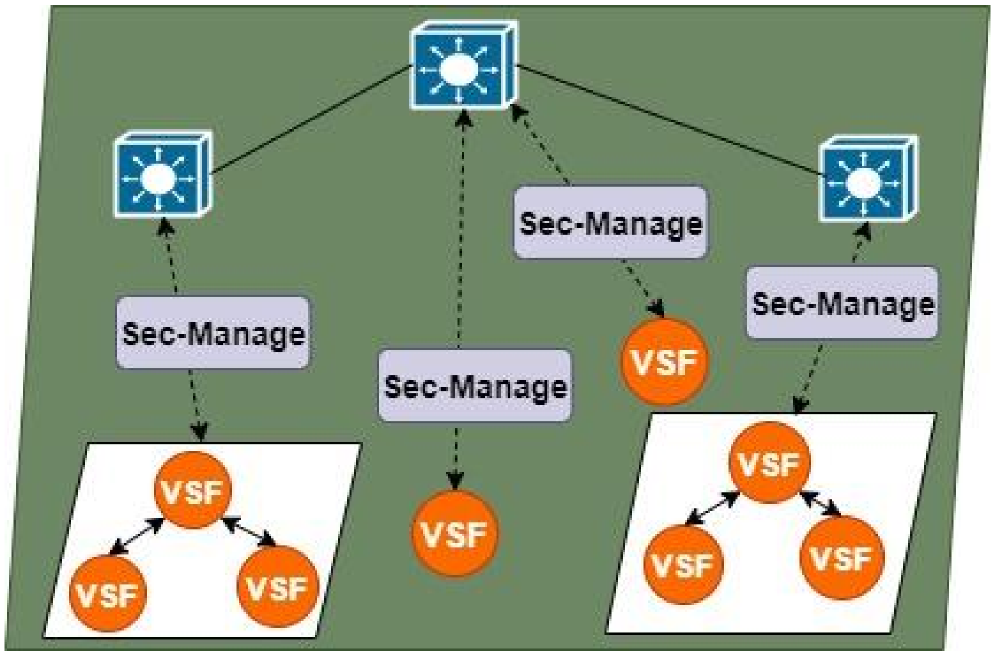


Figure 3.4 Security Data Layer

3.4 Application of SDS_2 to Data Centre Security

With the SDS_2 approach, we can design, implement, and modify the individual subsystems independently. A data center is an integrated cloud-SDNNFV infrastructure where entities include physical resources (physical servers, routers, links, storage, and their interfaces), tenants, and their virtual resources (virtual networks, virtual machines, virtual storage, virtual services, and their virtual interfaces).

A common approach to managing system complexity is identifying a set of layers with well-defined interfaces among them. Layering minimizes the interactions among the subsystems and simplifies the description of the subsystems. Security of a system is often achieved by ensuring its subsystem's integrity and authorized access to the system (subsystems) at their interfaces. The security isolation approach can identify not only physical but also virtual boundaries that are missing in traditional security mechanisms. Furthermore, security isolation effectively localizes security issues and can be tailored to deal with appropriate concerns.

With this in mind, SDS_2 can be implemented and offered as a security service to protect a data center. Depending on the data center, different numbers and types of virtual security functions can be instantiated, dynamic virtual security networks can be provisioned to interconnect those VSFs, and a logically centralized SDS_2 controller can be created on-demand to serve the required security service.

The provisioned SDS_2 configuration can be attached/imposed on the specified data center as dictated by its policies and architecture. The SDS_2 will enable security isolation through its interaction model and its software-based security functions located at critical locations in both physical and virtual layers within the infrastructure under the controller's control.

3.5 SDS_2 Features

This section describes the main feature of the proposed SDS_2 security model regarding the provision of on-demand security services in cloud infrastructure.

- **Logically centralized security controller**

A logically centralized security controller (SDS_2 security controller) located in the control plane has a global view of all the tenants, data, physical/virtual resources, and interconnections. It possesses all security policies concerning the accesses and interactions of these components. Armed with such global knowledge, SDS_2 can comprehensively provision a security service to monitor, detect, and protect the infrastructure. The security controller directly controls and manages virtual security functions.

- **Provision on-demand virtual security functions**

The feature represents the ability of the SDS_2 model to provide on-demand security functions to instantly respond to security threats at any time. Therefore, virtual security functions require to be automatically orchestrated to provision on-demand virtual security functions and at any place within a cloud infrastructure—these specific virtual security functions are designed to provide on-demand monitoring relying on an interaction model.

The SDS_2 enables the flexible and dynamic provision of on-demand virtual security functions in scalable cloud infrastructure. A security service can be established by individual/chaining virtual security functions, placed in a critical position within the cloud concerning interaction among cloud resources. According to on-demand security protection requirements, the SDS_2 orchestrates its specific underlying virtual security functions to achieve both security detection and prediction required by cloud resources.

- **Virtual security functions automatic programmability**

The SDS_2 enables the programmability of virtual security functions over interaction parameters through an innovative protocol. The security controller can program the data flows over the core network to deliver required security data, including interaction parameters from the security controller to VSF or vice versa. It dynamically programs the virtual security function according to interaction parameter changes.

- **Dynamic Security Isolation**

In multi-tenant cloud architecture, isolations are a crucial concept for both security and infrastructure management, and they ought to be considered at functional entity levels and appropriate abstraction levels of the infrastructure. Defining object boundaries is extremely difficult because virtual objects are dynamic in both characteristics and functionality. The construction of security boundaries in a cloud system is related to the characteristics of the interacting entities in the environment and the policies and constraints that govern their interaction.

In the SDS_2 security model, security policies are used to construct security boundaries between cloud objects at their interaction level. A novel policy-based interaction model is deployed to build security boundaries according to proposed interaction parameters dynamically. The model is governed by cloud system security policies and constrained by cloud interacting entities locations and levels. Security policies are used to construct security boundaries between cloud objects at their interaction level.

The SDS_2 constructs security boundaries dynamically at the interaction level. The SDS_2 uses the security policies/rules over the proposed interaction parameters model and the constraints imposed on the interacting entities. The security model builds a robust, dynamic, and automated security boundary to protect cloud assets relying on a solid and innovative interaction model and security policy expressions that govern the interactions.

- **Intelligent security detection and prediction mechanisms**

The SDS_2 security model proposes a proactive mechanism against security threats within a cloud infrastructure. The security model introduces a novel policy-driven interaction model that governs the relationship among entities in the cloud environment and develops intelligent algorithms for security breach detection and prediction.

The Policy-based interaction model proposes to securly construct dynamic security boundaries formed by authentic interaction parameters based on security rules extracted from the governing security policies. The model provides a framework for incorporating system security policies and entity constraints in constructing interaction boundaries and defining a security dictionary of expected/unexpected behaviour of cloud entities while they access resources in the cloud environment. The model concentrates on new dynamic security policy approach at the interaction level between cloud entities through a set of interaction parameters. The SDS_2 model focuses on detecting and predicting interaction security violations at the interaction level over violation of interaction parameters.

The interaction model is defined by parameters that control activities among components/entities in a cloud system. The model relies on an object model of interactions defined by four major parameters as Mode (M), Positional Relationship (R), Action (A), and time (t). Each parameter consists of different values. The M refers to possible mode and direction between entities during an interaction. The R represents all possible relation between entities considering their role at the time of an interaction. The A signifies possible actions between entities during an interaction. The t refers to valid time of an interaction.

According to the proposed policy-based interaction model, the SDS_2 model deploys automatic detection and prediction algorithms called ISVDP to identify security breaches related to interaction parameters. The algorithm also maps out possible future attacks based on expected violations of the currently defined interaction parameters. The algorithm automatically detects and predicts security boundary violations against interaction parameters related to requested interaction to validate/invalidate the requested interaction parameters.

- **Virtualization with SDN and NFV technologies**

SDN: by leveraging the SDN paradigm, the SDS_2 is able to provide an automatic and programmable security network between itself and its virtual security functions. It enables the security controller to dynamically create a connection between the security controller and VSFs at the required time. The principal of SDN and its protocol inspired the developing of an innovative Policy-based interaction protocol to control and manage virtual security functions.

NFV: The SDS_2 is inspired by NFV technology and virtual network function concepts to construct its unique virtual security function based on the interaction model. The VSF in our usage is created to perform a specific security function and is deployed at strategic locations in the cloud infrastructure that require protection. It is a software-based function constructed to protect cloud infrastructure against any type of interaction violations. VSF is a simple but efficient and intelligent security function, monitoring cloud entities interaction to detect and predict security violations.

In summary, the proposed SDS_2 delivers security as a service over a cloud infrastructure through defining security boundaries for cloud infrastructure with a novel interaction-based security isolation technique. It inherits the concepts of centralized controlling and separation of decision-making from security functions to build an intelligent and automated programmable security model.

3.6 Thesis Roadmap

It is worth noting that this chapter presents an overview of our software-defined security service model for provisioning on-demand security services and the construction of dynamic security boundaries. To build a robust security service model to provide security isolation for protecting cloud resources efficiently, the security service model requires the capability to orchestrate, manage and control on-demand security functions. This research leverages the SDN and NFV technologies.

Although SDN and NFV technologies are capable of meeting demands in relation to automation, programmability, and dynamic creation of network functions, there are significant challenges in applying these techniques to security architecture, mostly to protect cloud resources. In this research study, we propose the SDS_2 model to overcome challenges in adopting these technologies in constructing dynamic security boundaries.

We propose a software-defined security architecture to provide programmable, automated, and dynamic orchestration and management of on-demand security functions. We propose an intelligent security policy-based interaction model consisting of Interaction Security Violation Detection/prediction mechanisms to construct security boundaries. We present a dynamic and intelligent Virtual Security Function (VSF) to monitor interaction among cloud entities to enrich the limited functionality of security functions. To control and manage VSFs, we propose a policy-based interaction manage and control protocol, termed Sec-Manage. We design a security controller (SC) to control and manage VSFs and construct security boundaries within a cloud infrastructure.

To pave the way for designing and deploying an innovative integrated software-defined security platform, a roadmap for the rest of this thesis has been planned as follow.

Chapters 4, 5, and 6 describe the design and operation of the proposed Interaction Model, Sec-Manage protocol, and SDS_2 architecture together with novel components, including SDS_2 Security controller, Sec-Manage Protocol, and VSFs.

Chapter 4 describes an innovative policy-based interaction model, including its components and essential algorithms to detect and predict security violations. We propose a novel policy-driven interaction model that governs the interactions among entities in a cloud environment. According to our best knowledge, this is the first approach to use interaction parameters for building dynamic and automated security boundaries.

Chapter 5 describes the design and the implementation of the Sec-Manage protocol. The significant contributions of this chapter include: 1) Design and develop a novel Sec-Manage protocol that governs communication between the security controller and its VSFs. The Sec-Manage protocol focuses on transferring interaction security messages and required parameters between the security controller and VSFs. 2) A novel approach in programming behavior and configurational management of VSFs according to the proposed interaction model.

Chapter 6 describes the SDS_2 architecture and its main component, the security controller. It provides the overall design of the security architecture. The architecture proposed a novel security architecture that enables automation, programmability, and provision of on-demand security services to protect cloud resources against interaction security violations.

Chapter 7 describes the implementation of the SDS_2 platform with implemented elements SC, VSF, Sec-Manage, and policy-based interaction function. We also evaluate the performance of the platform in the provision of on-demand VSFs.

Chapter 8 concludes this dissertation by summarizing this study and suggesting future work.

3.7 Summary

This chapter presented the overall picture of our proposed software-defined security service model to provide on-demand security services and outlined the roadmap of this thesis. Firstly, we discussed the need for as well as the challenges to effectively construct security isolations within interaction level in cloud infrastructure to protect cloud resources. We then provided a high-level description of the proposed SDS_2 model. We discussed the main features of the SDS_2 model in relation to the provision of on-demand security services. Finally, we presented the Thesis.

Chapter 4

SDS_2 Policy-based Interaction Model for Cloud Security Breaches detection and Prediction

4.1 Introduction

Security breaches primarily result from some violation of the rules of interaction (or policy that governs the interaction) between objects when they interact. Unless one has a formal model of an interaction between objects, it is difficult to detect, predict or prevent security incidents. It has been recognized that security policies play a crucial role in all secured systems because they define what constitutes a security breach. In other words, security policies define the rules for secure interaction between objects in an environment.

This chapter introduces a policy-driven entity interaction model and algorithms to detect and predict security violations in cloud infrastructure. The focus is on object interaction and constraints, security policy, and security boundary isolation. We introduce a policy-driven interaction model that governs the relationship among entities in the cloud environment and develop algorithms for security violation detection and prediction.

The interaction model is defined by parameters that control activities among components/entities in a cloud system. The model provides a framework for incorporating system security policies and entity constraints in constructing interaction boundaries and defining the security dictionary of expected/unexpected behavior of cloud entities to access resources in the cloud environment.

This chapter is organized as follows. Section 4.2 describes the cloud object model to be used by the interaction model. Section 4.3 describes the proposed interaction model and its parameters. Section 4.4 discusses the proposed security policy-based interaction model. Section 4.5 describes ISVDP algorithms. Section 4.6 presents an evaluation of the proposed algorithms by simulating various interaction scenarios. Section 4.7 concludes this chapter.

4.2 Cloud Object Model used for Interaction Model

This section described the cloud object model and required components to be used in our interaction model.

A cloud is built by cloud providers with a multitenant architecture for sharing resources and delivering services. A cloud platform can be deployed as a public cloud, a private cloud, a community cloud, or a hybrid cloud. It is entirely service-oriented, which means everything is delivered as a service to cloud clients. Cloud virtualization technology provides a flexible and scalable on-demand computing and sharing resources to its clients. Virtualization also provides some degree of isolation among cloud tenants regarding services, applications, tenant networks, operating systems, and other resources. However, these isolation and virtual boundaries are not often visible to the relevant security controller and hence, securing the cloud, and its services is challenging. Virtual data centers are relying on their virtual technologies for provisioning and sharing their resources.

Traditional physical security mechanisms are not effective in dealing with threats and activities from virtual systems and virtual resource components, [123] as virtual boundaries between virtual components are not often well defined. Additionally, scalability of resource sharing in a multitenant cloud architecture introduced a new research area in academia and industry to build centralized security controllers capable of provisioning, monitoring and isolating physical and virtual resources in large-scale cloud data centers.

The infrastructure desires a logically centralized security controller with visibility on security boundaries within different layers. For this purpose, a security model was proposed as a dynamic, intelligent, and automated security service/model to tackle mentioned challenges in a multi-tenant cloud infrastructure [134]. It provides a security model as well as a security service called SDS_2 that applies on the object-oriented entities of a cloud environment, the interaction among them, and security policies that govern the interaction. The SDS_2 provides a security architecture to protect cloud assets with policies mapped to the cloud, tenant, and resource security policy levels.

The main idea of our centralized model centers around the interaction between constrained entities and is governed by system security policies for the detection and prediction of security breaches. An interaction can be defined as a *"relation"* between the different objects during a specific time slot. So, to define the security boundaries in terms of interaction, the system requires a sustainable object model. In our security architecture, we define an object as *"a component or a sub-component, both virtual and physical, that participates in a cloud environment and that can access/be accessed by other objects according to their properties, security constraints, and system policies."*

An object has a number of attributes, some are common among all objects (generic), and some represent specific constraints and characteristics of the object (specific). Each attribute defines some properties and hence together constitute a boundary for an object relative to other objects in its environment.

Generic parameters are pointing to common parameters required by all objects. These parameters are mandatory and identify the object (ID), the type of object in terms of its security policy level, which can be Cloud, Tenant, and Resource levels as defined hierarchically by our system. The Resource policy level identifies five sets of resources at the same level: Compute_Resource, Storage_Resource, Network_Resource, Application_Resource, and User_Resource.

Specific parameters refer to parameters that specify specific features/attributes of an object concerning its hierarchical position within the architecture identified by the role of the object and the object location that identifies its logical zone within the system. It should be noted that the policy level is related to the role of an object, and the location is related to the logical/physical location of an object. Each specific parameter carries explicit characteristics pertaining to that particular object.

An object can be simple or complex. A complex object includes nested attributes and may consist of a set of sub-objects. An object can be internal or external to a cloud, depending on its role/interaction. It should be noted that the policy level is related to the role of an object, and the location is related to the logical/physical location of an object.

The security model defines three main objects associated with the cloud, tenant, and resource security domains. Corresponding objects to these are Cloud Object, Tenant Object, and Resource Objects which include Compute Object, Storage Object, Network Object, App Object, and User Object.

The *cloud domain,* where cloud objects reside*,* classifies all the data, resources, and interactions at the cloud level while ignoring information related to lower domains like Tenant and Resource. At this level, the main parameters include cloud security policies (SP), which govern interactions policies among objects at the cloud level; data and resources policies, which only concentrate on cloud resource level (tenant, cloud-compute, -net, -storage resources) (Figure 4.1).

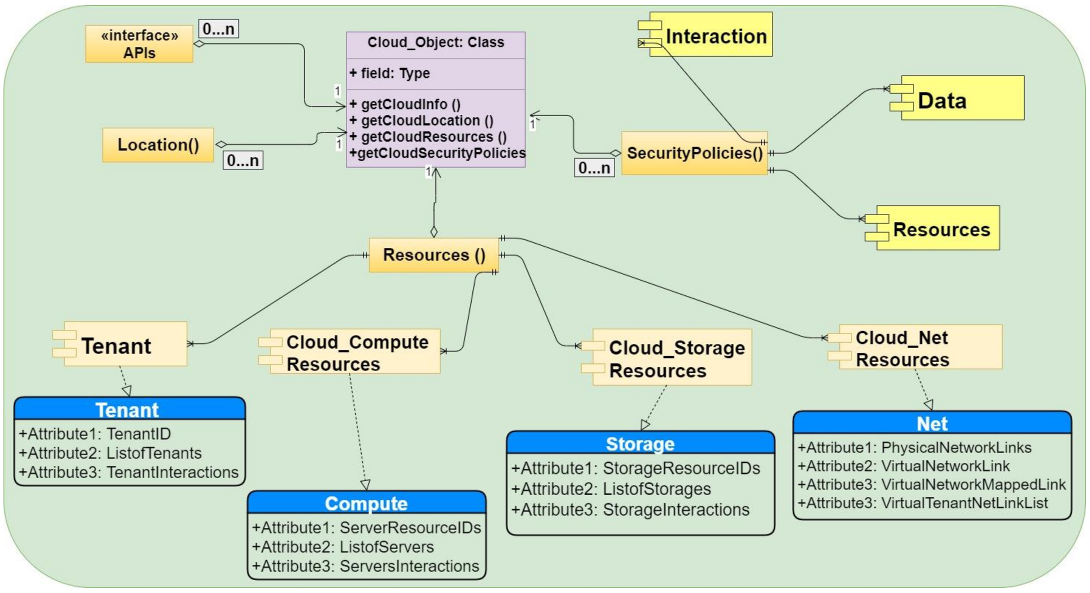

«interface» APIs
0...n
Cloud_Object: Class
+ field: Type
+ getCloudInfo ()
+ getCloudLocation ()
+ getCloudResources ()
+getCloudSecurityPolicies
Interaction
Data
Location()
0...n
SecurityPolicies()
0...n
Resources
Resources ()
Tenant
Cloud_Compute Resources
Cloud_Storage Resources
Cloud_Net Resources
Tenant
+Attribute1: TenantID
+Attribute2: ListofTenants
+Attribute3: TenantInteractions
Storage
+Attribute1: StorageResourceIDs
+Attribute2: ListofStorages
+Attribute3: StorageInteractions
Net
+Attribute1: PhysicalNetworkLinks
+Attribute2: VirtualNetworkLink
+Attribute3: VirtualNetworkMappedLink
+Attribute3: VirtualTenantNetLinkList
Compute
+Attribute1: ServerResourceIDs
+Attribute2: ListofServers
+Attribute3: ServersInteractions

Figure 4.1 Cloud Object domain parameters

The *tenant domain* only reflects attributes and parameters related to the tenant objects in the cloud domain. The focus is only on the tenants' structure and their parameters and resources.

The *resource domain* concentrates on the base underlying physical/virtual resources within the cloud system as distinct from resources at the cloud and the tenant domains. They provide detailed information related to each resource-object. Resource objects are defined similarly to cloud objects but for objects in the resource domain.

A *role (Rl)* assigns some responsibility to an object and necessary authorities/privileges to discharge its duty. The role is often not static and may change as circumstances demand. A role may be simple/complex assigned to an individual or a group of objects. A role is often associated with different layers of the architecture of a cloud system. It should be noted that 'role' is best defined using formal logic that entails complex rules to deal with dynamicity and multiple inheritances.

In our design, we avoid the complexity by equating a role with a hierarchical level in our defined cloud security architecture, where its attributes are defined explicitly when the role is assigned to a cloud object. We define an *entity (E)* as an integrated object consisting of the object's role and its object structure. The entity is a key concept in our structure to detect and predict security breaches in cloud infrastructure. The role assigned to the object will be considered based on object level and position within the system extracted from defined object parameters. An object may be assigned to a role/group of roles activated at a different system level. Objects may assume more than one role with different levels of authority in different domains. We use the *entity* as the main component within the interaction model.

4.3 Interaction Model

In the following section, we introduce our interaction model and its parameters. Security will always be a concern when entities start interacting with each other and with the infrastructure. In general, interaction is *the act of performing an action by an object on another. A natural disaster can also be considered a special interaction between an external object on a set of objects.* An action always entails some effects or consequences. Potential security violations may occur when an interaction takes place against policies governing the relationship between two/more parties. Consequently, interactions play a central role in security incidents in a system.

The main focus of SDS_2 is on the protection of a cloud system by anticipating possible security breaches and preventing them from happening. The SDS_2 proposes a novel interaction model that defines exceptional interaction parameters to detect and predict security violations. The following subsections describe the detailed structure of each parameter. The scheme centers around a new interaction model, entities connected to a cloud system, and security policies governing the system.

Table 4.1 Summary of Notations

Notation	Description
$\boldsymbol{E_i}$	E denotes entity i
$\boldsymbol{M}$	Denotes the interaction mode
$\boldsymbol{m_i}$	A set of mode relation values of the interaction mode
$\boldsymbol{d_n}$	A set of action direction values of the interaction mode
R	R denotes the interaction positional relationship
Rl	*Refers to set of roles on object*
$\boldsymbol{r_n}$	A set of positional relation values of R
T	Refers to interaction time consisting of $t_s(start\ time)$, $t_e(end\ time)$, $t_d(duration\ time)$, and $\alpha\ (interaction\ state)$
$\boldsymbol{A}$	Represent a set of all possible actions
P	Refers to system security policy
C	Refers to entities constraints
$\mathbf{S_k}$	Refers to set of security policies on k
$\boldsymbol{L^k}$	Refers to the location-based security policy of k interaction
$\boldsymbol{t^k}$	Refers to validate time for an interaction k
M	Set of permissible parameter values for interaction $I_{p,c}^k$
V	Set of non-permissible parameter values for $I_{p,c}^k$

In its general sense, an interaction takes place between simple or complex entities in a defined environment such as a cloud system. Figure 4.2 shows a simple/complex interaction between simple/complex entities. We proposed an interaction model for characterizing a relationship between objects.

The interaction model describes how objects interact with one another; it characterizes the *modes* of interaction, the *roles* of interacting entities, the *actions* one can perform against others, and the *time* of the interaction. In order to capture the essentials of interaction, we define an *object model of interaction* with four parameters or variables: mode (M), positional relationship (R), action (A), and time (t). Each parameter may take on a range of values. The range is determined or constrained by a) the interaction environment such as organizational policies, b) participating entities of the interaction in terms of their nature, properties, capabilities, and constraints, c) roles of the participating entities such as their relative positional relationship, and d) the time of the interaction.

These parameters will be defined later in this section. With these descriptions of the interaction object, we will be interested in the following operations:

1. We want to initialize an interaction, allowing default values for all parameters without any constraints.
2. We want to know what actions are possible and what are not, due to the constrained nature of the entities involved in the interaction.
3. We want to know if the interaction is permissible under a set of governing system policies.

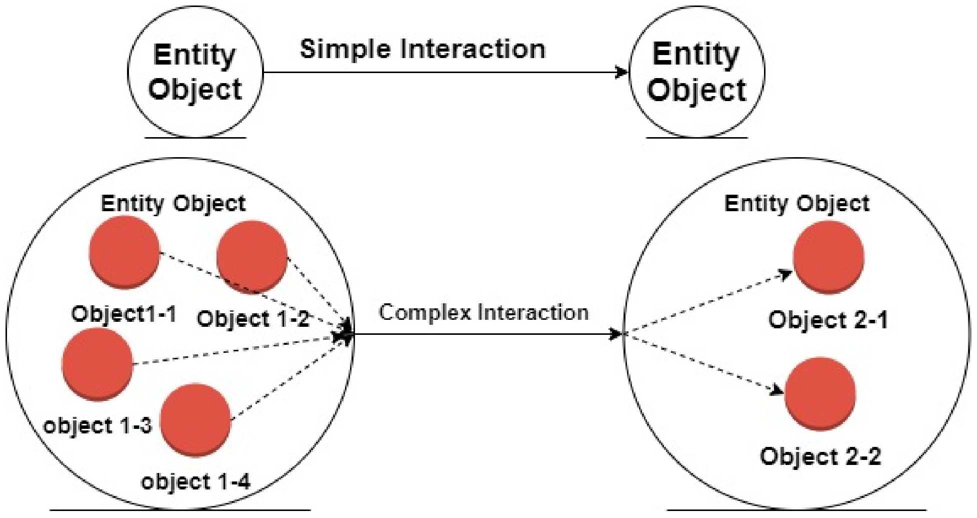


Figure 4.2 Interaction Types

Specifically, we can define a number of base operations on an interaction between entities:

- Initialize (I): Initialize I with default parameters M, R, A and t
- Mode ($I^k_{E_iE_j}$): Return all the possible modes between E_i and E_j for interaction k
- Relate ($I^k_{E_iE_j}$): Return all possible positional relations between E_i and E_j for interaction k
- Action ($I^k_{E_iE_j}$): Return all possible actions between E_i and E_j for interaction k

- State ($I^{k}_{E_iE_j}$ S_k): Return all allowed M, R, A, and t once the constraints for participating entities and security policies (S_k) have been applied to the interaction.

Additional operations involving entities, their constraints, and system policies relevant to security violation and detection will be described in Section 4.5.

4.3.1 Interaction Mode

Interaction mode (M) determines both the mode relationship (m) between objects such as one to one, one to many, and the action direction (d) from one object to another such as one way, both ways, of the interaction. Figure 4.3 illustrates all possible interaction modes of interaction. M consists of two parts: the first part refers to the mode relation (m_i), and the second refers to the action direction (d_n). So, the M is defined as a set of pairs consisting of m_i and d_n:

$$M = m_i \times d_n$$

where m_i refers to a set of possible relations between entities $m_i = \{m_1, m_2, m_3, m_4, m_5, m_6\}$; i= 1,2,3,4,5,6.

Each of the modes *m* signifies the following:

$$m_1 := 1{:}1(one{:}one), m_2 := 1{:}m\ (one{:}many), m_3 := m{:}1(many{:}1),$$

$$m_4 := m{:}m\ (many{:}many),$$

$$m_5 := 1{:}0\ (isolated), m_6 := 0{:}0\ (no\ relation)$$

An interaction's action direction may take on one of three possible values d_1 d_2, and d_3. Specifically, $d_n = \{d_1, d_2, d_3\}$ where n =1, 2, 3.

d_n may take on values and meaning as defined below.

$$d_1 = 1 :=\rightarrow (left\ to\ right), d_2 = 2 :=\leftarrow (right\ to\ left), d_3 = 3 :=\leftrightarrow (two\ way)$$

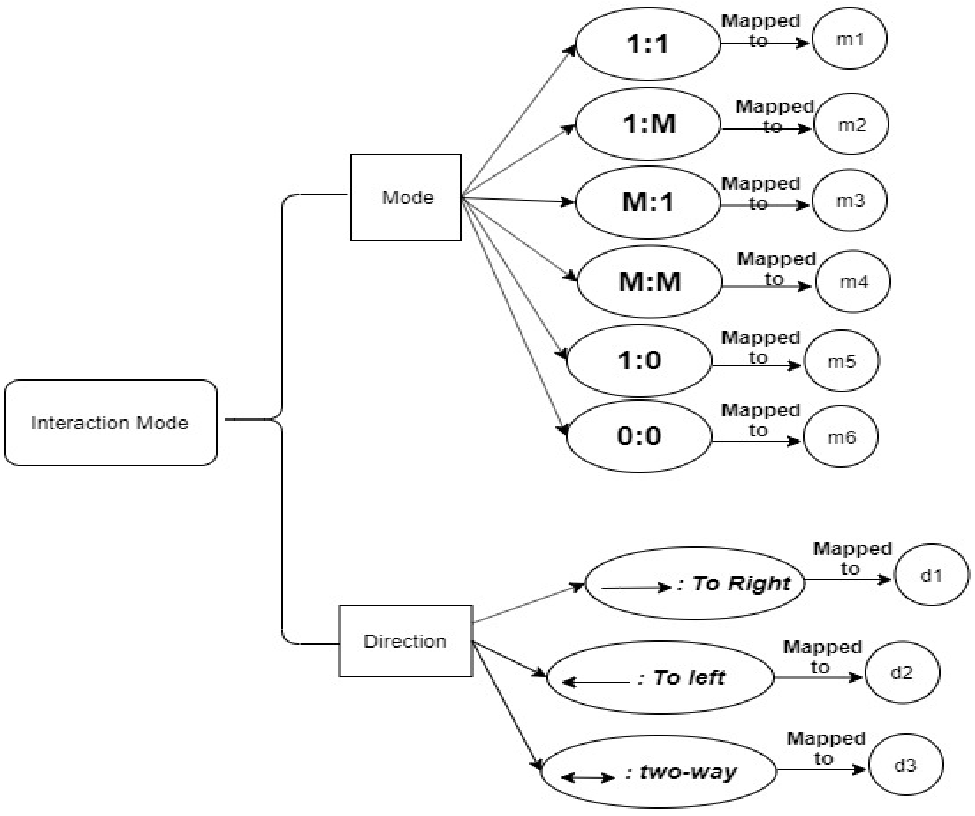


Figure 4.3 Interaction Mode

4.3.2 Interaction Positional Relationship (R)

An object within a system or an organization exists at a position either defined by its role within the organization or the layer or the domain within the system architecture. In an interaction, not only the role of an entity but its standing relative to the role of the other entities is essential as this may dictate whether the interaction is legitimate. For this reason, we consider an interaction positional relationship (R) as the relative positional relationship between the entities of an interaction.

The positional relationship determines an interaction action validity through defined rules, roles, layers, and policies associated with an interaction entities. For example, a security policy may specify that only objects at the same domain/level may interact. Interaction level is entangles with the role-based level assigned to each domain in the design. Each level entails classified security policies associated with object roles that determine a set of authorized actions. As "roles" may be of a complex nature with inheritance and may change during an entity's lifetime, we restrict and associate roles with three interaction positional relationships in any interaction between objects to three different security isolation layers of the security architecture: Cloud, Tenant, and Resource.

In this design, R denotes the interaction positional relationship according to entities relation during an active interaction.

R = {r_1, r_2, r_3} where r_1 is mapped to **down,** representing the interaction between objects from a high layer to a lower layer. r_2 is mapped to **up**, representing interaction from a low layer to a higher layer. r_3 is mapped to **equal,** representing the interaction between objects in the same layer. Knowledge about positional relationships among objects helps define the nature of the interaction and the security policy decision.

4.3.3 Interaction time (t)

Interaction time refers to the valid time for an interaction to take place in the system. The interaction time can be specified either by its start time and its end time (t_s, t_e) or start time and duration (t_s, t_d). There may be cases where the start time of an interaction is known, but its end time may be indeterminate depending on some environmental conditions. For such cases t_d is replaced by the $interaction\ state\ (\alpha)$ to indicate if the interaction is still on-going (**on**) or has stopped (**off**). Interaction time can thus be specified by:

$$t = \{(t_s, t_e)\ or\ (t_s, t_d)\ or\ (t_s, \alpha)\}$$

4.3.4 Interaction Action (A)

An interaction is meaningful if it conveys a particular set of actions. A security breach occurs when objects perform an action that violates their permissible interactions. An action is defined as a possible set of actions over an interaction between system objects by virtue of their specific relationship connected to the system.

An action is a set of possible activities that may be triggered by an event; however, the set of possible actions is often limited by nature and constraints on object/entities involved and security policy rules governing them and their interaction. Let ***A*** represent a set of possible actions that are chosen based on the types of objects found in a cloud environment. For our cloud security model, we studied cloud objects and established the set A of actions as follows:

$$A = \{\text{'read'}, \text{'write'}, \text{'modify'}, \text{'create'}, \text{'delete'}, \text{'execute'}, \text{'migrate'},$$

$$\text{'suspend'}, \text{'enable'}, \text{'disable'}, \text{' reset'}, \text{' l}\mathbf{ock}\text{'}, \text{'activation'}, \text{'}u\text{nlock'}, \text{'clear'}\}$$

Clearly, not all actions can be performed by an object as they are subject to system policies and object constraints. Table 4.2 describes the meaning of actions A.

Table 4.2 Action Description

Action	Description
Read (Re)	Permission to read the data on another Object
Write (W)	Permission to write data onto another Object
Modify (Md)	Permission to change (Write and Delete) existing data on another Object
Create (Cr)	The right to create instances of another Object
Delete (D)	The right to remove instances of another Object
Execute (Ex)	The rights to run an instance of another Object
Migrate (Mi)	The rights to re-map an instance of another Object
Suspend (Sp)	The rights to pause an instance of another Object
Enable (En)	The rights to run power up another Object
Disable (Di)	The rights to run power down another Object
Reset (Rt)	The rights to delete metadata and reboot instances of another Object
Lock (Lk)	The rights deny user access to another Object
Unlock (U)	The rights permit user access to another Object
Activate (Av)	The rights to make another Object available to a User
Clear (Cl)	The rights remove user data from another Object

4.4 Security Policy-Based Interaction Model

This section describes our policy-based interaction model and how we use security policies to detect and predict the security breaches at the interaction level. Security policies are fundamental for any effective solutions that secure an organization, a system, an infrastructure, a cyberspace, or a service because they provide a directive and scalable approach for handling a class of security issues with a single policy.

In our design, security policies are mapped to rules that determine the interaction parameters between entities. The proposed policy-based interaction model constructs dynamic security boundaries formed by legitimate interaction parameters according to security rules extracted from the governing security policies.

Our model focuses on security policies at the interaction level between entities through a set of interaction parameters. The complex structure of cloud infrastructure and the shared and dynamic nature of their resources demand robust security policy enforcement. So, it requires a clear definition of a boundary between violated and non-violated policies. Applying security policies at the interaction level allows a system to make visible previously undefined virtual boundaries between engaged entities through their interaction parameters.

In the following, we describe our policy-based interaction model and its required components. A policy can be defined as "an aggregation of policy rules" where policy rules are used to construct sets of conditions consistent with the set permissible actions [135]. Policy rules are often derived from human language statements extracted from service level agreements (SLAs) between users and service providers. NIST (2006) defines security policies as *"Aggregate of directives, regulations, rules, and practices that prescribe how an organization manages, protects, and distributes information."*

Our design security policies address rules and conditions that establish valid interactions between entities in a cloud environment. In SDS_2 architecture, we define a security policy (SP) as "*a directive that governs the interaction among simple/complex entities through specific constraints applied to the entities, their location, and their interaction parameters.*" Security constraints extracted from security policies determine the validity of a set of actions taking place during an interaction. Figure 4.4 illustrates the relationship among these components.

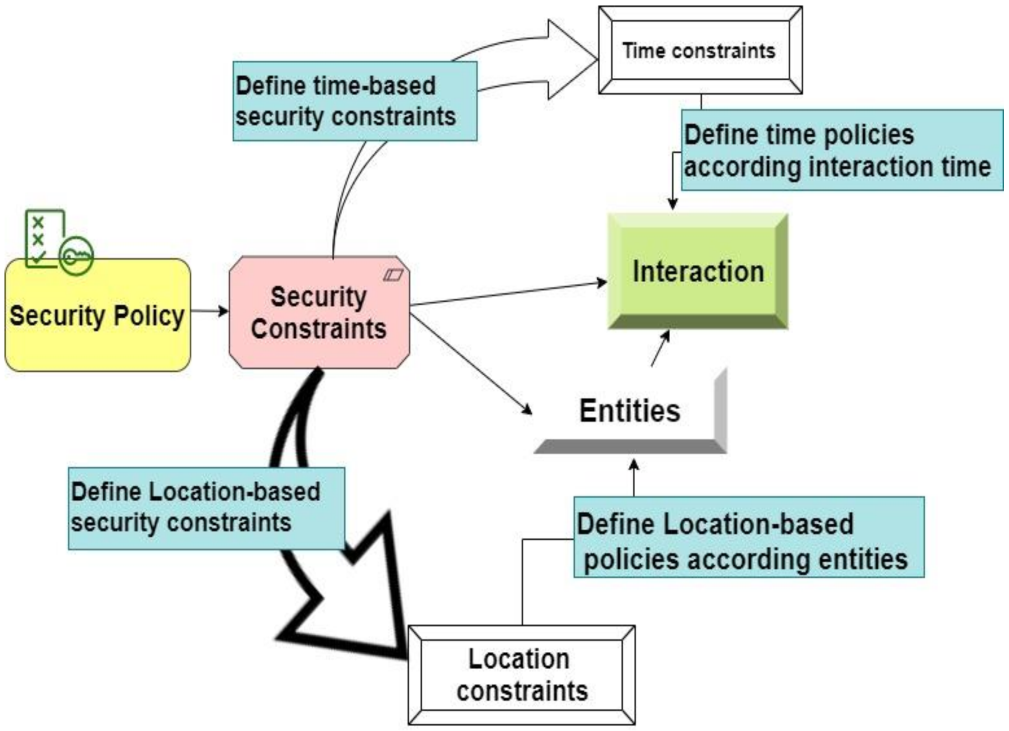

Time constraints
Define time-based security constraints
Define time policies according interaction time
Security Policy
Security Constraints
Interaction
Entities
Define Location-based security constraints
Define Location-based policies according entities
Location constraints

Figure 4.4 Security policy and its components

As discussed, system security policies, when applied to an interaction between the initiator/s (E_i), and the target entity/s (E_j), determine sets of parameters (described in section 4.2) that are secure (valid, or permissible) for the interaction. We define a *Security Policy-Based Interaction* model as shown in Figure 4.5.

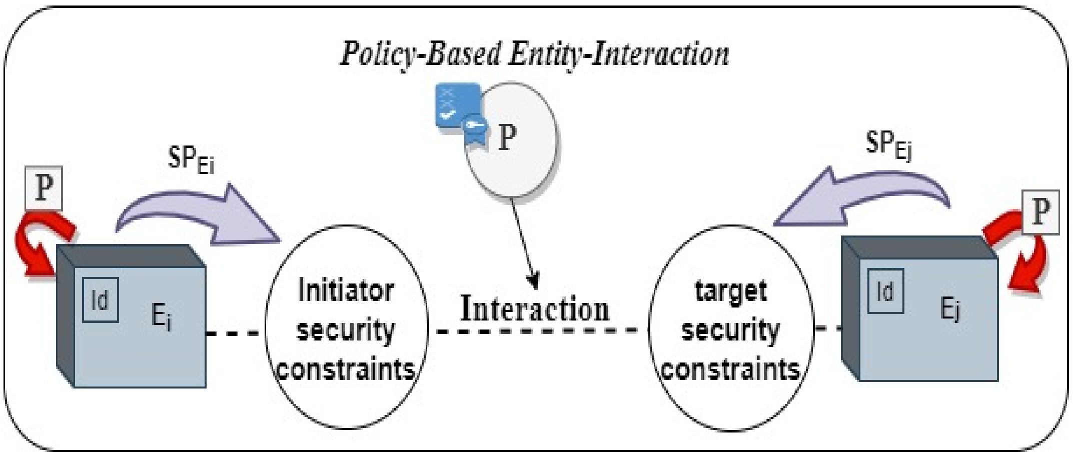

Policy-Based Entity-Interaction
P
SP_{Ei}
SP_{Ej}
P
P
Id
E_i
Initiator security constraints
Interaction
target security constraints
Id
E_j

Figure 4.5 Security Policy-Based Interaction Model

Security policies govern the validity of the parameters of the interaction. Together with the system security policies (P), security constraints (C) on entities further limit the interaction in time, isolation level, and location as defined by legitimate interaction parameters. SDS_2 architecture logically divides cloud infrastructure into three main security isolation levels (SILs) or boundaries for the Cloud, Tenant, and Resource cloud domains.

Recently, [136] introduced a security service framework with three security layers according to security domain divisions; however, the system only focused on divisions related to tenant resources and VMs in building isolation layers. We map security domains into security isolation levels that isolate each domain's entities according to their security policy levels and entities locations. Figure 4.6 shows these isolation levels.

Security policy in our context covers four aspects: system interaction policy, time-based security policy, dynamic location-based security policy, and entity-specific constraint policy. System interaction security policies are organizational sets of security policies that dictate allowable object interactions as specified by valid parameters of an interaction.

Time-based security policies dictate the valid time or time duration of an interaction. These policies are often specified at runtime because they are needed when dynamic operational circumstances demand.

Location-based security policies are required to deal with dynamic aspects of a cloud entity, such as changes in responsibility and logical/physical zone placement over time. Entity-specific constraint policies deal with the specific nature and properties of an entity. Some entities may not perform some activities because they do not possess the capability while others are capable, but their actions are constrained by relevant policies when they were instantiated. With these definitions, the set of security policies (S_k) relevant to an interaction I^k between E_i and E_j may be expressed by the following equation.

$$S_k\ \left(I^k,\left(E_i,E_j\right)\right)=P_{E_iE_j}\ \left(L^k\left(E_i,E_j\right),t^k\right)\right)=\ P_{E_iE_j}^{L^k,t^k}\qquad where\ i,j\in N\ and\ i\neq j$$

The notations are defined in Table 4.3. P denotes the system policy governing the entities, their location, and time. L denotes location-based policies for each entity. If E_i, E_j are placed in the same zone and same group zone policy, the location policies are the same for both. Security policy-based interaction model concentrates on two main policy concepts: general policies and local policies. General policies apply to all requests within the system, and local policies apply separately to each entity and their interactions within the system according to their location and assigned constraints. Both sets of policies are stored in separate security databases. Security policies are extracted during an interaction, and rules and constraints are assigned and applied to the interaction over the valid interaction time duration.

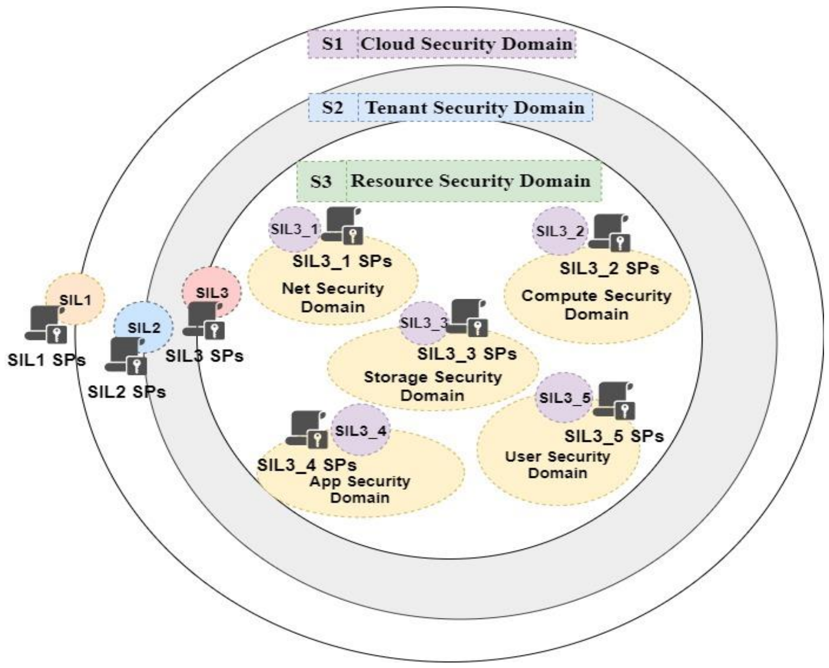


Figure 4.6 Security Isolation levels

Table 4.3 Required Notations

Notation	Meaning	Detailed expression
c_{jv}	Set of constraints associated with entity j	
E	An entity composed of role and object i	E= E_i^{jk}
I	An interaction object	
I_{init}^k	Interaction object k initialized with default parameters (unconstrained)	$I_{init}^k(*,*)$
I_C^k	Interaction object k with object constrained applied	$I_C^k(E_i, E_j)$ or I_C^k $(E_i(c_i$
I_P^k	Interaction object k with system policies applied	$I_P^k(E_i, E_j, S)$
$I_{P,C}^k$	Interaction object k with both system policies and object constraints applied	$I_{P,C}^k(E_i(c_{iu}), E_j(c_{jv}), S$
I_{req}^k	Interaction object k with parameters derived from an interaction request	$I_{req}^k(E_i, E_j)$
S_k	Interaction k policies derived from the system policies	$S_k = P_{E_iE_j}^{L^k, t^k}$

4.5 Interaction Security Violation Detection and Prediction Algorithm (ISVDP)

With the introduction of the formal model of an interaction and its relationship with security policy, we propose an interaction security violation detection and prediction (ISVDP) algorithm. The ISVDP operates over the SDS_2 cloud infrastructure with three levels of security isolation. The algorithm automatically detects and predicts security breached in relation to requested interaction according to validate/invalidate interaction parameters. The main parameters of ISVDP include:

- Initiator entity: an entity that initiates a relationship with another entity and establishes an interaction.
- Target entity (or Reactor): the entity of an interaction on which the initiator intends to perform certain actions.
- Entities constraints: the constraints extracted from local policies related to both initiator's and target's role, type, and their intrinsic properties.
- A complete set of system security policies defined over SDS_2 cloud and its isolation levels: Cloud, Tenant, and Resources.
- A requested interaction between the initiator and the target entities (for violation detection).

In ISVDP, a constraint is represented as *"a security statement which defines a set of conditions that limits the scope and the property of an interaction between an initiator and its target entity."* High-level security policies are written in human-language policies, which will be translated using a policy-translator within the SDS_2 controller. Armed with the translated security policies, a security controller determines the validity of an interaction between entities based on their defined interaction parameters. The detection and the prediction algorithms form two fundamental components of the ISVDP model. Both of them share and are built upon the initial three processing stages, as shown in figure 4.7 for a specific interaction k.

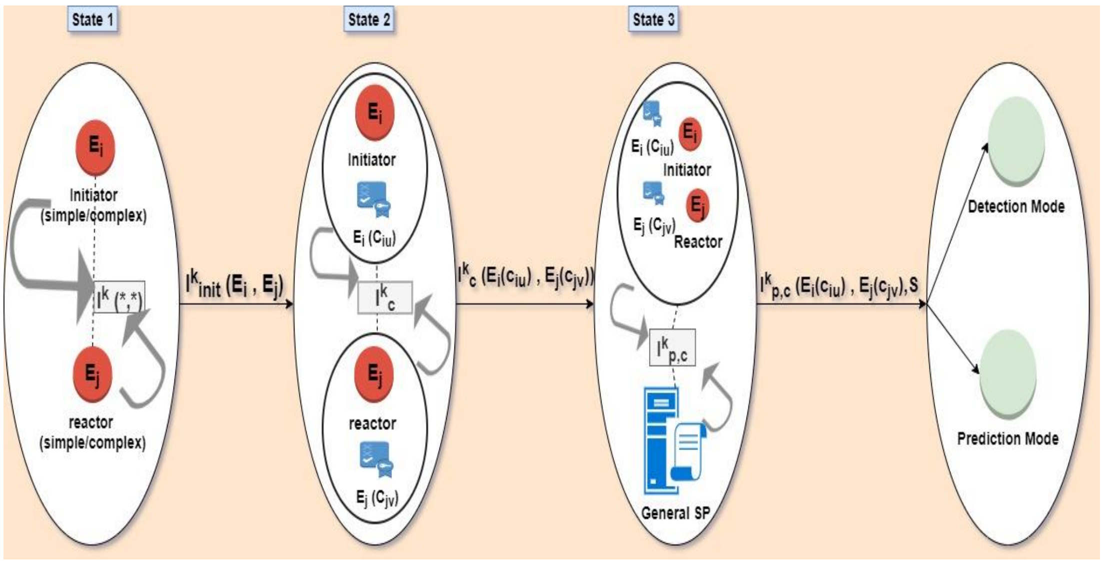


Figure 4.7 ISVDP stages

We define the following notations in Table 4.3. We define the following basic set of operations on an interaction object with the above notations, as shown in Table 4.4.

Table 4.4 Operations Defined on an Interaction Object

Operation	Meaning	Detailed expression
Initialize (I)	Initialize I with default parameters M, R, A and t	
Mode (I^k)	Return possible modes between E_i and E_j for inter-action k	$Mode\ of\ (I^k_C\ or\ I^k_P or\ I^k_{P,C})$
Relate (I^k)	Return possible positional relations between E_i and E_j for interaction k	$Relate\ of\ (I^k_C\ or\ I^k_P or\ I^k_{P,C})$
Action (I^k)	Return possible actions between E_i and E_j for inter-action k	$Action\ of\ (I^k_C\ or\ I^k_P or\ I^k_{P,C})$
Const (I^k)	Return possible interaction parameters after apply-ing constraints on interaction k	Const on (I^k_{init})
State (I^k)	Return all states of interaction k between E_i and E_j	$State\ of\ (I^k_C\ or\ I^k_P or\ I^k_{P,C})$
State (I^k, req)	Return all states the interaction as required by the request	$State\ of\ (I^k_{req})$
Policy (L, E_i)	Returns the set of system policies applied to entity i location	
Policy (I^k, req)	Return the set of system policies applied to interac-tion k	Policy on (I^k or I^k_{req})
Compare (I^m, I^n)	Compare the states of interaction m and interaction n, return differences in M, R, A, and t	

Stage 1: Initializing the interaction. At this initial stage, the objects involved in the interaction are made available with their security-rated properties. The interaction template is initialized with no constraints on interaction parameters. The requested interaction is also made available. The result of this stage is the object $I_{init}^{k}(*,*)$.

The algorithm intelligently identifies all involved entities and components in this stage. Additionally, the algorithm detects interaction types and parameters. Dynamically it can change interaction parameters according to the location and nature of entities and create initial interaction parameters between two entities.

Stage 2: Application of entity constraints over the interaction k. At this stage, the interaction parameters are modified according to the properties and constraints of the entities involved. The result of this stage is the object $I_{c}^{k}\left(E_i(c_{iu}), E_j(c_{jv})\right)$.

Stage 3: Application of the policy over the interaction k. At this stage, the parameters of interaction k will be modified by the constraints derived from the system policies that are applicable to the interaction k. The result of this stage is the object $I_{p,c}^{k}(E_i(c_{iu}), E_j(c_{jv}), P)$. The policy-driven interaction algorithm encompassing stages 1, 2, and 3 is shown in Algorithm 4.1.

Algorithm 4.1 Policy-driven interaction algorithm (PdI ())

```
Input: $E_i$, $E_j$, $SDS_2$ cloud objects' DB, System Policy statement (P)
Output: $I_{p,c}^{k}$ (M', R', A', t')
1: while request is valid do
    1: for $E_i$, $E_j$ do
        2: Intialize ($I^k$) //get interaction parameters for $E_i, E_j$ without applying constraints and set $I_{init}^{k}$
        3: if $I_{init}^{k} \neq$ Null
                Const ($I_{init}^{k}$)  //get interaction parameters by applying constraints on $I_{init}^{k}$ and set $I_C^k$
            4: $I_C^k$ = State ($I_C^k$) // return parameters after applying constrains
            5: Policy ($I^k$) // get system policies (P) applied to the $I^k$
            6: $I_{p,c}^{k}$ = State ($I^k$) // return parameters after applying policy system
        7: end if;
    8: end for;
9: end while;
```

4.5.1 Interaction Security Violation Detection

In case of a triggered interaction within the cloud system, the detection algorithm determines if the interaction is safe or violates a system's security policy or specifically if a security breach has occurred. With the global knowledge of the cloud environment and the interactions among entities, the security controller intelligently schedules to execute the ISVDP algorithm on suspicious circumstances, on a specific request or triggered events, or on a regular basis.

The algorithm considers each interaction parameter under consideration to discover if any inconsistency has occurred relative to the security policies, hence the interaction parameters, dynamically applicable to the interaction. The module goes through the four fundamental stages described above and proceeds to stages 4d, 5d, 6d, and 7d for violation detection.

Stage 4d: The requested interaction policy level is analyzed according to defined security isolation levels explained in section 4 *(Domain (I^k_{req})).*

Stage 5d: The interaction under consideration between the specified objects is analyzed, resulting in a set of interaction statuses required by request: $I^k_{req}(E_i, E_j)$.

Stage 6d: The algorithm intelligently detects each object interaction parameter rules based on security domain and location Domain *($I^k_{req}(E_i, E_j)$)* and Loc *(E_i, E_j).*

Stage 7d: By analyzing $I^k_{p,c}\ (E_i(c_{iu}), E_j(c_{jv}), P)$, and $I^k_{req}(E_i, E_j)\ interaction\ parameters$ the algorithm determines if the requested actions are within the set of actions allowable by the policies and the constraints imposed on the entities of the interaction.

The algorithm returns the validation status of the interaction: either Safe or Violate. Safe means that the requested interaction does not violate any policy related to each/any interaction parameter and is not a security breach. Violate means that the requested interaction violates one of the parameters (M, R, A, t) or location of the allowed security policy that governs the interaction. The algorithm returns whenever a violation of an interaction parameter is detected. However, in cases where policies governing the interaction parameter are undefined (either due to an oversight or situations not yet encountered), it will decide if there is a possibility to partially accept the interaction and initiate an alert to a decision-maker to create a new policy to cover the newly discovered situation. Figure 4.8 shows the decision process of the ISVD algorithm.

We use (M', R', A', t') to denote State $(I^k_{p,c})$ and $\left(M'', R'', A'', t''\right)$ to denote State $\left(I^k_{req}\right)$. In the detection process, all system policies, including location, entity constraints, are applied to the interaction k to obtain all the interaction's allowable parameters. Figure 4.8 shows the detection approach in determining the validation status of the requested interaction k. The detection algorithm will stop the process of discovering the first interaction parameter violation and activate the security alarm within the security controller. Algorithm 4.2 describes the ISVD detection algorithm, which analyses the I^k_{req}, extracting the number and types of involved entities during the requested interaction.

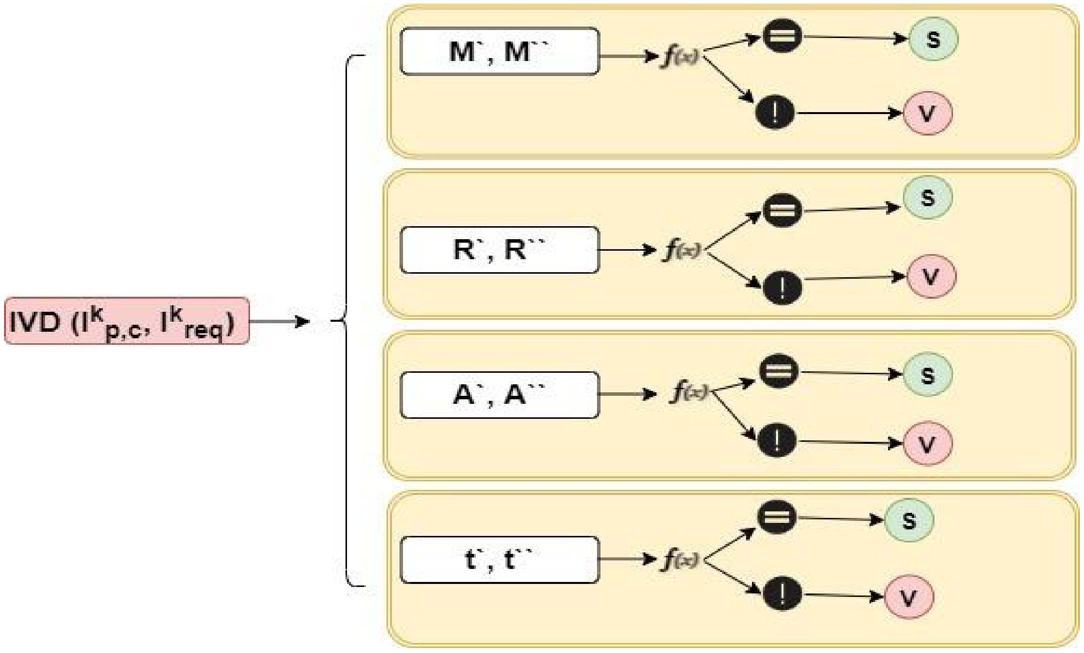


Figure 4.8 ISVD

Algorithm 4.2 Interaction Security Violation Detection (ISVD ()) algorithm

```
Input: $I^k_{req}$(requested interaction), $I^k_{p,c}$ ,
Output: Safe | Violate
1: for received $I^k_{req}$ do
  2: if Domain ($I^k_{req}$)==True then
  3: Policy (L, E) //Returns set of system policies on the current location of entities during initiation of $I^k_{req}$ and set L
    4: If L == True (location is verified) then
      5: PdI () //call the algorithm 1 to get $I^k_{p,c}$
         $I^k_{P,C}$ = State ($I^k_{P,C}$)
      6: $I^k_{req}$ = State ($I^k$, req) // get request interaction parameters
    7: for $I^k_{req}$ and $I^k_{P,C}$ do
      8: diff := Compare ($I^k_{P,C}$, $I^k_{req}$) // returns difference (diff) parameters between $I^k_{P,C}$ and $I^k_{req}$
      9: P := Policy ($I^k$, req) // returns system policies applied to interaction parameters
      10: if diff satisfies P then
        11: state ( $I^k_{req}$) is safe
        12: else state ( $I^k_{req}$) is Violate //rise security violation alarm to security controller, isolate the interaction
      13:   end if;
    14: end for;
   15: end if;
  16: end if
17: end for;
```

4.5.2 Interaction Security Violation Prediction

In this section, we describe the prediction algorithm and its functionality. The ISVP prediction algorithm enables interaction violation predictability based on permissible values of the parameters of the interaction. The algorithm is different from the detection algorithm in that it determines all “Safe” interactions and all potential “Violate” interactions under the system security policies and constraints imposed on the interaction parameters between given entities.

The prediction algorithm automatically discovers the probability of possible future violations according to the current state of validation interaction parameters. For each interaction parameter, it discovers upcoming violation values. It analyses the safe/valid state of entities interaction and predicts the possibility of future violations according to unacceptable interaction parameters within the system. The prediction algorithm considers each interaction parameter and determines to invalidate parameter' values through prediction approaches. The prediction algorithm proceeds through the three stages of Algorithm 4.1 and proceeds through stages 4p and 5p, as shown in Algorithm 4.3.

Algorithm 4.3 Interaction Security Violation Detection (ISVD ()) algorithm

```
Input: $I^k_{p,c}$
Output: V set of possible potential violate interaction parameters
1: PdI () // set $I^k_{p,c}$
2: for $I^k_{p,c}$ do
  3: while $T_M$ = Mode ($I^k_{p,c}$) do // get all possible sets of M extracted from $I^k_{p,c}$ from safe mode
       $V_M$= opposite ($T_M$) // set of possible violated mode parameters extracted from valid ($T_M$)
  4: end while;
  5: while $T_M$ = Relate ($I^k_{p,c}$) do // get all possible sets of R extracted from $I^k_{p,c}$ from safe mode
       $V_R$ = opposite ($T_R$)// sets of possible violated R from $I^k_{p,c}$ from safe mode
  6: end while;
  7: while $T_A$ = Action ($I^k_{p,c}$) do // get all possible sets of A extracted from $I^k_{p,c}$ from safe mode
       $V_A$ = opposite ($T_v$) // sets of possible violated A from $I^k_{p,c}$ from safe mode
  8: end while;
9: V = ($V_M$, $V_R$, $V_A$) // set of predicted and possible violation interaction parameters according $I^k_{p,c}$
10: end for;
```

Stage 4p: The stage outputs all possible "Safe" interaction parameters between the given entities considering all constraints and security policies.

Stage 5p: The outputs are all potential "Violate" interactions between the given entities.

It is done by inspecting each parameter (M, R, A, t) and applying security constraints on each parameter. If M_s (safe parameters defined for M during k interaction) is the allowed set of safe modes, then $M_v = M – M_s$ is the set of violating modes (M is all possible values). Similarly, R_s and R_v are the set of allowed relational positions and violate relational positions, respectively; A_s and A_v are the set of allowed actions and violate actions, respectively. Similar notations are used for time and location.

The results allow the system to predict possible security breaches if interaction parameter conditions are not met. These conditions display the predicted violations in terms of interaction parameters. Ideally, all possible violations relative to the current interaction can be discovered/predicted; however, if all the interaction parameters are allowed to vary independently of one another, the analysis can be computationally expensive and not practicable. Realistically, we may want to address and predict the most likely violations.

We thus restrict ourselves to simple situations where one parameter varies at a time to illustrate the prediction process. In the predicting state, the system anticipates all possible different situations that current interaction parameters between defined entities can face. For instance, if the valid actions between two objects are defined as "read", all other possible actions can be considered violations of interaction parameters considering object nature and constraints. So, the system can stop the violation using its stored predicted violation parameters rather than going through lower layers and nested policy discovery.

All opposite interaction parameters against validate parameters are considered potential interaction parameter violations in the presented prediction algorithm. The security controller runs the ISVDP algorithm to discover the probability of future attacks according to each interaction parameter for an interaction, say *k*. It is an intelligent mechanism that focuses on interaction parameters and their possible forthcoming violation during an interaction.

4.6 Interaction scenarios and results

In this section, we demonstrate our policy-driven security scheme by using a security controller in verifying allowable interactions and detecting policy violations between entities in a cloud infrastructure based on our proposed model of interaction. We built the security controller from scratch in Java language and run our ISVDP algorithm in an Ubuntu machine with 16 GB RAM, Intel® Core (TM) i7-7600U CPU. The results evaluate the efficiency of the ISVDP algorithms in discovering and predicting security violations.

We set different scenarios according to various interaction types and analyze the results to evaluate the proposed interaction model and its components for each case. We simulate the interaction between different types of objects within the system to detect and predict security violations according to our ISVDP model. We consider the CloudSimSDN-NFV framework to simulate the cloud infrastructure and build our security controller and ISVDP algorithm. Figure 4.9 demonstrates the implementation process.

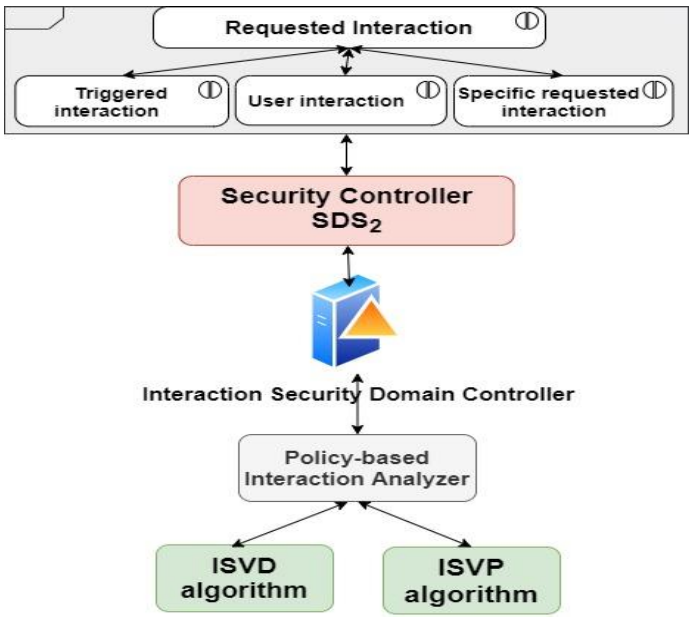

Requested Interaction
Triggered interaction
User interaction
Specific requested interaction
Security Controller SDS$_2$
Interaction Security Domain Controller
Policy-based Interaction Analyzer
ISVD algorithm
ISVP algorithm

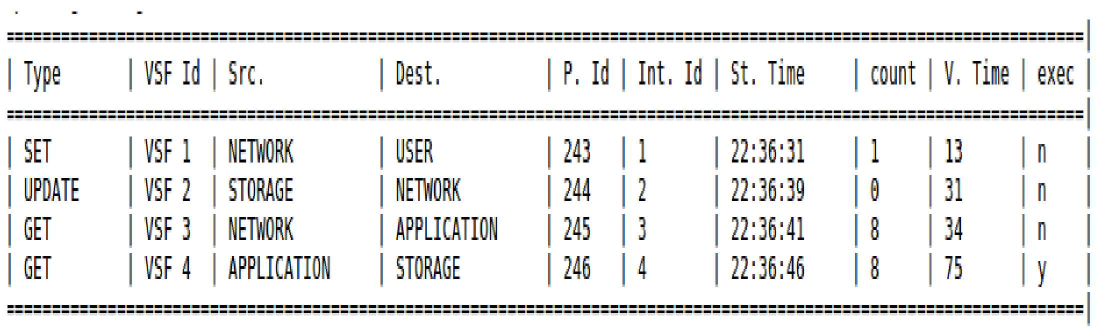
Figure 4.9 Implementation process

Scenario 1: User interaction. In this scenario, the security controller (SC) receives interactions triggered by a user. The SC identifies the user and interprets the requested interaction. According to the user level and rights, the security controller determines the security policies related to the user and involved objects. The requested interaction is sent to the interaction security domain controller to extract security policies and interaction parameters. The security controller initiates a virtual security function (VSF) designed to monitor the interaction based on received validated interaction, entities policies, and constraints. The analyzer function is responsible for running the ISVDP algorithm to detect and predict security violations.

Scenario 2: Specifically requested interaction. In this scenario, a specific interaction runs within the system. The specific interaction is considered as a request to monitor a specific interaction being performed by the security controller. This scenario occurs when the security controller decides to monitor an interaction between specific entities within the system. The security controller triggers an interaction to be monitored among specific entities. It will happen mainly in two sub-scenarios 1) randomly monitor entities based on its statistics received from its virtual security functions; 2) activates monitoring of a sensitives entity within the system on specific time slots.

Scenario 3: Triggered interaction. The security controller activates a virtual security function to monitor a triggered interaction. This scenario occurs when an abnormal interaction is triggered between entities within the system. An undesired interaction may occur. The security controller initiates and commands reports from relevant virtual security functions over suspicious entities and then executes the ISVDP algorithm to assess the situation.

The functions within the security controller perform the ISVDP algorithms to produce the results. As demonstrated in Figure 8, the system analyses the requested interaction and involved objects. Security policies and objects' constraints are extracted according to object security isolation layers and entities location. In this study, however, we mainly concentrate on the interaction between simple objects and their interactions. Figure 4.7 shows the flow process of our ISVDP algorithms. In the first step, *the system creates entities within the cloud system by substantiating the identified objects and* their defined role. Figure 4.10 demonstrates extracted data from an interaction and a VSF assigned to that specific interaction.



```
==================================================================================================|
| Type      | VSF Id | Src.        | Dest.       | P. Id | Int. Id | St. Time   | count | V. Time | exec |
==================================================================================================|
| SET       | VSF 1  | NETWORK     | USER        | 243   | 1       | 22:36:31   | 1     | 13      | n    |
| UPDATE    | VSF 2  | STORAGE     | NETWORK     | 244   | 2       | 22:36:39   | 0     | 31      | n    |
| GET       | VSF 3  | NETWORK     | APPLICATION | 245   | 3       | 22:36:41   | 8     | 34      | n    |
| GET       | VSF 4  | APPLICATION | STORAGE     | 246   | 4       | 22:36:46   | 8     | 75      | y    |
==================================================================================================|
```

Figure 4.10 Extracting involved objects and assigning a monitoring security function to each interaction

For demonstration purposes, Figure 4.11 shows a part of the security controller that investigates a specific requested interaction. The controller initializes the interaction according to analyzed parameters in a specific time slot. Then the system automatically detects and predicts required parameters and discovers security breaches.

The orange box shows extracted information from triggered interaction by the security controller, which can be run manually by the security controller. The action is a validated action that is calculated after interpreting the requested interaction. The red boxes demonstrate detection results as well as prediction parameters. We consider cloud objects of different types and determine possible allowable interactions.

For each scenario, objects can be at the same or different access levels. System policies are applied to achieve valid entity interaction parameters. For simplicity, interaction time is assumed valid the whole time under consideration. We detailed an interactive case study between two entities to describe discovering and validating the interaction between the two entities. In the following case, we describe how a policy-based interaction analyzer will extract required interaction parameters to be sent to the assigned security function.

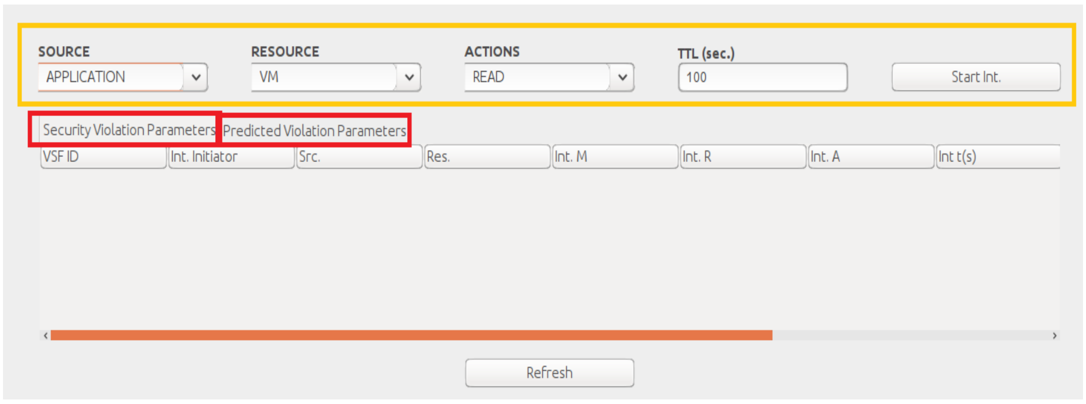

SOURCE: APPLICATION
RESOURCE: VM
ACTIONS: READ
TTL (sec.): 100

Figure 4.11 The controller interface according to interaction

VM-Storage interaction: interaction between a virtual machine and a storage entity. In the first step, the program at the controller level creates participating entities (if it has not existed yet) based on the information stored in the security database SecDB. EntityCreation.O () → Object (O_i) ∧ Role (Rl_n^O) → E_i, EntityCreation.O () → Object (O_j) ∧ Role (Rl_n^O) → E_j. Then the program discovers and predicts all possible interactions for each entity without considering their constraints: Initialize (I) → $I_{init}^k\ (E_i)|(E_j)$.

In the next step, both role-based constraints and intrinsic object-based constraints will be extracted and applied accordingly over the interaction between the two entities. Entity constraints are extracted using *Const (I^k)* and applied to the interaction parameters. As a result, possible interaction parameters for I (E_i, E_j) are determined, Const (I^k) → $I_C^k\left(E_i(c_{iu}), E_j\left(c_{jv}\right)\right)$.

The policy translator then extracts possible interaction parameters permitted by the system policy statement (P). In our testing model, we define several predefined general security policies for each scenario. Then, at the final stage, the program calls *SysPolicyapplier()* to apply policies on $I_c^k\left(E_i, E_j\right)$ and get all possible interaction parameters: PolicyInterpreter (P) → ⁽M_{sys}, R_{sys}, A_{sys}, t_{sys}), Policy (I^k, req) → $I_{p,c}^k(E_i, E_j)$.

As depicted in Table 4.5, the first and second columns show values of the VM and the storage entities interaction parameters after their constraints have been applied. The third column shows the interpretation of the system policy on object interaction parameters. The last column shows all possible interactions between the two entities as determined by the allowable interaction parameters after all constraints and system policies are considered. The results indicate that the only allowable actions are Re, W in an allowable pair of (m_1, d_2) mode of interaction between the VM and the storage entities within the cloud system.

In our system, we consider *t* as an acceptable duration time that interaction can take place. For violation detection, the security controller calls algorithm 4.2. It analyses the coming request and extracts required parameters and calls $State\ (I_{req}^{k})$. During this phase, the requested interaction statement requests the *removal of a file from the storage object requested by the virtual machine at the same level.*

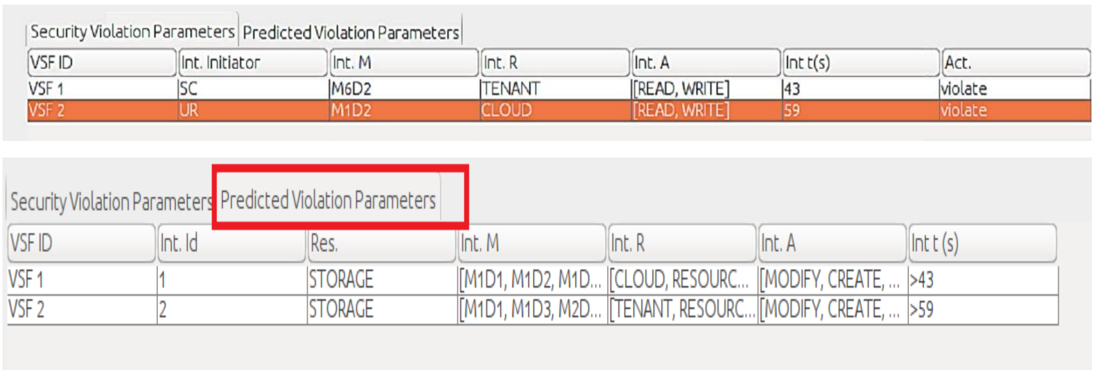
Table 4.5 Collected data from the controller for VM-Storage interaction

	I-Object 1	**I-Object 2**	**SysPolicy**	$I_{p,c}^{k}$
M (m/d)	(m_1, d_1)	(m_1, d_1)	(m_1, d_2)	(m_1, d_2)
R	Cloud	Cloud	cloud	cloud
A	Re, W, D	Re, W	Re, W, Cr, D	Re, W
T	600ms	600ms	300ms	300ms

The program translates the coming request, which detects the *delete* violation as delete action against $I_{p,c}^{k}$. It raised a security alarm, indicating a violation of the requested interaction. For violation prediction against possible attacks, the system will call algorithm 3 to predict possible violations against the parameters of $I_{p,c}^{k}$.

The system calculates possible violation parameters relative to allowable $I_{p,c}^{k}$ parameters. In this case, interaction actions except for Re, W are considered as action violations. More importantly, this algorithm can enumerate all possible interaction violations between two entities (those not allowable by $I_{p,c}^{k}$) by systematically going through the mode, the positional relationship, the action, and the time parameters of the interaction.

For example, if we keep all parameters except the mode parameters fixed, we can declare that other modes except m_1 and d_3 are potential (or predicted) violations. Similarly, the system considers any positional relation except *cloud* as a security breach and stores the data. An insider/outsider request that involves any of the predicted violation parameters will be investigated in anticipation of potential security breaches: *ISVP* $(I_{p,c}^{k})$ → $V\ (V_M, V_R, V_A, V_t)$ (Figure 4.12).


Security Violation Parameters
Predicted Violation Parameters

VSF ID	Int. Initiator	Int. M	Int. R	Int. A	Int t(s)	Act.
VSF 1	SC	M6D2	TENANT	[READ, WRITE]	43	violate
VSF 2	UR	M1D2	CLOUD	[READ, WRITE]	59	violate

Security Violation Parameters
Predicted Violation Parameters

VSF ID	Int. Id	Res.	Int. M	Int. R	Int. A	Int t (s)
VSF 1	1	STORAGE	[M1D1, M1D2, M1D...	[CLOUD, RESOURC...	[MODIFY, CREATE, ...	>43
VSF 2	2	STORAGE	[M1D1, M1D3, M2D...	[TENANT, RESOURC...	[MODIFY, CREATE, ...	>59

Figure 4.12 Implementation results for VM-Storage interaction

We executed various tests according to various scenarios to show expected results. Table 4.6 reveals some result samples that the security controller captured by performing many cases. In the table, *Int* reveals validated parameters expected after running the PdI () algorithm. After running the ISVD () algorithm, it shows the results using the *Act* parameter (s: safe, v: violate). We monitored our system's performance according to the number of interactions that are triggered within the system from any resources, the detection processing time, and the time until the system detects the status of the requested interaction.

Table 4.6 Expected results of the simulated scenarios

VSF ID	Src.	Res.	Int. Init	Int.				Act.	P. Id	exec
				M	R	A	t			
VSF 6	VM	Storage	SC	(m_1, d_3)	Cloud	Re, W	3000ms	s	3	Y
VSF 3	User	Storage	SC	(m_4, d_3)	Tenant	Re, W	900ms	v	3	N
VSF 2	Storage	APP	UR	(m_1, d_1)	Cloud	Md	10500 ms	v	22	Y
VSF 9	User	App	AT	(m_4, d_2)	Re-source	Md	1000ms	v	23	Y
VSF 7	VM	Storage	SC	(m_2, d_2)	Tenant	Re, W	800ms	s	30	Y
VSF 11	App	Storage	SC	(m_5, d_2)	Re-source	Re, W	600ms	v	13	N
VSF 8	Net	VM	UR	(m_4, d_1)	Tenant	Ex, Re	600ms	s	19	N
VSF 22	Storage	VM	AT	(m_1, d_2)	Cloud	Re	300ms	v	22	N

Figure 4.13 illustrates the system's average decision processing time in dealing with various interactions from different resources. The number of interactions is the average of the decision processing times for all three defined types of interaction. Figure 4.14 represents the relation between several interaction classes within the SDS_2 security controller software.

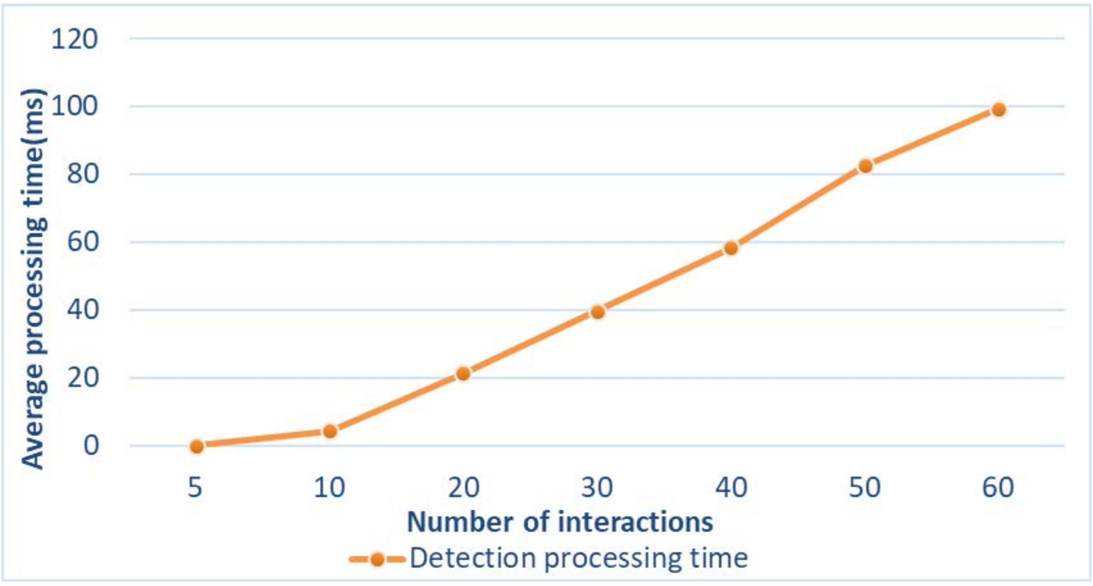
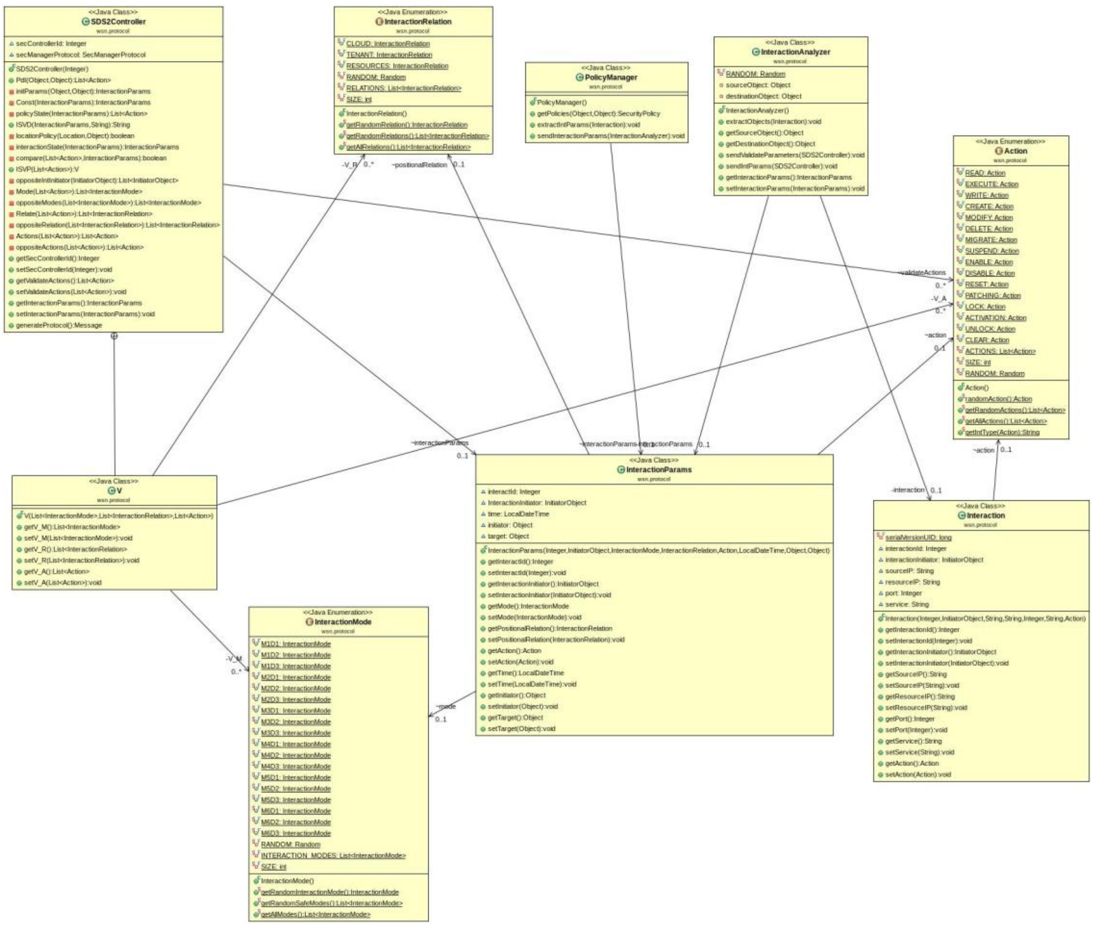


Figure 4.13 Performance monitoring according to interaction detection processing time



<<Java Class>> SDS2Controller wsn.protocol
<<Java Enumeration>> InteractionRelation wsn.protocol
<<Java Class>> PolicyManager wsn.protocol
<<Java Class>> InteractionAnalyzer wsn.protocol
<<Java Enumeration>> Action wsn.protocol
<<Java Class>> V wsn.protocol
<<Java Class>> InteractionParams wsn.protocol
<<Java Class>> Interaction wsn.protocol
<<Java Enumeration>> InteractionMode wsn.protocol

Figure 4.14 Overview of SDS$_2$ interaction classes

4.7 Summary

This chapter has taken a novel approach with the proposed Policy-based Interaction Model to provide isolation within the cloud infrastructure. The proposed model introduced a dynamic construction of security boundaries based on our constructed interaction model and its parameters. An intelligent security algorithm called ISVDP is developed to provide proactive detection and prediction in relation to the interaction parameters to secure cloud resources. Security policy rules pertaining to entities and their location are further applied to the interaction parameters to determine the overall validity of the participating entities interaction. To the best of our knowledge, the policy-driven interaction model is the first in a new direction for combatting security incidents systematically.

Chapter 5

Sec-MANAGE Protocol

5.1 Introduction

In chapter 3, we proposed a software-defined security service (SDS_2) model. The proposed model contains specific virtual security functions (VSFs) to act as an interaction monitoring component within the system. A security controller manages VSFs within the software-defined security network. Clearly, an efficient protocol is required between the controller and the VSFs for management and control purposes. To manage and control these VSFs in accordance with the SDN/NFV principles, the security model required an efficient but light-weight protocol to transfer the interaction values.

Traditional IP networks rely on switches and routers to relay packages according to their routing table entries. Separating flows from others that belong to a specific destination requires a large number of identifying parameters in networking systems. SDN uses OpenFlow protocol for communications between the network controller and the switches. However, the protocol is heavy and rigid due to the nature of routers and switches and the requirements of end-to-end flows. It requires 12 matching parameters to identify a flow and a take a complex set of actions. Clearly, not all these parameters are required by the SDS_2 for communicating security interaction information between the security controller and a VSF. Additionally, VSFs are not endpoint routing devices, and hence complex configuration and routing features are irrelevant.

OpenFlow protocol was introduced to standardize communication between network functions (switches, routers) and their centralized OpenFlow Network controller. Moreover, SDN OpenFlow also needs other protocols like OF-Config to configure networking devices. These protocols combine for managing routing, configuring, and other network functions. OpenFlow and OF-Config are explicitly designed for flow-based SDN switches; they are totally ineffective for resource-constrained devices or specific virtual functions because of their different nature.

This chapter proposes a Sec-Manage protocol that provides a solution for controlling and managing security functions. The Sec-Manage is designed to configure VSFs and control the behavior of the underlying virtual security resources. This chapter investigates the design and implementation of the Sec-Manage protocol. This work has been published in Cyber Security in Networking Conference in Switzerland.

This chapter is organized as follows. Section 4.2 presents a brief description of SDS_2 security service and proposed interaction model (chapter 3 and 4). Section 4.3 presents the proposed Sec-Manage protocol. This section demonstrates the design of the Sec-Manage components and required messages. Section 4.4 presents an evaluation of the proposed Sec-Manage protocol in managing the VSFs. Section 4.5 concludes this chapter.

5.2 Software-Defined Security Service (SDS_2) and Interaction Model

This section provides a brief review and summary of the SDS_2, the security controller's role in managing its VSF network, and the policy-based interaction model.

To reap the benefit of SDN and NFV paradigms, the SDS_2 model is structured in three main layers, the security application layer, the security control layer, and the security data layer. The security application layer contains security applications and interfaces. The developers can deploy their security policies and applications regardless of the knowledge about the underlying security functions through a Northbound API.

The security control layer accommodates the security controller and its components. Its functions interpret security requirements, like security policies, and analyses interaction parameters based on the interaction model. The security controller directs security policy rules, interaction parameters, and instructions to VSFs through the Sec-Manage protocol. The security data layer hosts virtual security functions (Figure 5.1)

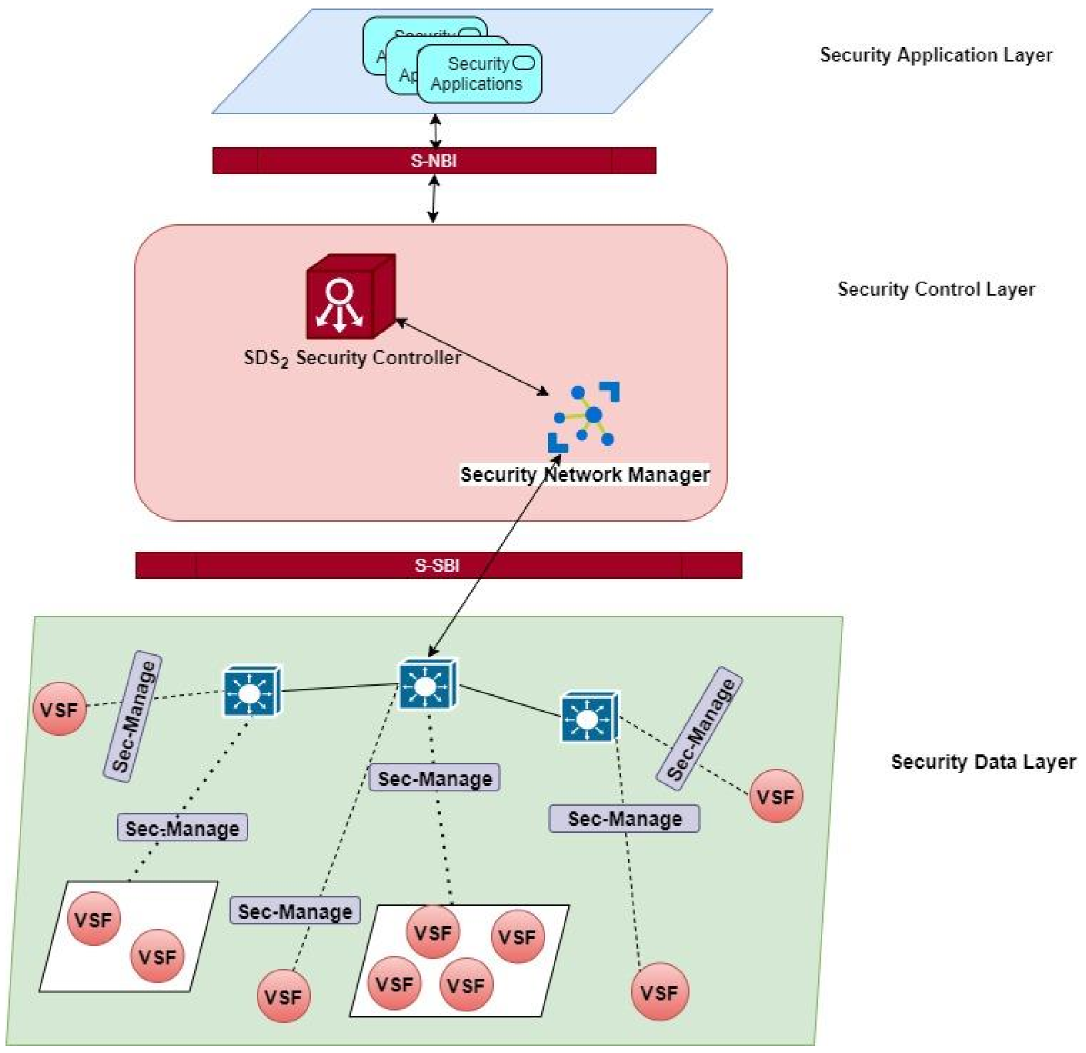


Figure 5.1 SDS$_2$ three layers

5.2.1 SDS$_2$ Security Controller

The SDS$_2$ security controller is responsible for i) processing security interactions, ii) initiating, controlling, and managing virtual security functions, iii) analyzing security interaction parameters to be sent to its VSF, iv) monitoring and initiating interactions security policies. To handle those responsibilities, the SDS$_2$ security controller hosts several essential modules known as VSF Manager, Sec-Net control, Security DB, Security Policy Manager, and Interaction Detection/Prediction Manager.

The security controller initiates a VSF when the task of monitoring a specific interaction is needed. The security controller interprets interaction security requirements through its interaction analyzer function. It then sends instructions and extracted interaction parameters to the assigned VSFs through the Sec-Manage protocol, as depicted in Fig 5.2.

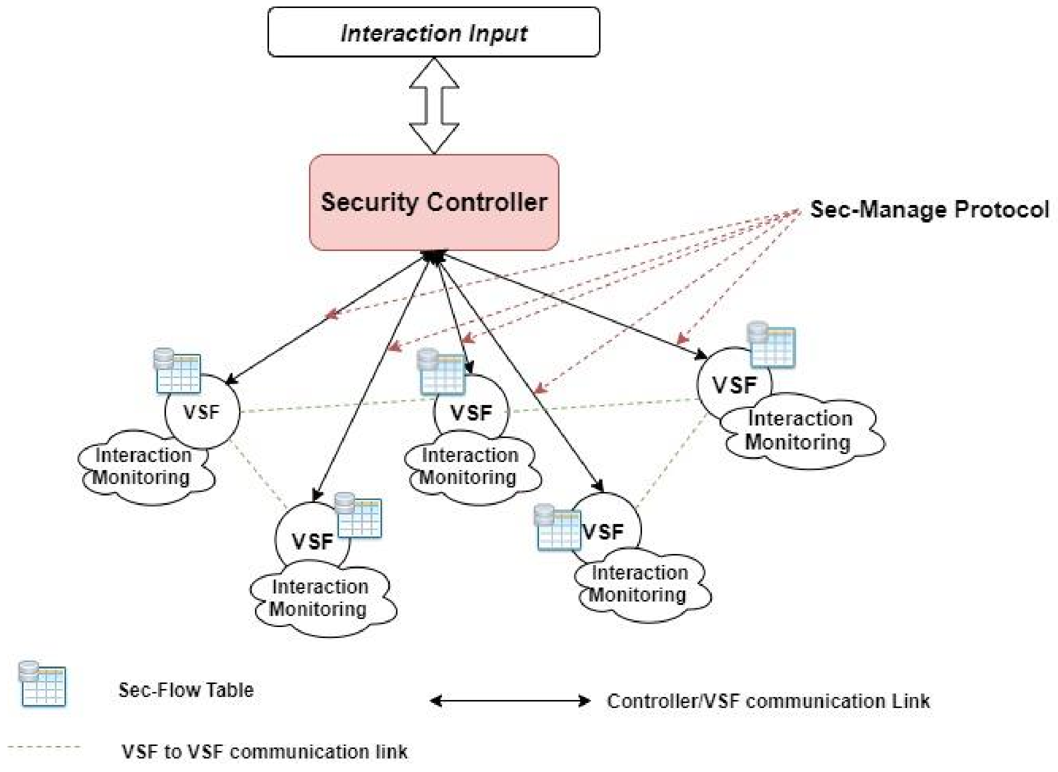


Figure 5.2 SDS$_2$ Security controller and Sec-Manage protocol

5.2.2 SDS$_2$ Policy-based Interaction Model

We proposed an innovative SDS$_2$ Policy-driven Interaction model to detect and predict security breaches in our earlier work. The proposed model was defined based on parameters that control actions amongst entities in cloud infrastructure. The interaction was defined as “an act of performing an action by an object on another.” The model places a particular focus on object interactions as they play a critical role in security incidents within a system. According to the proposed policy-based interaction model, a security violation occurs due to an observed interaction against defined security policies governing the relationship between involved entities.

The SDS_2 proposed the policy-based interaction model as a security defense against interaction violations in the cloud system. The interaction model is governed via security policy expressions.

The interaction model relies on different security constraints to protect cloud infrastructure. It describes how entities interact with one another and relies on an object model of interaction defined with four main parameters: Mode (M)- possible modes between entities for an interaction; Positional Relationship (R)- possible potential relations between entities for an interaction according to role-based level; Action (A)- possible actions between entities over an interaction; and time (t)- the valid time-duration of an interaction.

The Mode consists of two parts, including the n:m mode relationship between entities and the action direction from one object to another. The Positional Relationship regulates an interaction action's validity via rules, roles, layers, and policies applied with entities involved in an interaction. The interaction time refers to an interaction's valid time. The Action is a possible set of actions over an interaction. A set of values defines each parameter. Security policies govern the validity of interaction parameters. The proposed model introduced Interaction Security Violation Detection/ Prediction (ISVDP) algorithms in relation to the defined interaction parameters.

The interaction model and its exclusive parameters enable VSFs to monitor relevant parameters required for security violation detection and prediction algorithms. For this purpose, the SDS_2 security controller requires to communicate with a VSF for security data. The VSF monitors the targeted interaction and uses the Sec-Manage protocol to supply interaction data to the security controller. Figure 5.3 illustrates an overall view of the Sec-Manage functionality between the security controller and VSF. In the section, we detail how to obtain required values related to the interaction model for VSF.

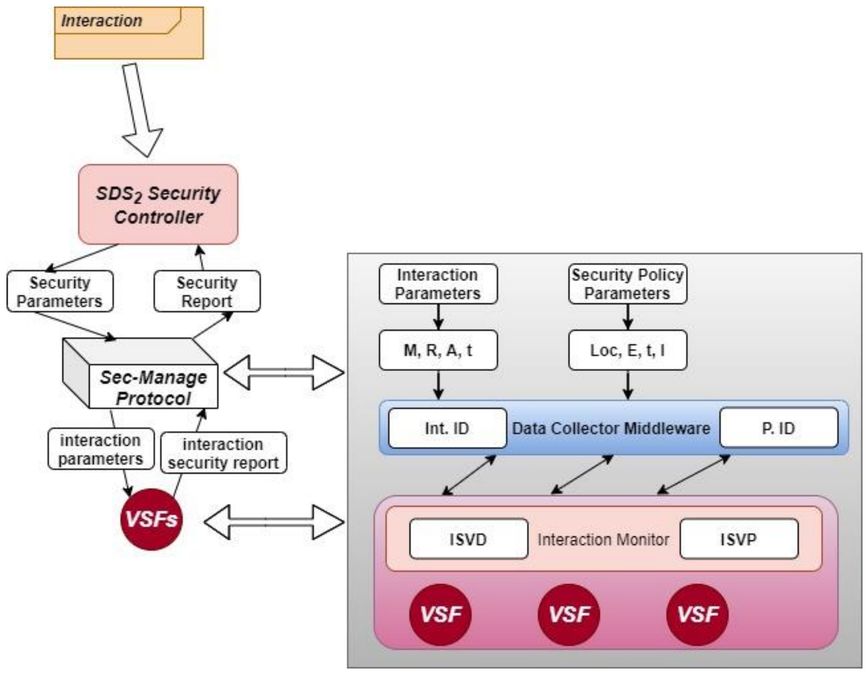


Figure 5.3 Interaction and Sec-Manage Protocol

5.3 Sec-Manage Protocol Design

The Sec-Manage protocol is designed in the style of OpenFlow [3] and OF-Config [10] but for virtual security functions according to our policy-based interaction model for security services. OpenFlow protocol concentrates on flow rules in terms of modifying, deleting, adding, and setting rules to control OpenFlow switches behavior. However, what is required by the SDS$_2$ security model, is not only a streamlined communication protocol that handles both configuration and management of VSFs but also a channel for exchanging parameters-specific to the security policy-based interaction model. The Sec-Manage protocol enables controllers to send the required instructions to profile the VSF behavior dealing with an interaction.

The protocol permits controller to 1) communicate with VSFs; 2) configure the forwarding and config-table entries; 3) send security instructions to the security functions; 4) dynamically program VSFs; 5) get status information of a VSF; 6) collect information regarding the security parameters collected by VSF during an interaction.

The Sec-Manage protocol determines message types between SC-VSF/VSF-VSF, the forwarding and config table structure, and message formats. It also specifies how a VSF reacts and operates over an interaction.

The protocol also allows VSFs to 1) communicate with the security controller; 2) send security warning to SC; 3) send its statistics and state to update security controller VSFs' database; 4) intelligently communicate with other VSFs. 5) update required interaction parameters; 6) send detection and prediction results back to the security controller.

Figure 5.4 demonstrates the connectivity phases between a VSF and the SC. The first step establishes the connection between the security controller and the VSF. Then it sends a hello packet to authenticate the VSF/s. After that phase, the SC configures and manages the VSF until the end of its chain cycle. The details of protocol design and its message types are described in the following section.

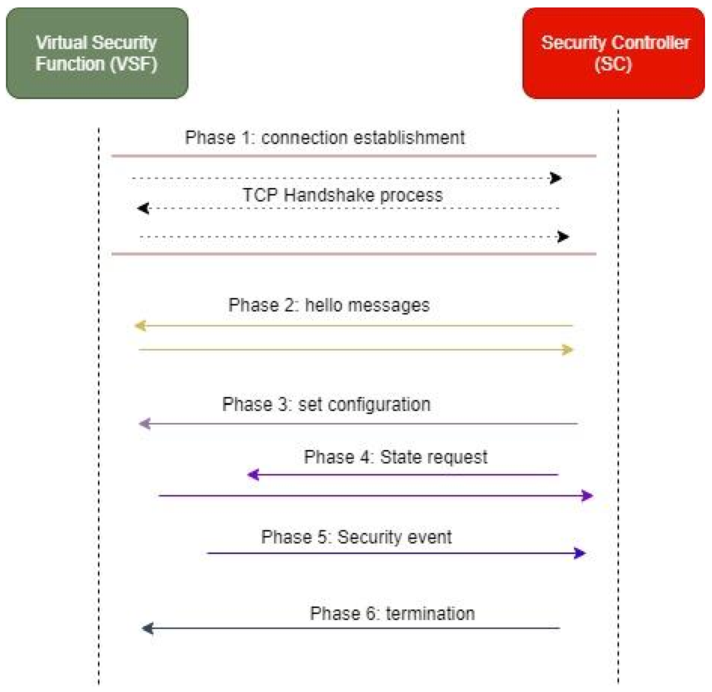


Figure 5.4 Connection establishment

5.3.1 Sec-Manage packet header

The protocol also allows VSFs to 1) communicate with the security controller; 2) send a security warning to SC; 3) send its statistics and state to update security controller VSFs' DB; 4) intelligently communicate with other VSFs. 5) update required interaction parameters; 6) send detection and prediction results back to the security controller. Figure 5.5 shows the Sec-Manage header and payload.

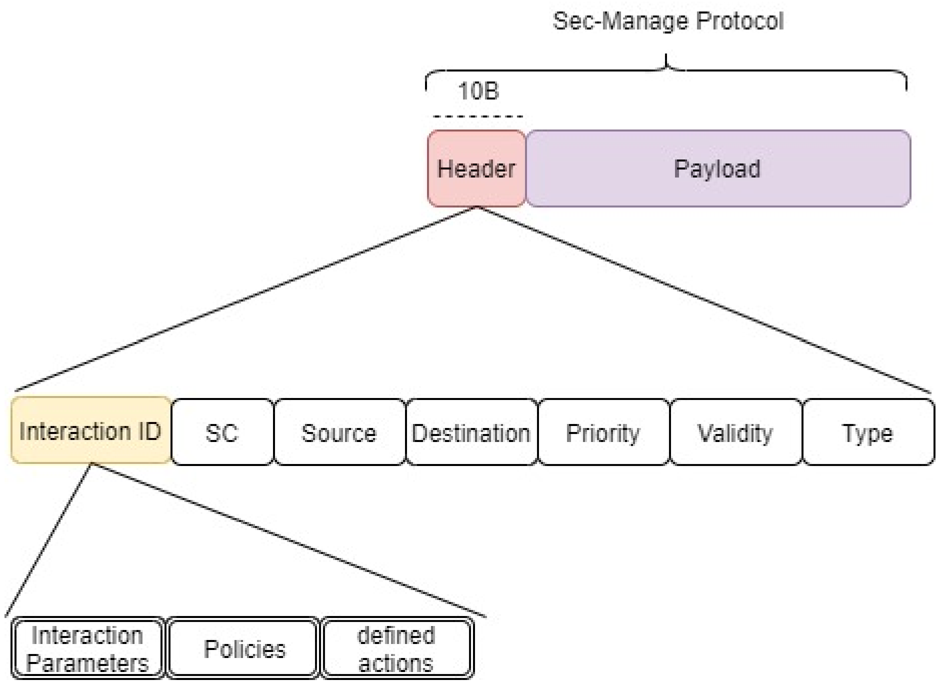
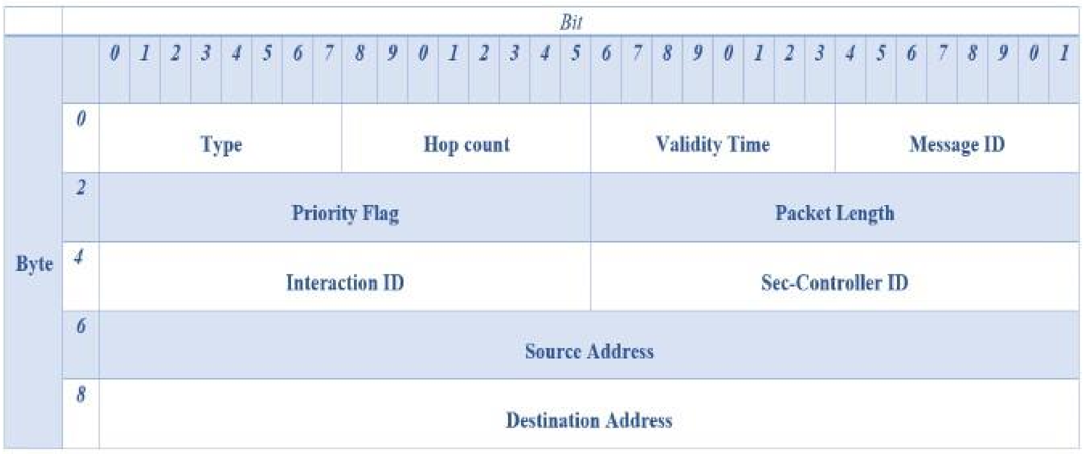


Figure 5.5 Sec-Manage Header and payload

Fig 5.6 demonstrates the structure of the Sec-Manage Header. The packet header consists of:

- Source Address (2 bytes): Shows the address of the source sending a packet
- Destination Address (2 bytes): Shows the destination address of the packet
- Type (1 byte): Specifies the packet type
- Hop count (1 byte): Specifies the number of valid hops that a message can travel within our virtual security network

- Validity Time (1 byte): Demonstrates the valid time in which a packet can exist and travel in the system. If SC or VSF receives the packet with validity time=0, the packet will discharge immediately
- Message-ID (1 byte): Presents unique ID provided to each message which identifies the message type
- Priority Flag (2 bytes): Shows the priority of message in terms of receiving instruction or performing actions
- Packet length (2 bytes): Specifies the total size of the packet, including both header and payload
- Interaction ID (2 bytes): Refers to the ID of the requested interaction
- Sec-Controller ID (2 bytes): Indicates the ID of the security controller that initiates the VSF.



	Bit			
Byte	0 1 2 3 4 5 6 7	8 9 0 1 2 3 4 5	6 7 8 9 0 1 2 3	4 5 6 7 8 9 0 1
0	Type	Hop count	Validity Time	Message ID
2	Priority Flag		Packet Length	
4	Interaction ID		Sec-Controller ID	
6	Source Address			
8	Destination Address			

Figure 5.6 Sec-Manage Protocol Header

5.3.2 Message types

The different types of messages are transferred within the payload. Each message carries different information between the SC and its VSFs. However, to achieve our goal, we categorize three main messages as required to be sent via Sec-Manage protocol: i) Security Controller (SC) to VSF- messages direct from SC to security functions; ii) VSF to Security Controller- messages sent from security functions to the controller; iii) VSF to another VSF- messages transfer between security functions in same/different domain.

The proposed protocol grouped the message types into three categories 1) SC-to-VSF; 2) Symmetric message (VSF/SC, SC/VSF); 3) Asynchronous message (VSFs-to-SC). The security model minimized the number of messages transferred due to security agility reaction to security incidents (Figure 5.7).

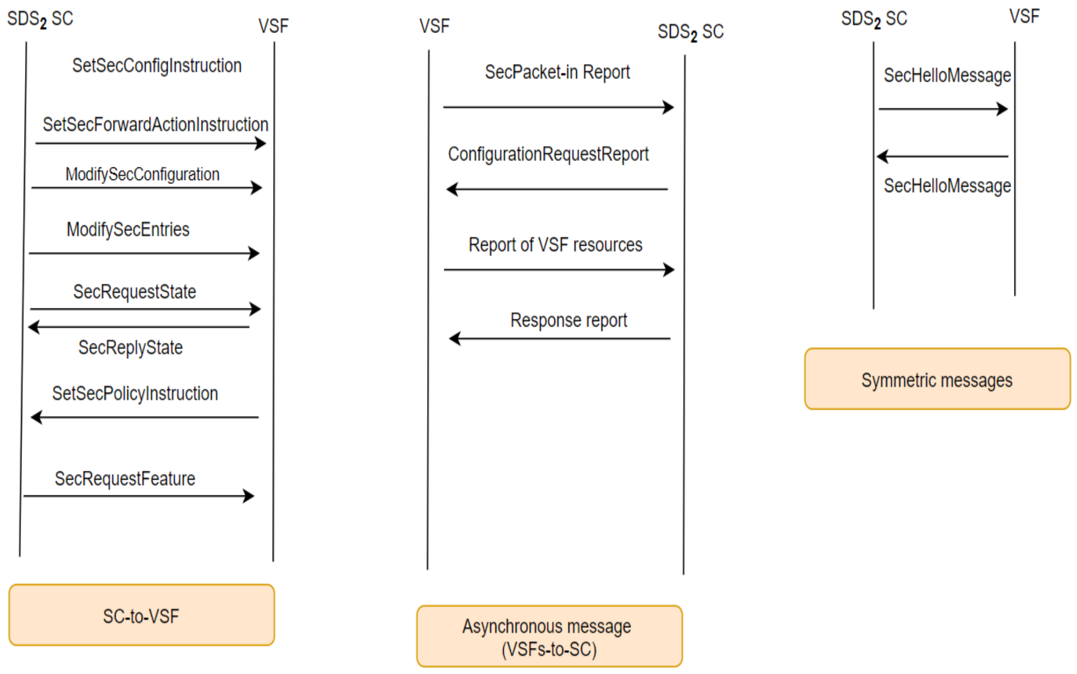


Figure 5.7 Sec-Manage message types

❖ SC-to-VSF messages

The security controller triggers these types of messages. The messages can be considered as send/receive or send only messages. In send/receive, the security controller expects the bulk of information to be sent back from the VSF, while in sent only, the security controller does not expect a response. These messages used to be sent to configure and manage VSFs. Required messages to be exchanged between SC-VSF are defined as follows:

- SetSecConfigInstruction/SetSecForwardActionInstruction: It enables the security controller to query and set configuring/forwarding parameters on VSF. The VSF is expected to send a reply only in case of requesting a configuration from the security controller
- ModifySecConfiguration: The security controller can modify VSF configuration values
- ModifySecEntries: The security controller can modify entries parameters within the VSF
- SecRequestState: The security controller request to collect statistics on VSF status and its functions
- SecReplyState: The VSF sends this message to the controller as a response to SecRequestState
- SetSecPolicyInstruction: The message is for the security controller to set/update policies related to VSF
- SecRequestFeature: The security controller collects VSF information related to its action, interaction prediction parameters, VSF's status, VSF's connectivity.

❖ **Symmetric message (VSF/SC, SC/VSF)**

These types of messages will be initiated by either the security controller or the VSF during a connection setup without solicitation. The message is sent during the lifetime of VSF for notification purposes.

- SecHelloMessage: The hello messages are exchanged between the security controller and VSFs to verify their liveness during a periodical time slot.
- SecErrorMessage: Sent from VSFs to the security controller in case of fail request configuration or security function malfunctioning.

❖ **Asynchronous message (VSFs-to-SC)**

VSF initiates these types of messages to inform the security controller in case of any update or event/state changes. A VSF sends these messages without a security controller soliciting them. It enables the VSF to report changes regarding object interaction and the monitoring state back to the security controller.

SecPacket-in Report: Reports on the status of VSF and its behavior in handling the interaction. It updates the security controller regarding applied changes.

- Inform the security controller on interaction status
- Update the security controller on its RequestedFeature Report changes
- Notify security controller on the discovery of any types of entry confliction
- Inform the security controller about the completion interaction monitoring process
- Alert the security controller in case of malicious modification
- Update security controller in case of dynamic changes of its actions related to an interaction inquiry
- Update the security controller on predicted interaction parameters

ConfigurationRequestReport: It enables a VSF to request handling an operation that does not match its instructions. The VSF informs the security controller regarding upcoming interactions with no existing instruction to be handle.

5.3.3 Forwarding Interaction Table Specification

The table entails three main components: Interaction Matching, Instruction, and Counters, as presented in Figure 5.8. The interaction matching window is matched against the upcoming interaction field in the Sec-Manage header field. Corresponding actions in the instruction window will take place in case of a match. The counter window describes the statistics of matching entry.

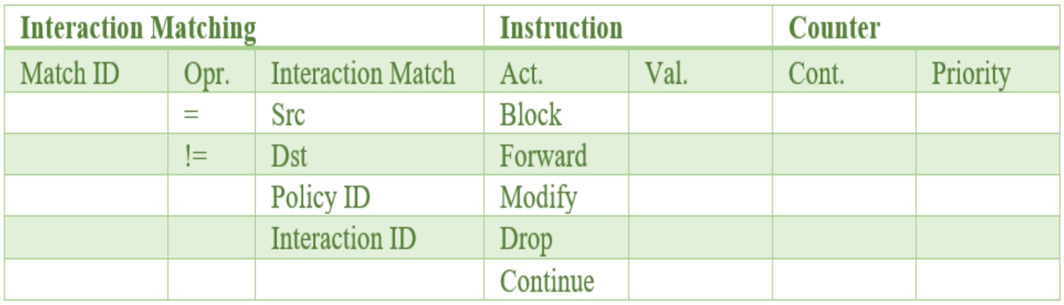
1. Interaction matching: Provides matching information extracted from arriving interaction packet header. The window is comprising of three parameters.

- Match ID: Specifies the ID of matching interaction. It matches an incoming interaction packet with a matching field in the header.

- Interaction Match: Indicates which field of header requires to be matched in the interaction matching window. So, not all header fields are required to be matched, and less matching process is performed to find the match.
- Opr: Indicates a comparison between the matching header and interaction matching value. The values can be considered as equal (=), and different (!=).

2. Instruction: Specifies the corresponding action to be applied to interaction entries. It is composed of two parameters.

- Action (Act.): provides types of action to be executed according to matching interaction. In this table, we defined types of action to be considered such as Forward, Modify, Block, Drop, continue [11]; Continue: continue the interaction process.
- Value (Val.): Indicates the value of an action.



Interaction Matching			Instruction		Counter	
Match ID	Opr.	Interaction Match	Act.	Val.	Cont.	Priority
	=	Src	Block			
	!=	Dst	Forward			
		Policy ID	Modify			
		Interaction ID	Drop			
			Continue			

Figure 5.8 Interaction Forward table structure

3. Counter: Collects the data and statistics in terms of updates/any changes in interaction entries and matched entries.

- Count: Reveals a number of interactions matched against table entries. Counters are maintained for interaction entries and statistics parameters.
- Priority: Indicates the priority of matching entries and their action. It is used in case of multiple actions which are possible related to entries.

5.3.4 Config Interaction Table Specification

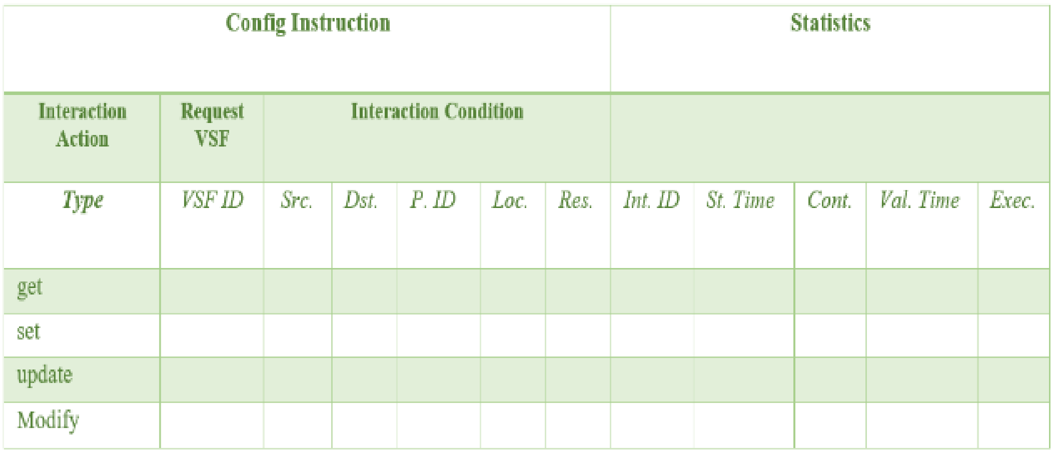
The table entries provide instructions on how a VSF handles an interaction. The table structure consists of two main components: Config Instruction and Statistics (presented in Figure 5.9).

1. **Config Instruction:** Includes three main components: Interaction Action, Request VSF, and Interaction Conditions.

- Type: Indicates valid action/s type applied to an interaction entry in the table.
- VSF ID: Specifies the VSF/s connected with the interaction entry.
- Interaction Condition: Indicates matching conditions applied to the interaction entry.
- Src: Specifies the source object in the interaction and related constraints.
- Dest: Specifies the destination object in the interaction and related constraints.
- P.ID: Directs to policy parameters assigned to interaction entry (local/general policies).
- Loc: Indicates conditions related to the location of the source, destination, and interaction.
- Res: Specifics conditions related to resource/s involved in an interaction.

2. **Statistics:** Provides statistics data on the interaction entry. The statistics will be updated after a match discovery.

- Int.ID: Indicates interaction/s assigned to a VSF.
- Start time (St. Time): Shows the exact starting time of interaction.
- Counter: Shows the number of changes within a specific time.
- Validate time (Val. Time): Specifies the valid time of the target interaction, defined by the SC according to the applied policies. When it is 0 the interaction entry is expired.
- Executed: Specifies if an action has been executed.



Config Instruction							Statistics				
Interaction Action	Request VSF	Interaction Condition									
Type	*VSF ID*	*Src.*	*Dst.*	*P. ID*	*Loc.*	*Res.*	*Int. ID*	*St. Time*	*Cont.*	*Val. Time*	*Exec.*
get											
set											
update											
Modify											

Figure 5.9 Interaction Config Table

- Config table in software

Establishing a config interaction table required several classes displayed in figure 5.10. The displayed classes are forming an entry table within the config interaction table.

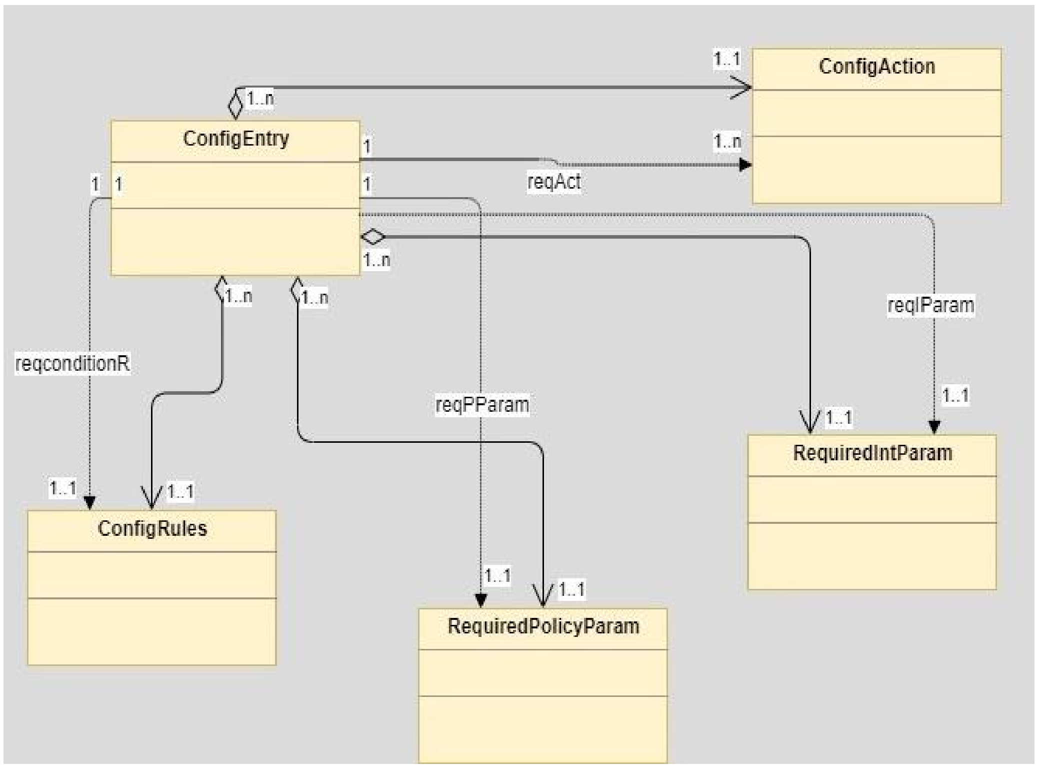


Figure 5.10 Interaction configuration table class of diagram

5.4 Implementation and Performance Evaluation

5.4.1 Implementation Set up

To demonstrate the proposed Sec-Manage protocol's performance, we develop our environment where our Sec-Manage protocol can manage and control our underlying interaction-based virtual security functions. We deployed a testbed using two Ubuntu 14.04 VM running on top of the OpenStack cloud platform. A security controller is software written in Java. The whole environment is running on top of the VMware virtual environment.

Specifically, this section demonstrates 1) the feasibility of Sec-Manage as an effective communication protocol between the SC and its VSFs in a virtual security network; 2) the capability of Sec-Manage in supporting our policy-based interaction protocol for security violation detection and prediction. The results demonstrate how the controller configures its virtual security function using Sec-Manage protocol using the VSF's forwarding and config tables.

The results also demonstrate that the protocol allows the security controller to direct its virtual security functions to monitor targeted object interaction and collect interaction data for the security violation and detection algorithms. Our implementation prototype is shown in Figure 5.10. To test the proposed protocol, a number of interactions are triggered to access a specific resource. The security controller is to look into this interaction request. Figure 5.11 demonstrates the process of connection between Client and Controller.

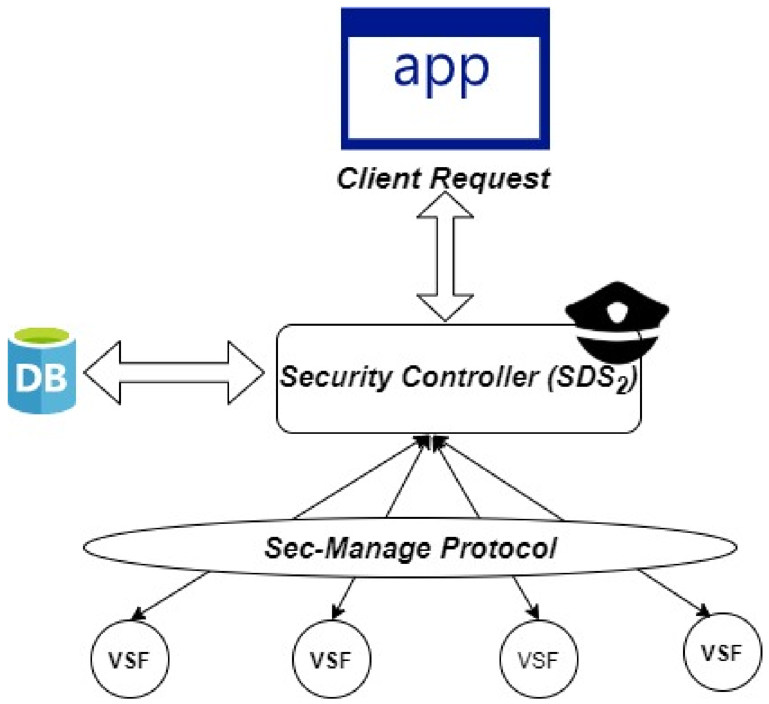


Figure 5.11 The implementation of the prototype

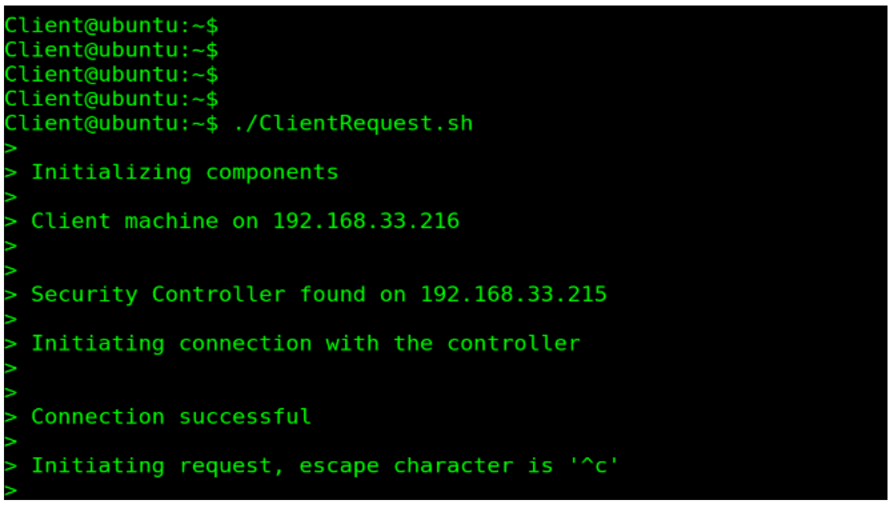
The software modules consisting of security controller, southbound interface, virtual security representation in charge of the SDS_2 security controller, the Sec-Manage protocol, and VSFs. The security controller includes classes responsible for extracting required interactions and policy parameters, orchestrating underlying interaction-based virtual security functions, networking, and communication between the security controller and VSFs. The southbound interface module consists of classes for our proposed Sec-Manage protocol, including messages, interaction forwarding, and config tables. The virtual security function modules consist of java classes representing the VSF.

A database was established as a built-in DB within the security controller to store and update information regarding VSFs locations, interaction and policy parameters, status, and attributes of VSF.

5.4.2 Performance Evaluation

According to the triggered interaction and its domain/ location, the security controller initiates a VSF. It then sends SetSecConfigInstruction / SetSecForwardActionInstruction message to VSF via Sec-Manage. The requested interaction contains a set of parameters. The request parameters indicate the required resources, identification of interaction (Int. ID), interaction actions (ACT.), source ID, started time, security controller ID (Sec-ID).

Targeted interaction contains potential interaction parameters and user privileges. The security controller then extracts the targeted interaction according to its tangled resources and analyze required security policies. The Sec-Manage protocol transfer interaction parameters through the Int.ID that includes essential interaction parameters. It also transfers the required policy parameters by the P.ID parameter. The Sec-Manage enables VSFs to report back the results of interaction security violation monitoring. The following figures demonstrate the forwarding table, config table, and security controller service table. The security controller service table shows the list of VSFs and their status during and after an interaction. Figure 5.12 shows a client request script to be sent to the controller.

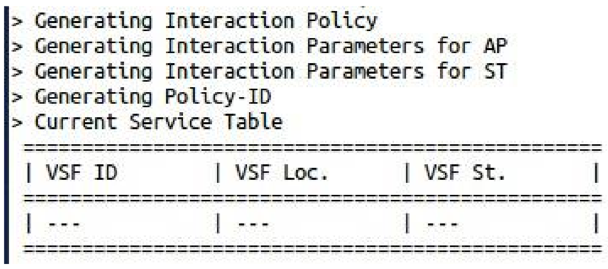


```
Client@ubuntu:~$
Client@ubuntu:~$
Client@ubuntu:~$
Client@ubuntu:~$
Client@ubuntu:~$ ./ClientRequest.sh
>
> Initializing components
>
> Client machine on 192.168.33.216
>
>
> Security Controller found on 192.168.33.215
>
> Initiating connection with the controller
>
>
> Connection successful
>
> Initiating request, escape character is '^c'
>
```

Figure 5.12 Client sending request to the security controller

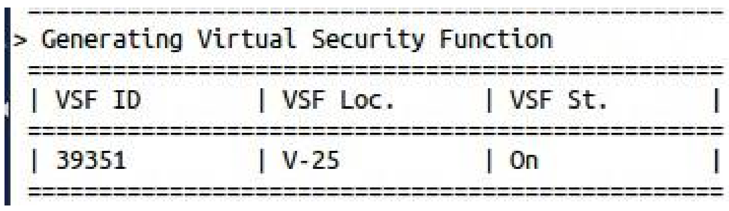
The figures follow two main scenarios: 1) indicates the tables before configuration 2) demonstrates tables record after configuration. The security controller uses a random number allocation for its functions to record their activities.

Figure 5.13 and 5.14 illustrate the security controller service table before and after initiating a VSF. For the first time that the security controller receives an interaction, the table has no record store inside.



```
> Generating Interaction Policy
> Generating Interaction Parameters for AP
> Generating Interaction Parameters for ST
> Generating Policy-ID
> Current Service Table
 ===============================================
 | VSF ID        | VSF Loc.      | VSF St.      |
 ===============================================
 | ---           | ---           | ---          |
 ===============================================
```

Figure 5.13 Current Service Table

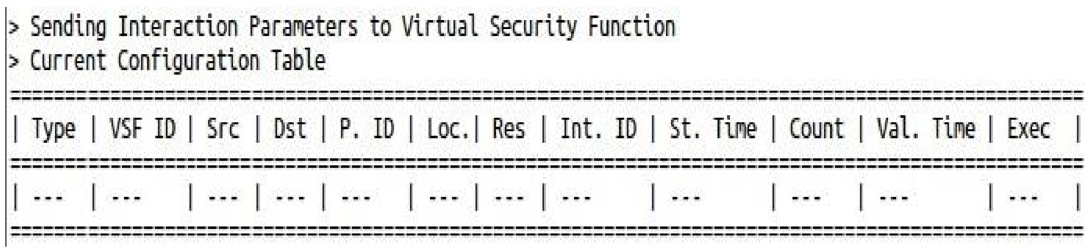


```
 ===============================================
> Generating Virtual Security Function
 ===============================================
 | VSF ID        | VSF Loc.      | VSF St.      |
 ===============================================
 | 39351         | V-25          | On           |
 ===============================================
```

Figure 5.14 Security controller service table after receiving an interaction

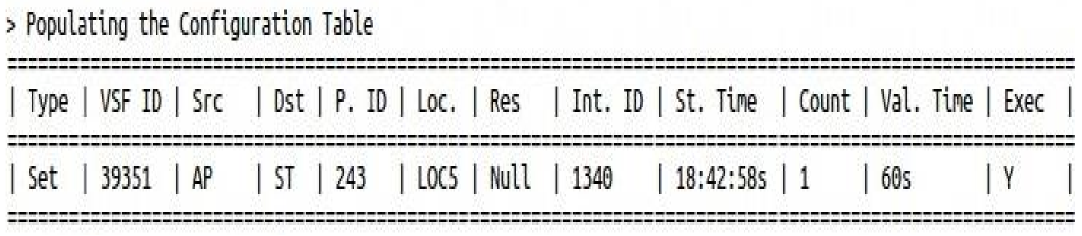
The VSF ID is a unique identifier assigned to each VSF. The VSF Loc shows the location that VSF is placed to monitor the targeted interaction. The VSF St. displays the current status of the security function within the system. The VSF status can be on, off, expire, idle, or delete.

After analyzing the interaction and its involved entities, the config table will be configured accordingly. Figure 5.15 and 5.17 demonstrate results before VSF configuration. Figure 5.15 and 5.16 demonstrate the Config table before and after the configuration.



```
> Sending Interaction Parameters to Virtual Security Function
> Current Configuration Table
==========================================================================================
| Type | VSF ID | Src | Dst | P. ID | Loc.| Res | Int. ID | St. Time | Count | Val. Time | Exec  |
==========================================================================================
| ---  | ---    | --- | --- | ---   | --- | --- | ---     | ---      | ---   | ---       | ---   |
==========================================================================================
```

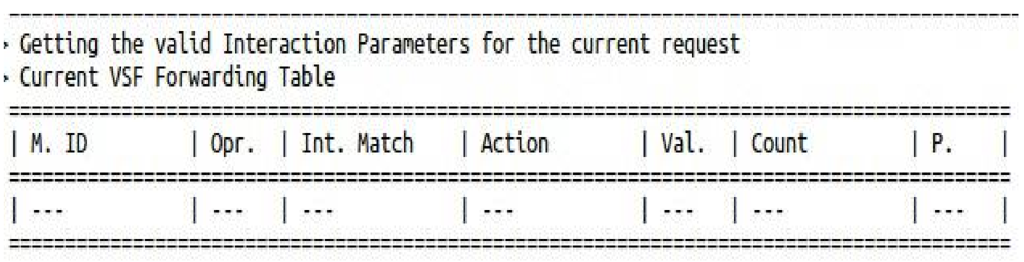
Figure 5.15 VSF Config Table before



```
> Populating the Configuration Table
==========================================================================================
| Type | VSF ID | Src  | Dst | P. ID | Loc. | Res  | Int. ID | St. Time  | Count | Val. Time | Exec |
==========================================================================================
| Set  | 39351  | AP   | ST  | 243   | LOC5 | Null | 1340    | 18:42:58s | 1     | 60s       | Y    |
==========================================================================================
```

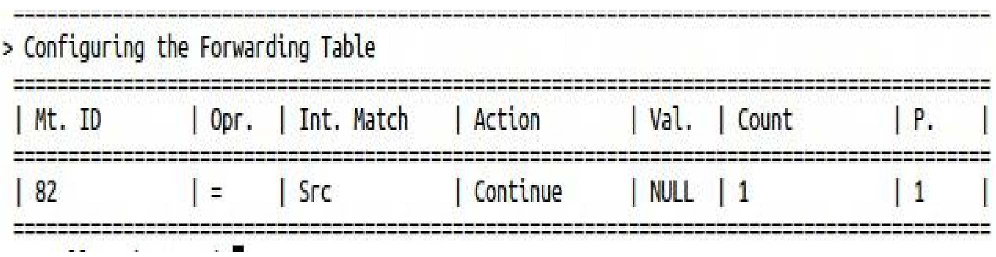
Figure 5.16 VSF Config table after configuration

Figure 5.17 and 5.18 exhibited the VSF forwarding table before and after the configuration.



```
-------------------------------------------------------------------------------------
> Getting the valid Interaction Parameters for the current request
> Current VSF Forwarding Table
=====================================================================================
| M. ID        | Opr. | Int. Match  | Action       | Val. | Count        | P.  |
=====================================================================================
| ---          | ---  | ---         | ---          | ---  | ---          | --- |
=====================================================================================
```

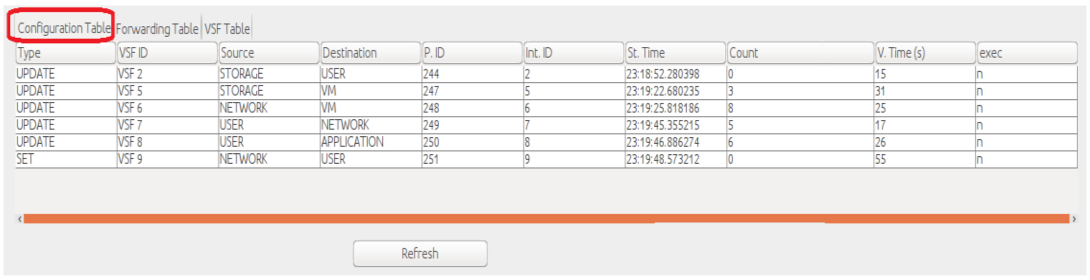
Figure 5.17 Current VSF forwarding table



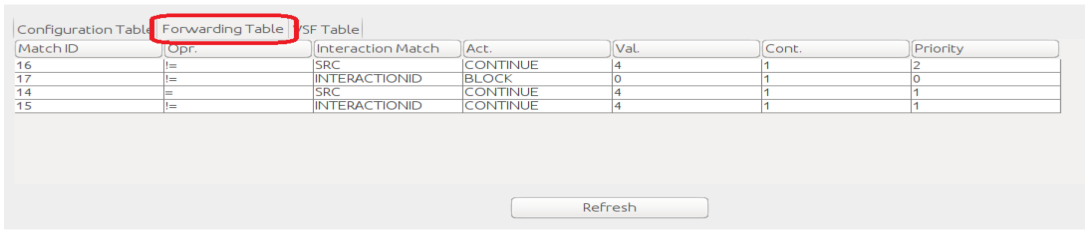
```
-------------------------------------------------------------------------------------
> Configuring the Forwarding Table
=====================================================================================
| Mt. ID       | Opr. | Int. Match  | Action       | Val. | Count        | P.  |
=====================================================================================
| 82           | =    | Src         | Continue     | NULL | 1            | 1   |
=====================================================================================
```

Figure 5.18 VSF Forwarding table after configuration

Through Sec-Manage, the SDS_2 security controller can handle multiple interactions and orchestrate VSFs to process the requests. Figure 5.19 demonstrates the configuration and forwarding table from the Device-level side. It shows the configuration of VSF in relation to various triggered interactions within the system.



Type	VSF ID	Source	Destination	P. ID	Int. ID	St. Time	Count	V. Time (s)	exec
UPDATE	VSF 2	STORAGE	USER	244	2	23:18:52.280398	0	15	n
UPDATE	VSF 5	STORAGE	VM	247	5	23:19:22.680235	3	31	n
UPDATE	VSF 6	NETWORK	VM	248	6	23:19:25.818186	8	25	n
UPDATE	VSF 7	USER	NETWORK	249	7	23:19:45.355215	5	17	n
UPDATE	VSF 8	USER	APPLICATION	250	8	23:19:46.886274	6	26	n
SET	VSF 9	NETWORK	USER	251	9	23:19:48.573212	0	55	n



Match ID	Opr.	Interaction Match	Act.	Val.	Cont.	Priority
16	!=	SRC	CONTINUE	4	1	2
17	!=	INTERACTIONID	BLOCK	0	1	0
14	=	SRC	CONTINUE	4	1	1
15	!=	INTERACTIONID	CONTINUE	4	1	1

Figure 5.19 SDS$_2$ GUI interface demonstrating multiple interaction forwarding and configuration records

5.5 Summary

This chapter proposed designing and implementing the Sec-Manage protocol to address challenges in providing an effective communication channel between the SC and VSFs for security violation and prediction based on the policy-based interaction model. The Sec-Manage protocol provides streamlined and effective communication between the security controller and its virtual security function/s in the underlying virtual security network. It is also tailored to our proposed policy-based interaction model to collect and transfer the interaction parameters to the software-defined security controller to manage, detect, and predict security violations in the cloud system. The Sec-Manage protocol enables the SDS$_2$ controller to control the VSFs dynamically. The proposed protocol paves the way for further research and deployment on orchestration and deployment of virtual security function and interaction security violation monitoring and prediction.

Chapter 6

Software-Defined Security Service Architecture and Components

6.1 Introduction

Cloud computing has become an alternative IT infrastructure where users, infrastructure providers, and service providers all share and deploy resources for their business processes and applications. Business customers are shifting their services and applications to cloud computing since they do not need to invest in their own costly IT infrastructure but can delegate and deploy their services effectively to cloud vendors and service providers [137]. In parallel to cloud computing, the software-defined networking (SDN) paradigm has enabled the automation of virtual networks and network management with centralized control. Network functions virtualization (NFV) pushes the concept even further by allowing virtualization (software implementation) of network functions, traditionally realized by hardware, and deploying them on commodity computing devices. Specifically, the SDN principle decouples the network control plane from its data plane. The SDN enables providers with the capability of provisioning automatic and on-demand network services through a programmable and logically centralized network controller.

In this chapter, we introduce our software-defined security architecture. The chapter describes the main component of SDS$_2$ architecture, the SDS$_2$ security controller. This chapter mainly concentrates on demonstrating SDS$_2$ architecture working alongside its main components via the SDS$_2$ security controller. The SDS$_2$ architecture provides functionalities for SDS$_2$ service to be performed in the cloud environment to protect virtual resources. The architecture enables automation, programmability, and on-demand underlying virtual security functions. The Policy-based Interaction model to protect the cloud resources in an innovative method has been discussed in chapter 4. The architecture introduces the core components and enables management and orchestration of VSFs by Sec-Manage protocol, discussed in chapter 5. The architecture has been published in [37, 134]. The architecture consists of different main components and three main layers as displayed in Figure 6.1.

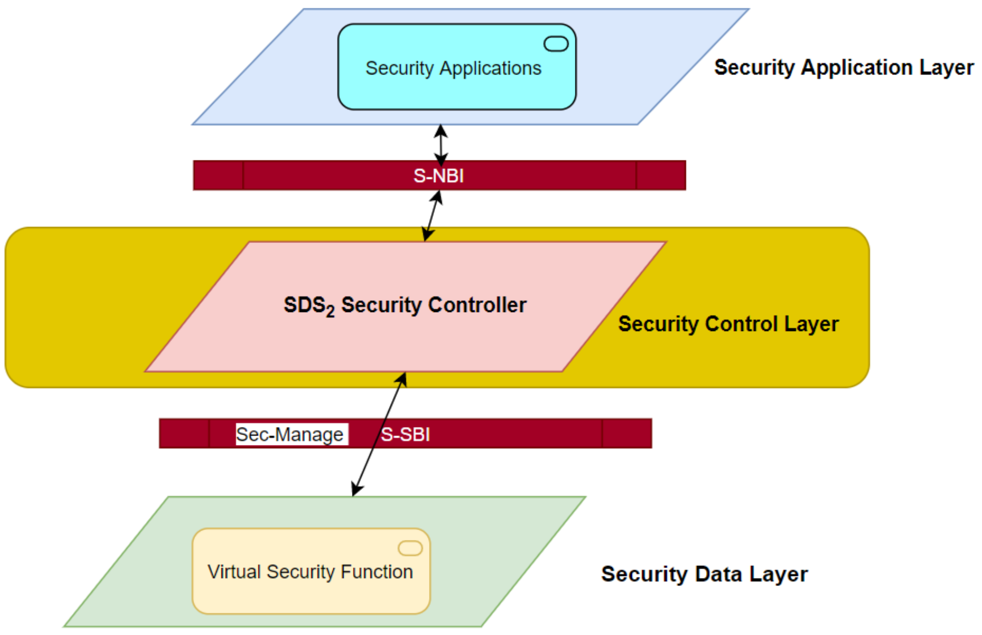


Figure 6.1 SDS$_2$ Layers

The remainder of this chapter is organized as follows. Section 6.2 describes the overall SDS$_2$ architecture. Section 6.3 presents the structure and functionalities of the SDS$_2$ security controller. Section 6.4 describes the software implementation of the

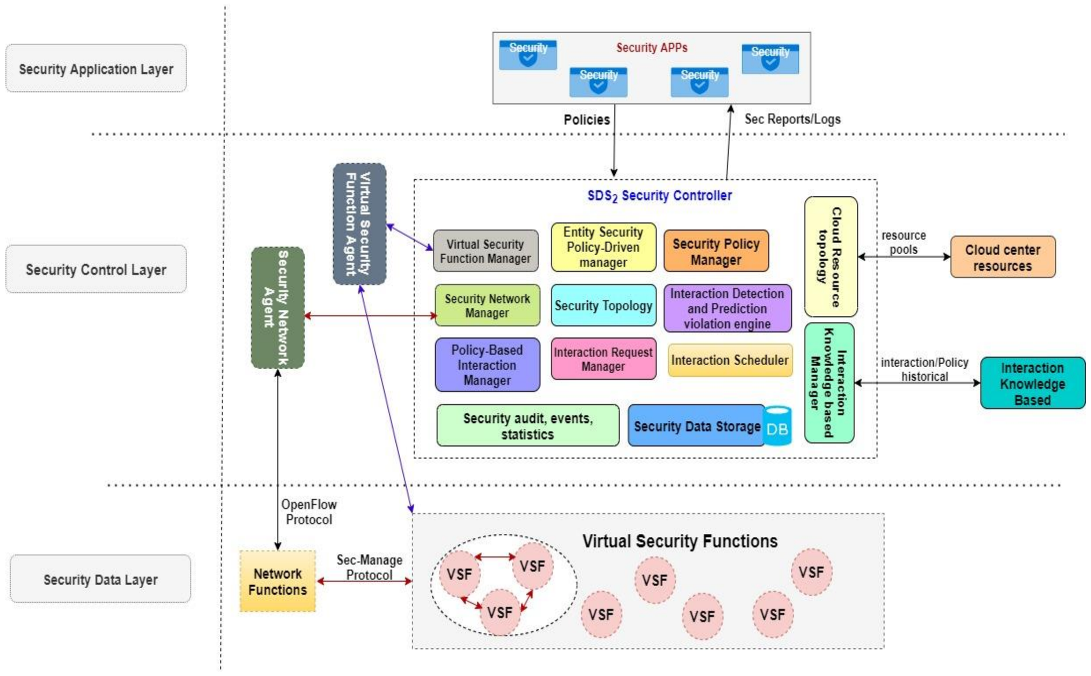
SDS_2 controller. Section 6.5 demonstrates an implementation scenario. Section 6.6 demonstrates the working of the SDS_2 security controller as the main component of SDS_2 architecture. According to the interaction model, this section demonstrates performance evaluation related to the SDS_2 security controller's ability to orchestrate virtual security functions. Section 6.7 concludes this chapter.

6.2 Software-Defined Security Service (SDS_2) Architecture

Reflecting on the model and service aspects discussed in the earlier section, we propose a Software-Defined Security Service architecture (SDS_2) to support SDSec Service that can be deployed by cloud providers to protect its integrated cloud infrastructure. The proposed architecture embraces the SDN and NFV principles. To reap the benefits of SDN and NFV paradigms, the SDS_2 architecture is structured in three security layers: Security Application Layer, Security Control Layer, and Security Infrastructure/data Layer. The proposed SDS_2 architecture is shown in figure 6.2.

The SDS_2 deploys the very virtual resources of the cloud to provide its protection service. It behaves like a trusted tenant overseeing and providing the security service for the cloud infrastructure. SDS_2 belongs to the new software-defined approach that manages security by separating the security forwarding and processing plane from the security control plane. The SDS_2 utilizes concepts and techniques of cloud, SDN, and NFV.

Applying the NFV concepts for security, virtualization technologies are used to implement virtual security functions (VSFs) on a VM or industry-standard commodity hardware. These virtual security functions can be created on-demand and moved to or instantiated in strategic locations in a software-defined dynamic virtual network environment. Applying the SDN concepts for security, network virtualization is deployed to provision virtual security networks (VSNs) connecting virtual security functions. A logically centralized SDSec controller forms a domain-wide view of the underlying network of virtual security functions. The SDSec controller can program, configure, and control the VSFs autonomously. Applying cloud computing concepts for security, physical storage, network, and computing resources are virtualized to accommodate virtual network functions, virtual security networks, and virtual security storage. The cloud platform is used for orchestrating the provisioned security components to provide security services for the target cloud infrastructure.



Security Application Layer
Security Control Layer
Security Data Layer
Security APPs
Security
Policies
Sec Reports/Logs
SDS2 Security Controller
Virtual Security Function Agent
Security Network Agent
Virtual Security Function Manager
Entity Security Policy-Driven manager
Security Policy Manager
Cloud Resource topology
Security Network Manager
Security Topology
Interaction Detection and Prediction violation engine
Policy-Based Interaction Manager
Interaction Request Manager
Interaction Scheduler
Interaction Knowledge based Manager
Security audit, events, statistics
Security Data Storage
DB
resource pools
Cloud center resources
interaction/Policy historical
Interaction Knowledge Based
OpenFlow Protocol
Network Functions
Sec-Manage Protocol
Virtual Security Functions
VSF

Figure 6.2 SDS$_2$ Architecture overview

The proposed security architecture decouples security functions and security networks from the underlying infrastructure. The software-defined security architecture centres around an *Interaction model* among objects to detect and predict security breaches. It relies on the proposed *Security Policy-Based Interaction Model,* which utilizes virtual security functions (VSFs) within the software-defined security network. The security architecture consists of three main components: Security Controller, Virtual Security Functions (VSF), and a Sec-Manage Protocol. In this section, we discuss the main functions in detail. The SDS$_2$ deploys the very virtual resources of the cloud to provide its protection service. It behaves like a trusted tenant overseeing and providing the security service for the cloud infrastructure. SDS_2 belongs to the new software-defined approach that manages security by separating the security forwarding and processing plane from the security control plane.

Figure 6.2 is the overview of SDS_2 architecture. It comprises three main planes: the security application layer, the security control layer, and the security infrastructure/data plane. At the top of security architecture, there are a diversity of security APPs developed by third parties. The middle of the architecture consists of a security controller and its main modules. The architecture's bottom is the security data plane, including virtual security functions and their connectivity through Sec-Manage protocol. An implementation of a platform that deploys the architecture is detailed in chapter 7.

With the proposed SDS_2 architecture, we address four significant contributions: i) providing automation, orchestration, and configuration of virtual security functions in a cloud infrastructure, ii) programming on-demand virtual security functions, iii) enabling intelligent, proactive security system protecting cloud resources, iv) deploying intelligent, dynamic and on-demand security boundaries for cloud resources according to cloud’s entities interaction model. An SDS_2 security architecture is composed of three main layers/planes.

Security Application Layer Security applications are found in this layer. This layer communicates with the SDS_2 security control layer through an intent-based security northbound interface that allows applications/orchestrators to express their required security services regarding their application-specific requirements, including security policies rather than their systems' structure/services.

Security Control layer This layer includes an SDS_2 security controller, which has a complete view over its VSFs and their interconnectivity within the cloud infrastructure. It is essential to differentiate the SDS_2 security controller from SDN controller where security controller deals only with security services, security functions, and their private virtual security network. In another word, SDN controller construct dynamic connectivity among virtual network functions while SDS_2 security controller mainly concentrates on providing on-demand security services for cloud resources protection.

The SDN controller deals with creation of dynamic networking for underlying network functions. The security controller consists of various components, and each is responsible for a specific task. Since the SDS_2 controller has a global view over the system, it can construct appropriate responses to security incidents in real-time.

Security Data Layer The virtual security functions (VSFs) are placed in this layer and communicate with the SDS_2 controller through a specific protocol. A simple protocol has been designed to program, configure, and manage VSFs and allow them to report their operational status to the controller. It includes the intelligent and dynamic networking communication mechanism and the link between the controller and VSFs and even among VSFs. It should be noted that VSFs are not switches or routers; they only perform their defined security functions and relay their data/status to their controller and other VSFs when directed, such as in chaining operations.

6.2.1 Virtual Security Function (VSF)

The VSF is a security element or function implemented in software and deployed on a virtual resource such as a VM in a physical server (host). It is a generalization of NFV VF that abstracts a physical security appliance and is deployed on a commodity server.

A VSF is created to perform a specific security function. It is a software object that can be created, instantiated, and operated on any VM. A VSF is a software entity with a life cycle starting from the instant when it is created through its operation and its termination. A service chaining function can chain VSFs to create a new security function. It can also be combined with others to create complex security functions. Typical VSFs include firewalls, virus scanners, intrusion detection systems, security gateways, and in-depth packet inspections. Other functions include policy/rule checkers, and security metric meters. In the proposed SDS_2 architecture, the primary function of a VSF is to intelligently monitor the requested interaction according to proposed interaction parameters (depicted in Chapter 4).

6.2.2 Sec-Manage Protocol

The key purpose of the proposed protocol between SDS_2 Security Controller and VSFs is to transfer security messages and interaction parameters between a security controller and its VSFs. The Sec-Manage protocol is proposed as a bridge between the security controller (SC) and the VSFs to transfer interaction values according to the SDS_2 Interaction Model. The main aims of designing the Sec-Manage protocol are 1) to provide direct communication between the SDS_2 security controller and its VSFs and 2) to transfer the parameters pertinent to the security aspects of objects' interaction between a VSF and the SC to monitor parameters of an interaction to detect and predict security violations. The proposed Sec-Manage protocol is described in chapter 5.

6.2.3 Policy-based Interaction Model

The software-defined security architecture centers around an interaction model among objects to detect and predict security breaches, relying on the proposed Security Policy-Based Interaction Model, which utilizes virtual security functions (VSFs) within the software-defined security network.

The policy-driven interaction model governs the interactions among entities in a cloud environment. The interaction model and its unique parameters have been developed for agile detection and prediction of security threats against cloud resources. The model deals with external and internal interactions among entities representing diverse participating elements of different levels of complexity in a cloud environment. This component has been described in detail in chapter 4.

6.2.4 SDS$_2$ Security Controller

Like an SDN controller, the SDS$_2$ controller is the whole security system's brain, controlling its components and operations. It has a global view of its virtual security network and interconnected virtual security functions. The SDS$_2$ security controller consists of multiple components which mainly deal only with security functions and security services.

The SDS$_2$ controller has a complete topological graph of the connectivity of its virtual security functions (VSFs), allowing it to construct appropriate responses to attacks in real-time. The controller will construct service chaining of VSFs to create new security services to address emerging threats. Security intelligence is logically centralized in the software-based controller that maintains the global view of the security network. Hence, the global view of the security status of the protected system appears to the security applications and policy engines as a single security element. The SDS$_2$ security controller must be able to construct essential services and compose complex services into new services based on the capability of its underlying network of virtual security functions. The security controller is programmable. It configures and manages all virtual security functions under its control through its virtual security network using a Sec-Manage protocol. The SDS$_2$ allows the security Network manager alongside with the virtual security function manager to configure, manage, secure, and optimize network security resources (VSNs and VSFs) quickly via dynamic, automated programs.

The basic set of components of the SDS$_2$ security controller may consist of entity security policy-driven manager (ESPM), security policy manager (SPM), policy-based interaction manager (PIM), virtual security function manager (VSFM), the security network manager (SNM), interaction request manager (IRM), interaction detection and prediction engine (IDPE). Figure 6.3 demonstrates some of the main components within the SDS$_2$ Security controller.

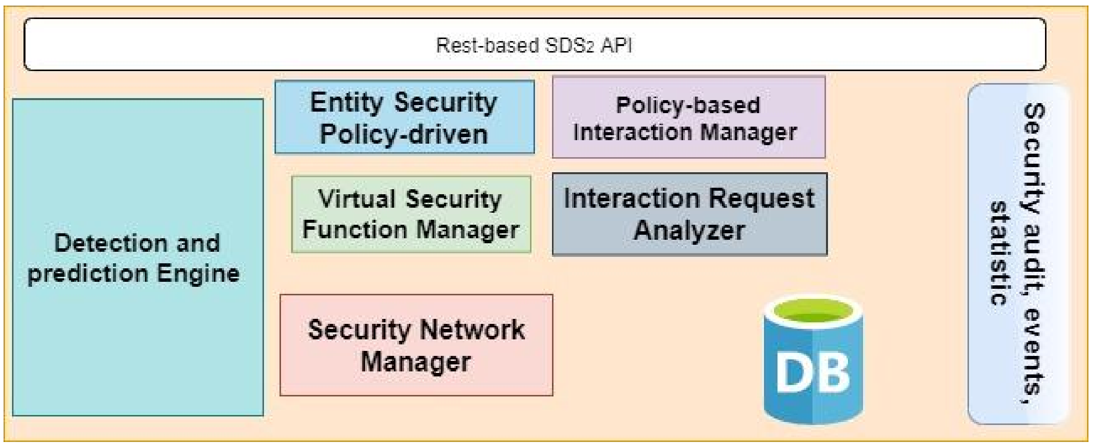

Rest-based SDS2 API
Entity Security Policy-driven
Policy-based Interaction Manager
Virtual Security Function Manager
Interaction Request Analyzer
Detection and prediction Engine
Security Network Manager
DB
Security audit, events, statistic

Figure 6.3 overview on SDS$_2$ major components

Security Policy Manager (SPM) System security policy-related activities are co-located in this module. This module provides security policy parameters related to general policies transferred from provided system policies. Security policies define the desired behavior of the heterogenous application, systems, networks, and any type of object within the system. General policies apply to all requests within the system.

Entity security policy-driven manager (ESPM) It provides specific policies related to each entity during an interaction. This module focuses on policies applied to each entity according to their features and nature. This module is responsible for extracting and translating local policies. Local policies apply separately to each entity and their interactions within the system according to their location and assigned constraints.

Policy-based interaction manager (PIM) It manages security policies applied to each interaction. It is responsible for triggering intelligent algorithms to detect and predict security violations according to interaction parameters.

Virtual Security Function Manager (VSFM) This component is responsible for creating a specific interaction software-based virtual security function (VSF). It orchestrates specific VSFs to provision on-demand security services within the security system. VSFs are created and positioned in critical locations of the infrastructure. Each VSF is launched as a specific security function such as firewall, security gateways, in-depth packet inspections (DPI), intrusion detection systems, or a mission-specific software-based security agent. These VSFs are dynamic, on-demand, and intelligent. Their primary responsibility is to detect security breaches through monitoring possible penetration doors for attacks from diverse aspects based on interaction parameters. The SDS_2 uses NFV technology for the creation of its VSFs.

Security Network Manager (SNM) This component is responsible for creating and managing the dynamic Virtual Security Network (VSN). It is responsible for providing a direct communication path between the security controller and VSFs or vice versa. A streamlined protocol is deployed for routing security messages between VSFs and SDS_2 security controller/other VSFs. It uses Sec-Manage protocol to transfer the required interaction parameters and policies between both security controllers and VSFs. The small routing table is created inside each VSF consisting of IP address, unique controller ID, and access port.

Interaction request manager (IRM) This component analyzes and manages requested interaction within the security system. The module is responsible for detecting the types of interaction. It interprets interactions in their high-level language and translates them for the security controller modules.

Interaction detection and prediction engine (IDPE) This component runs the prime algorithms for detection and prediction. It is in direct connection with the PIM module within the security controller.

SDS_2 Secure DB provides updated information for the operation of the security controller and its core modules. The DB stores updated interaction parameters and security policies related to each entity. It is responsible for storing security policies, system tenant and resource information, interaction parameters, and VSFs data.

Security Life Cycle Monitoring This component looks after the life cycle of each VSF from its initialization till expiration. Specifically, this component ensures that a

VSF will not create security issues due to virtualization even after its termination. Information related to the VSF is encrypted and maintained in the secure DB.

Security Audit, Events, and Statistics It is responsible for keeping records of all security architecture events.

6.3 SDS$_2$ Security controller – Functioning Mechanism

The SDS$_2$ security controller's main functions are to analyze the requested interactions and dynamic creation of virtual security functions so that it can orchestrate and manage on-demand security functions within the system. The SDS$_2$ has installed the following mechanisms to achieve and perform its functionalities.

6.3.1 Virtual Security Function orchestration approach

The core functions of the security controller include the creation and orchestration of virtual security functions. The security controller orchestrates virtual security functions to handle any interaction violations among cloud resources. It initiates the VSFs according to the requested interaction and availability of VSFs within the system. It can reuse existing virtual security functions if their status is IDLE.

We design an algorithm (Algorithm 6.1) to associate an appropriate VSF, which will satisfy the interaction request and current states of available virtual security functions. The security controller checks the states of VSFs and assigns them to the requested interaction. Since VSFs are software-based components, the security controller can initiate them according to the system load and the VSF's status. A "VSF State" is computed according to their job status and availability in handling the tasks.

Algorithm 6.1 VSF Creation

```
Inputs: required parameters as Resources, locations, an updated list of requested interaction, and associated VSFs
Outputs: the list of initiated VSFs and requested interaction
Switch VSF_Initiator do
case "GET":
    get a list of requested interactions and associated VSF with GET action according interaction type in the required locations from the updated VSF list
    if the list is empty then
        initiate a new VSF and assign the required resources
        set the new VSF status ON
    else
        get a list of current VSFs with their status
        set the VSFs with IDLE status to ON and assign the resources
    end if
case "VSF_ON":
    initiate the new VSF and assign the resources
case "VSF_OFF":
    change the VSF status to OFF after expiration and release the resources
```

6.3.2 VSF Configuration Approach

The security controller needs to associate VSFs to an intended or requested interaction. It requires configuring the VSF according to the interaction and policy parameters. The security controller associates an interested interaction with a VSF based on its functionalities, availabilities, and locations. The VSF status determines a VSF’s availability for the required task.

The Security controller sends configuration messages via the Sec-Manage protocol. However, to control configuration message overhead between the security controller and VSFs, the security controller may reuse the current interaction and policy parameters assigned to objects involved in the interaction and the current VSF configurations. The Security controller only transfers the required/updated parameters in the case of any changes. We design an algorithm (Algorithm 6.2) to associate a requested interaction with a VSF.

Algorithm 6.2 Association Requested interaction

```
Input: a set of requested interactions with parameters to be monitor by VSF
Output: a set of associated VSF_IDs and Interaction Parameters
1: for each requested interaction in queue (RIQ) do
2:   if (VSF_status is ON & Int_ID valid) then
3:       I=getListInteraction (i); // update I with requested int_id and location_id
4:         L =getListVSF(v); // update V with list of VSF_id, location
5:   else
6:     Exit ();
7:   for each v.vsf_id in V of each i.int_id in I do
8:         AssignInttoVSF(v); // Return appropriate assigned VSF_IDs for each requested interaction
9:   end for
10: end for
```

6.4 SDS$_2$ Security Controller – Software Implementation

This section describes the implementation of the SDS$_2$ security controller in terms of its functional components. The security controller consists of different classes in relation to its components. The overall security controller main classes diagram is shown in figure 6.4.

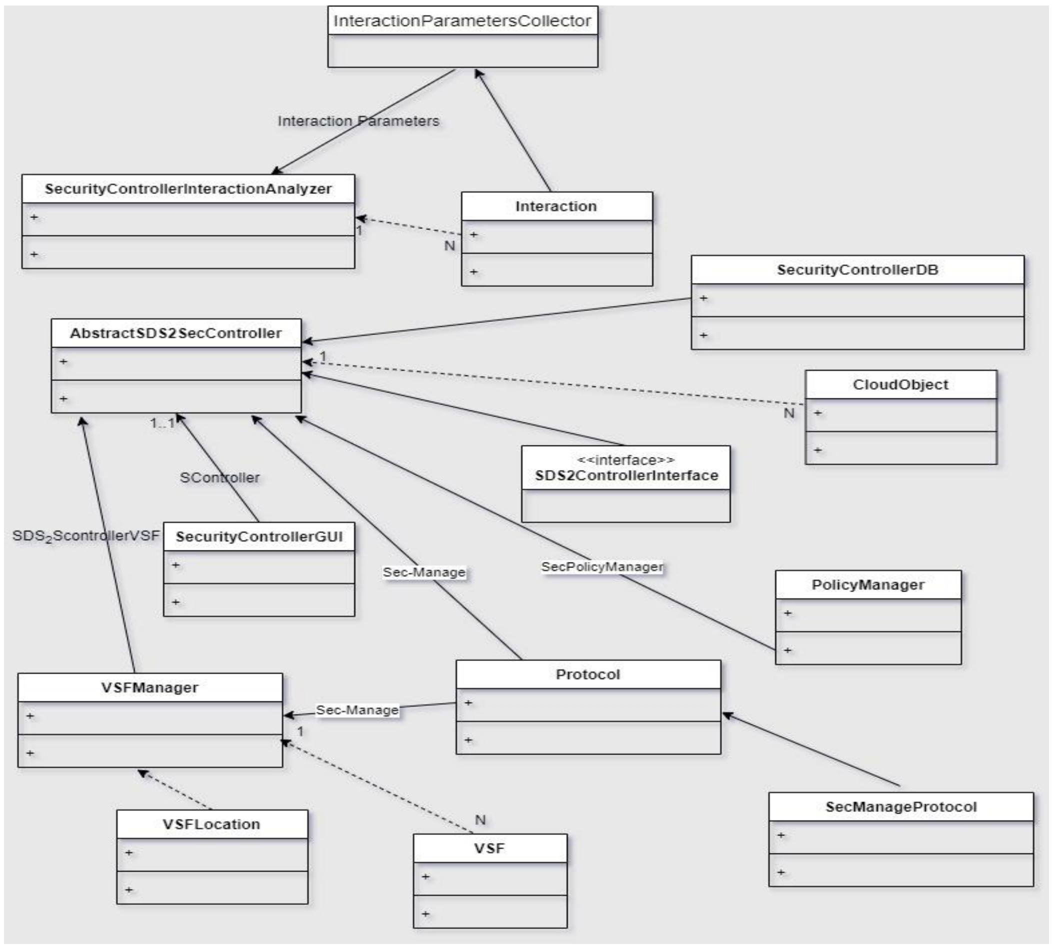


Figure 6.4 Class Diagram of SDS$_2$ Security Controller

The SecurityControllerIntercationAnalayzer class is responsible for analyzing the requested interactions within the system. The InteractionParametersCollector interpret entities interaction parameters required by SecurityControllerIntercationAnalayzer. The CloudObject class defines cloud objects required by SDS$_2$SecurityController. It includes properties of each object, including their specific attributes related to each object. The VSFManager class orchestrates the virtual security functions via VSFResources and VSFLocations. The VSF class delivers virtual security functions attributes and resource IDs.

The protocol class connects the SDS$_2$ security controller and VSFs via the Sec-Manage protocol class, including required parameters and headers. The SecurityControllerDB class provides a connection between the SDS$_2$ database and all functions of the security controller. The database is created with MySQL, including main tables. The tabl_VSF tables store VSFs' information such as VSF_id (Identification number of each initiated VSF), Loc_id (location of associated VSF within the system like Loc01 or Loc02), Int_id (interaction ID assigned to each VSFs).

6.5 Results and Performance evaluation

The main aim is to provide on-demand security services where virtual security functions are orchestrated via a logically centralized security controller. To demonstrate the practical realization of the SDS_2 Security controller in orchestrating VSFs, we deploy the architecture and security controller that manage and control VSFs within the cloud simulation environment. The VSFs can be orchestrated to provide on-demand security services for one or multiple interactions.

The primary responsibilities of VSFs are monitoring requested interaction and discovering the violations against interaction parameters at the interaction level. In this case, various interactions were initiated to evaluate the SDS_2 security controller functionality in handling and orchestrating the VSF related to multiple interaction configuration. Any interaction is dynamically assigned to a VSF for further monitoring.

The SDS_2 security controller initiates VSFs on-demand according to triggered interaction within our environment. The system can orchestrate its underlying virtual security functions to handle multiple simultaneous triggered interactions, as shown in figure 6.5.

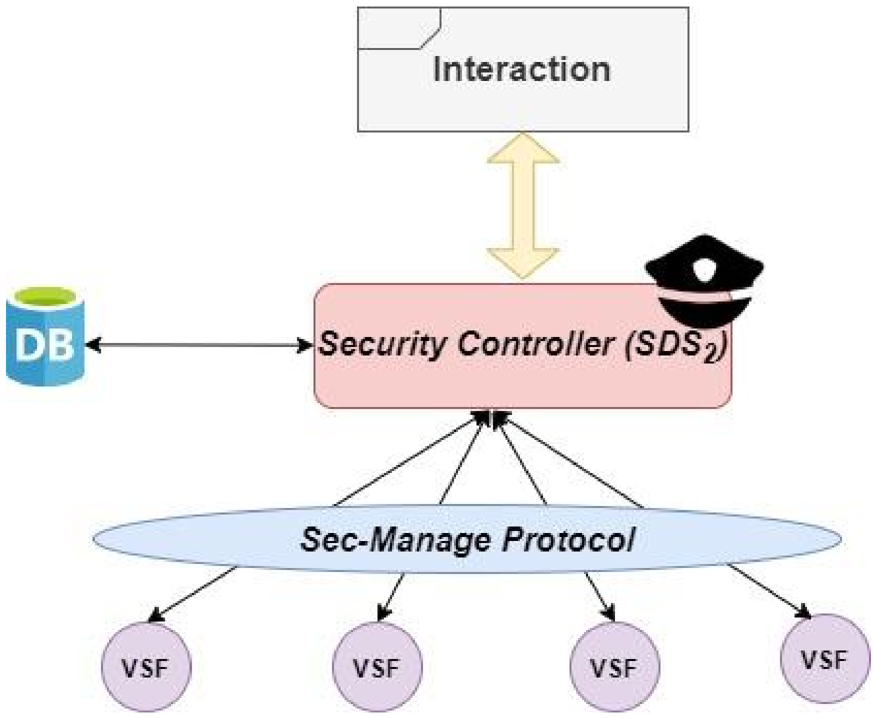


Figure 6.5 Implementation Prototype

6.5.1 Implementation Set Up

The SDS$_2$ security controller is a software platform written in Java and built using Eclipse. The security controller is connected to a secure DB built-in MySQL to store required parameters. The design's three main elements consist of a security controller, VSFs, and a Sec-Manage protocol. The three key software modules, including security controller, southbound interface, and interaction-based software security function, are responsible for the SDS$_2$ security controller, the Sec-Manage protocol, and underlying VSFs.

The security control module is responsible for analyzing the interested interaction, orchestrating, controlling, and managing the involved VSFs, detecting and predicting security violations based on associated VSF and algorithms, and the communication between VSFs and their security controller. The southbound interface is composed of Sec-Manage protocol messages, forwarding and configuring table, interaction status, and policies parameters. The module, which represents interaction-based VSF, contains classes for defining VSF and its software-based functionality and resources. We build a network using java classes where the security controller communicates with VSFs via the Sec-Manage protocol.

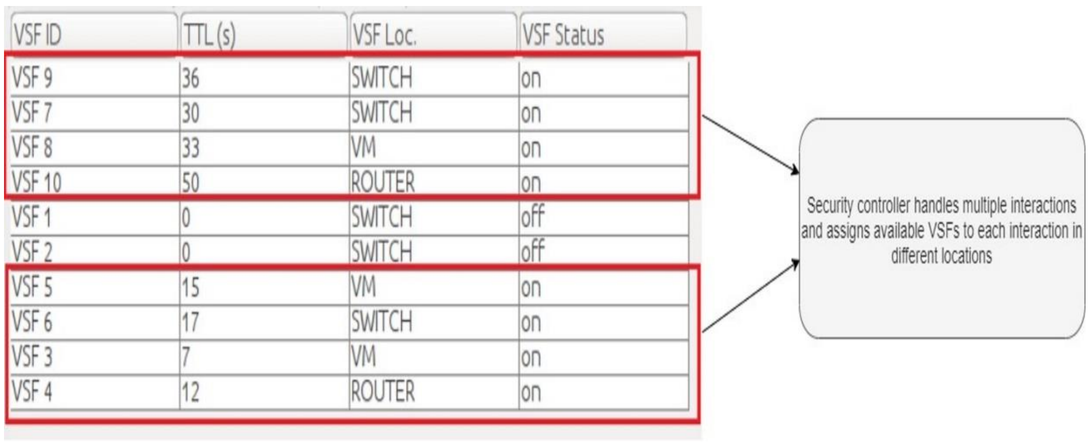
6.5.2 SDS$_2$ Security controller - Performance Evaluation

This section concentrates on demonstrating performance results related to the SDS$_2$ model. The results demonstrate the ability of the security controller in initiating and orchestrating VSFs. The Sec-Manage makes it possible for the SDS$_2$ security controller to instruct VSFs to achieve on-demand security interaction monitoring and transfer interaction parameters to the security controller. The Sec-Manage protocol empowers the security controller to detect and predict security violations according to VSFs security messages. Using the designed features, the security controller can achieve the following results:

- Dynamic and on-demand initiation of VSFs via VSFManager functionality according to requested interaction (**figure 6.6**)
- Responding and handling dynamically to simultaneous triggered interaction within the system (**figure 6.6**)
- Displaying and updating the status of virtual security functions and requested interactions (**figure 6.7**)
- Evaluating the message overload and system performance in relation to the construction of multiple VSFs (**figure 6.8** and **figure 6.9**)

Figure 6.6 demonstrates the dynamic creation of VSFs by the security controller. In this figure the interaction is triggered, and the SDS$_2$ security controller initiates a new VSF with ID 36 to monitor an interaction via VSFManager. Through the Sec-Manage protocol the security controller can instruct the VSF according to interpreted interaction parameters. The VSFManager calls its classes to create the VSF and assign the required resources to initiated VSF. Each interaction is assigned to a specific VSF with a unique ID that lasts till the end of the interaction lifetime.

The TTL in the VSF table shows a live time for each VSF during an interaction. The security controller decides where to locate the VSF according to location and condition of interaction. The VSF status shows the current status of each VSF. Figure 6.6 shows most VSFs are running according to their status as “ON” which means they are already associated with interaction and running. The security controller creates a new VSF after checking its VSF table within the database. So, if there is no same previous interaction and no existing VSF associated with the triggered interaction, the security controller initiates a VSF and assigns it to the requested interaction.

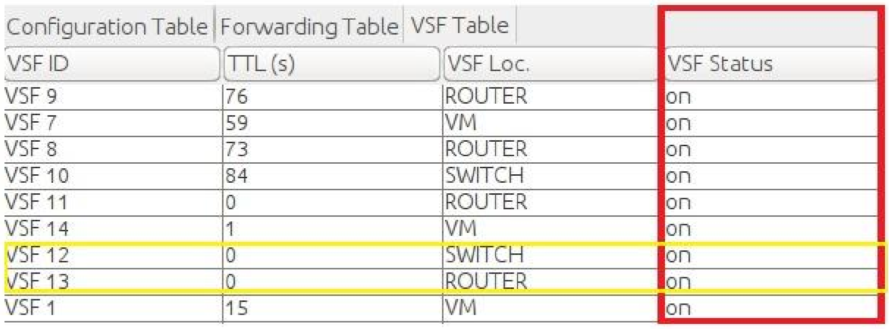

Security controller handles multiple interactions and assigns available VSFs to each interaction in different locations

VSF ID	TTL (s)	VSF Loc.	VSF Status
VSF 9	36	SWITCH	on
VSF 7	30	SWITCH	on
VSF 8	33	VM	on
VSF 10	50	ROUTER	on
VSF 1	0	SWITCH	off
VSF 2	0	SWITCH	off
VSF 5	15	VM	on
VSF 6	17	SWITCH	on
VSF 3	7	VM	on
VSF 4	12	ROUTER	on

Figure 6.6 multiple interactions handled by the security controller

The security controller is capable of handling multiple interactions simultaneously (shown in Figure 6.6). In this case, the security controller receives multiples interactions in the same period. The security controller creates an interaction queue pool and assigns the VSFs according to the priority system related to monitoring and responding to requested interaction. The security controller assigns the interaction to the most proper and available VSFs based on their triggered location and priority. The Sec-Manage protocol provides the security controller with the capability to configure and manage VSFs directly. After interaction validates, time ended, the security controller changes the initiated VSF via their status. Figure 6.7 (a, b) demonstrates expected results according to the security controller process—the security controller programs underlying VSFs to handle multiple requested interactions through VSFs configuration table via Sec-Manage protocol. As described in figure 6.7 (a), both VSF 12 and VSF 13 exceed their time to live when TTL turns to 0. The security controller updates the VSF table and changes the status for both interactions, as shown in figure 6.7 (b), from “on” to “off/expired.”

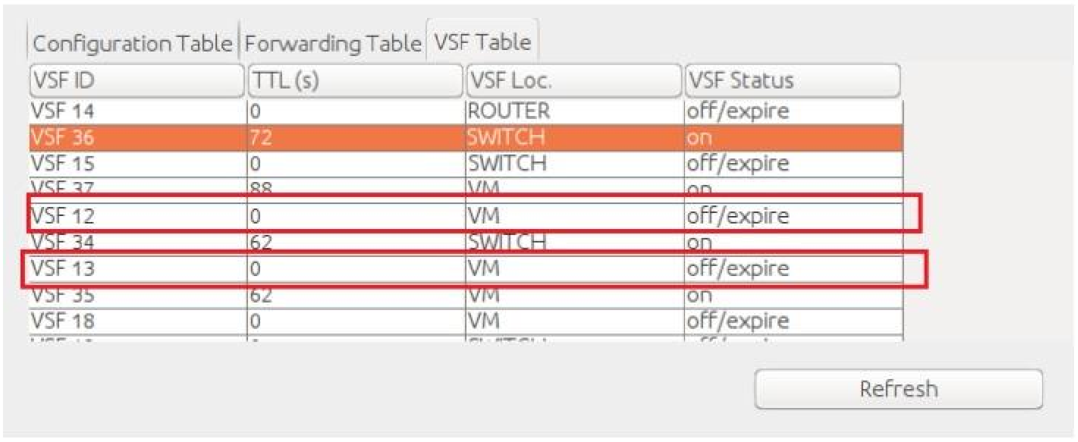
a) Status of VSFs during an interaction and before update when TTL turns to 0



Configuration Table | Forwarding Table | VSF Table

VSF ID	TTL (s)	VSF Loc.	VSF Status
VSF 9	76	ROUTER	on
VSF 7	59	VM	on
VSF 8	73	ROUTER	on
VSF 10	84	SWITCH	on
VSF 11	0	ROUTER	on
VSF 14	1	VM	on
VSF 12	0	SWITCH	on
VSF 13	0	ROUTER	on
VSF 1	15	VM	on

b) Status of VSF after update and TTL=0



Configuration Table | Forwarding Table | VSF Table

VSF ID	TTL (s)	VSF Loc.	VSF Status
VSF 14	0	ROUTER	off/expire
VSF 36	72	SWITCH	on
VSF 15	0	SWITCH	off/expire
VSF 37	88	VM	on
VSF 12	0	VM	off/expire
VSF 34	62	SWITCH	on
VSF 13	0	VM	off/expire
VSF 35	62	VM	on
VSF 18	0	VM	off/expire
[illegible]	[illegible]	[illegible]	[illegible]

Refresh

Figure 6.7 Status of VSFs

The security controller prevents overload of messages sent among its components (security controller and VSFs) by assigning existing VSF to an interested interaction. The security controller reduced the overload of messages sent between itself and its underlying VSFs via reconfiguration of existing VSF for a repetitive interaction from the same source. In this case, the security controller only sends required changes through Sec-Manage protocol and reduces the number of messages to be transferred. The security controller keeps track of interactions, VSFs, and involved initiator and reactor. Regarding the count number of interactions handled by a VSF, the security controller can reuse offline VSFs in similar interactions based on initiator and reactor. Table 6.1 demonstrates assigned interaction_id, the number of interactions (count) assigned to each VSF, and the current status of the VSFs (EXEC).

In the following table, VSF id (3) was able to handle/monitor 2 requested interactions previously completely and currently it is monitoring the third one with interaction id (3) as EXEC is Y. Similarly, in another case, VSF id (2) is currently monitoring interaction id (7). The VSF id (4) is assigned to monitor interaction id (12), but it is not running or unable to monitor the interaction. In this case, the security controller changes VSF 4 status to “OFF/expire” and assigns the interaction to another VSF.

Table 6.1 Security Controller VSF Orchestration Results

VSF ORCHESTRATION

VSF_ID	***INT_ID***	***Count***	***EXEC***
VSF 3	***3***	***3***	***Y***
VSF 2	***7***	***8***	***Y***
VSF 4	***12***	***0***	***N***
VSF 12	***12***	***1***	***Y***

Two performance metrics evaluate the SDS_2 security model's efficiency: controller processing time in handling multiple interactions and average provisioning time to initiate VSFs (as shown in figure 6.8 and figure 6.9). The security controller processing time in this scenario signifies the total time from when the security controller receives the requested interaction when it initiates the VSF and assigns it to that specific interaction. The average processing time of handling multiple interactions depends on: i) several simultaneous requested interactions, ii) similarity between currently requested interactions with previous/existing configuration of VSF, iii) the availability of existing IDL VSFs. For instance, for each new interaction with no previous configuration availability, the processing time will be longer than in other cases. The reason is the lack of previously existing stored data on the requested interaction used by the security controller. So, the security controller must spend more time analyzing, collecting, and transferring the data to VSF. Besides, in some cases, the security controller requires to initiate new VSFs and assign new resources.

However, at the time of requested interaction, if there are still idle VSFs, it will reduce the time by excluding the time required to initiate a new VSF. However, to reduce the processing time, the security controller uses a previous VSF, and instead of deleting them, they are just put idle mode for further processing.

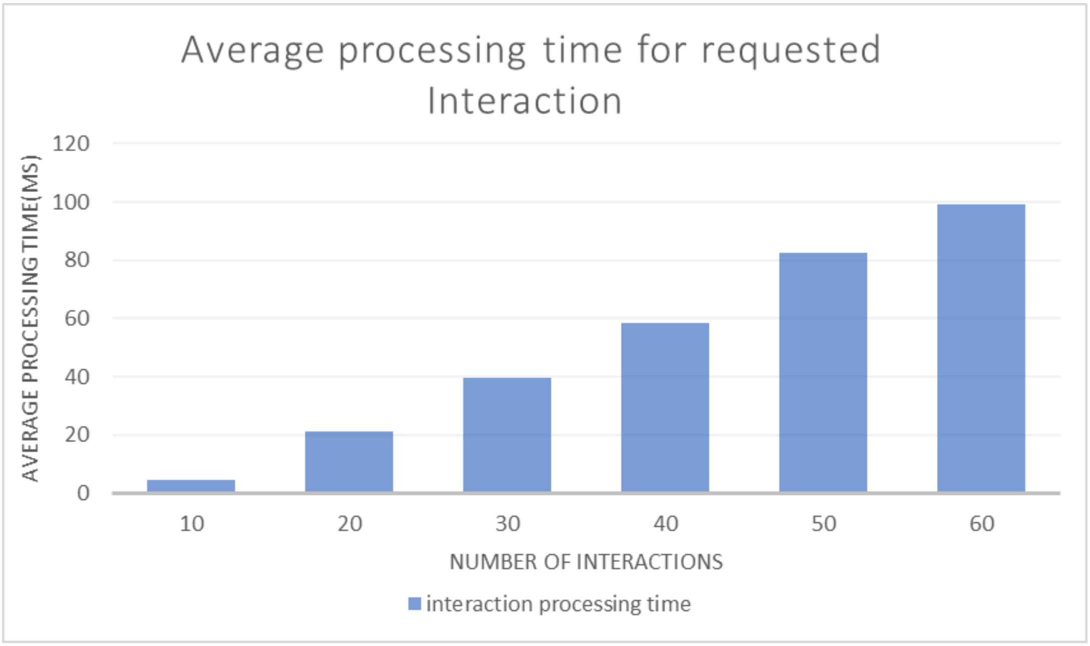


Figure 6.8 SDS$_2$ security controller the average processing time for all types of interactions

Figure 6.8 describes the average processing time for the security controller to analyze the inputs according to the number of requested interactions. The interaction numbers include all three types of interaction handled by the security controller within the system. The number of triggered interactions is increased by 10 each time from 10 to 60. The more interactions triggered, the higher the processing time growth. The reason relies on the time that the security controller requires to analyze different types of interactions. The time the security controller requires increased slightly as the number of requested interactions increased. However, since some interactions are repeated during the system, the increase is not sharp as the security controller can use its stored data assigned to each repeated interaction.

Figure 6.9 demonstrates the average time that the security controller requires to initiate a VSF. To evaluate the SDS_2 performance according to VSFs initiation times, we consider increasing the VSF number by 10 each time from 10 to 50. The aim is to demonstrate the time required by the security controller to initiate the VSFs and assign their resources through its VSFManager and its functionalities.

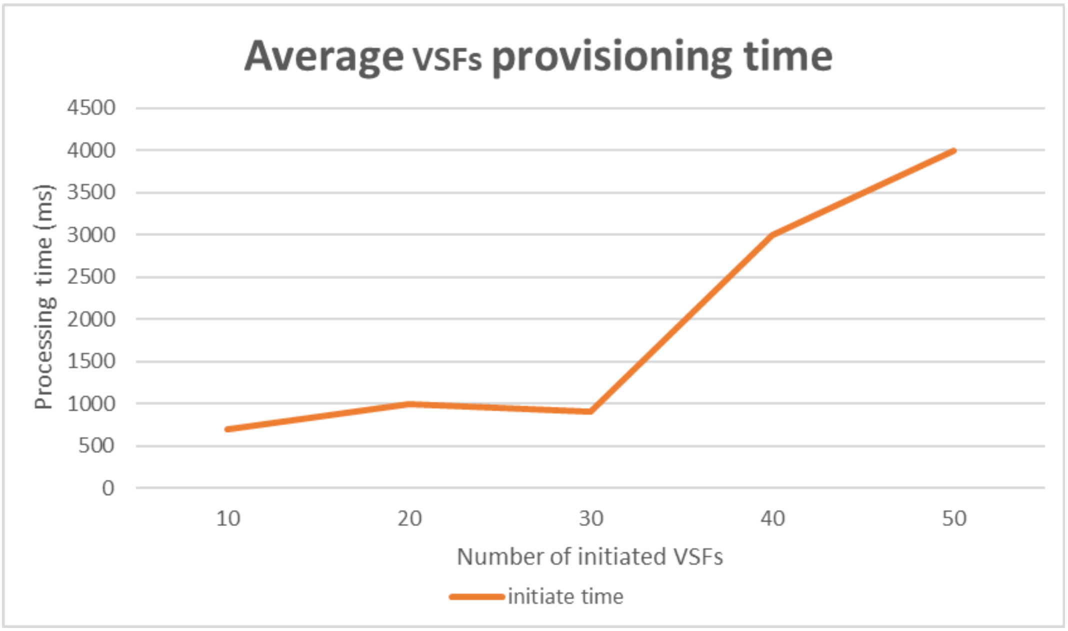


Figure 6.9 Provisioning On-demand security services

We noticed that by increasing numbers of VSFs to handle interactions, there is a sharp rise in terms of initiation time of VSFs between 30 to 40. This increase is that in the edge of 30, the validation time for most of the existing VSFs expired, as a result, the security controller requires to initiate new VSFs to handle the interested interactions. With the expiration of the majority of existing VSFs, the security controller needs to construct new VSFs, including the time to run the new VSFs (time to assign their required resources) and their configurations.

One of the reasons is preserving VSF security for not holding a VSF for a long time in a specific location. Regardless of their popularity in handling the number of interactions, after a time, their TTL will go down to 0. In this situation, the security controller requires to initiate the new VSF via VSFManager.

6.6 Summary

In this chapter, we have introduced our software-defined security service architecture with main components including Security Controller, Intellectual algorithm (discussed in chapter 4), Sec-Manage protocol (described in chapter 5), and VSFs. In this chapter we demonstrate the proposed security architecture for protecting cloud infrastructure. We developed a new security controller that uses the Sec-Manage protocol to dynamically and efficiency orchestrate VSFs in response to detection and prediction of security violations. In this chapter a detailed description of functional components of security architecture has been provided.

Chapter 7

Software-Defined Security Service Platform

7.1 Introduction

As discussed in Chapter 3, the proposed software-defined security service model delivers a resolution to secure cloud infrastructure by employing a logically centralized security control to monitor, orchestrate, and manage on-demand security services. Our proposed security model decouples security functions and security networks from underlying infrastructure. We have introduced a logically centralized Software-Defined Security Service (SDS_2), a policy-based interaction model, and Sec-Manage protocol in our earlier chapters.

For the purpose of controlling and managing of virtual security functions, we proposed security service architecture for supporting the orchestration, coordination, and provision of security services within the cloud infrastructure (Chapter 6).

This chapter describes detailed functionality of the components in the SDS_2 platform (described in Figure 7.1). Moreover, in this chapter we conduct a performance analysis using proposed SDS_2 platform with all proposed components including the security controller, the Sec-Manage protocol, the policy-based interaction algorithm, and various VSFs.

The reminder of this chapter is organized as follows. Section 7.2 presents the architecture and components of the SDS_2. Section 7.3 describes the procedure of the SDS_2 platform in the provision of on-demand security services. Section 7.4 presents the platform implementation. Section 7.5 determines performance evaluation. Section 7.6 summarizes this chapter.

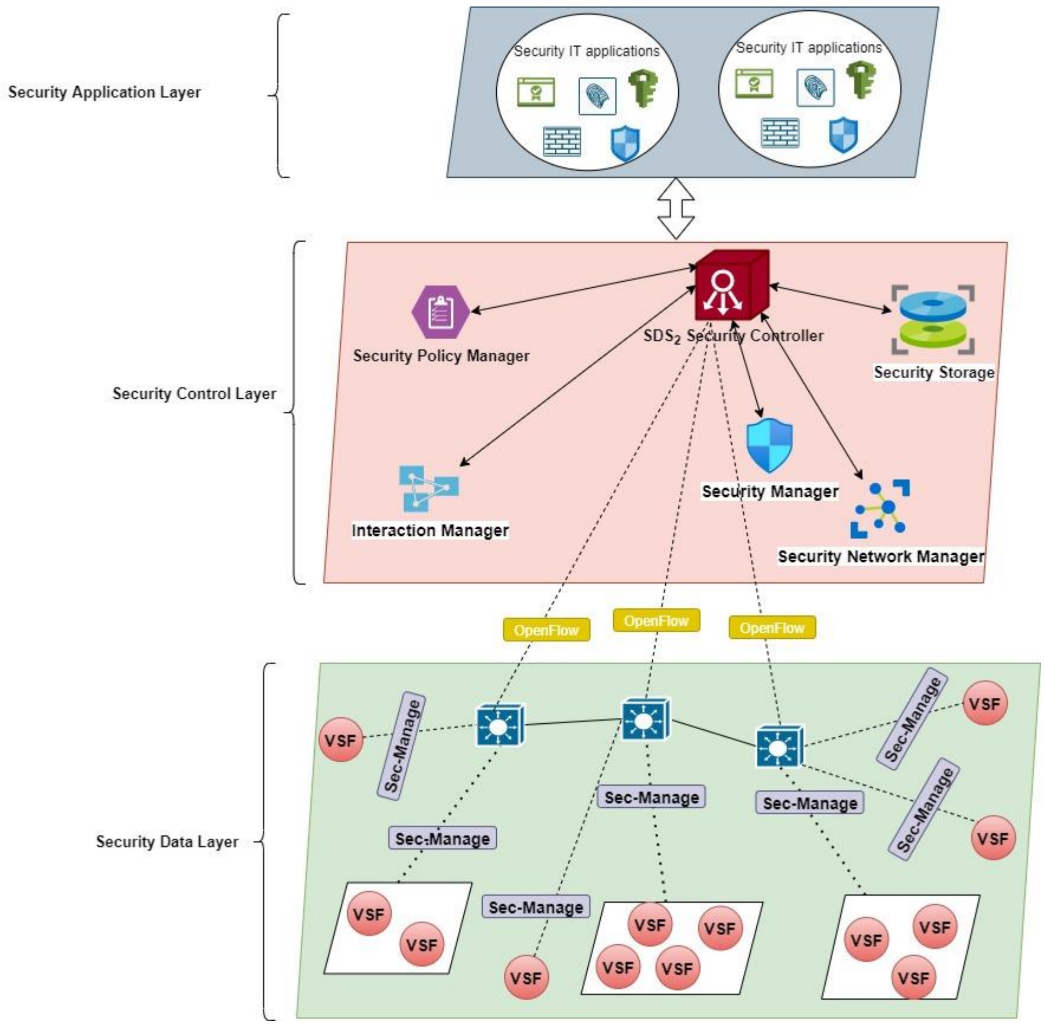


Figure 7.1 Overview of SDS_2 Layers and components

7.2 Integrated Software-Defined Security architecture

This section describes the integrated software-defined security service structure using cloud, SDN, and NFV concepts.

The SDN, NFV, and cloud all share the software-defined concept where physical resources are virtualized into software components. They share the underlying physical infrastructure and the virtualization layer and require controllers and orchestrators to provision services. Naturally, SDN, NFV, and cloud evolve into an integrated software-defined infrastructure or software-defined system (SDS) to optimize the use of resources, eliminate the redundancy in their structure, and provide a richer set of services on demand.

However, the security of such a software-based virtual environment will entail more than just the security issues common to all domains, the security issues specific to each domain, the security gaps among them, and the security of the overall infrastructure. For this purpose, the SDS_2 is deployed to provide security in such integrated software-defined infrastructure. The SDS_2 takes advantages of three technologies to establish its main functionalities:

Cloud Infrastructure Cloud provides a multitenant environment to share the infrastructure resources. The cloud resources enable SDS_2 to initiate its interaction-based virtual security functions.

Software-Defined Networking (SDN) SDN provides automatic programmable virtual connectivity among cloud objects. The SDN is based on the separation of the network control from the data forwarding functions, allowing the controller to directly program the underlying infrastructure and present it as a high-level network functionality abstraction to applications and network services [138]. The SDS_2 utilizes the SDN features, constructing dynamic and direct communication between its specific security controller and underlying virtual security functions.

Network Function Virtualization (NFV) NFV technology provides a vast number of software-based virtual functions within the system. It enables network functions to be realized and executed as software instances in a VM or container on a single host instead of customized hardware appliances. NFV implements network functions using software virtualization methods and operates them on top of underlying hardware equipment. The SDS_2 applies the NFV concept to create a specific software-based virtual security function to monitor the cloud resources interaction. The VSF inspired by Virtual Network Function (VNF) structure focuses only on security features.

The SDS_2 structure has been designed in such a way that it can be integrated into any software-defined infrastructure. The conceptual architecture of SDS_2 within the integrated Cloud/SDN/NFV is shown in figure 7.2.

The upper layer includes security services, interfaces, and tools to define security policies for cloud objects. A set of pre-defined security policies is simulated and used over the platform according to the nature and feature of defined cloud objects. The policies are interpreted to govern and validate constraints and rules for entities within the system, such as an eligible access control action to be taken during a triggered interaction. The high-level policies act as input to the interaction security policy module in the platform structure's middle layer. In chapter 4, we explained our novel policy-based interaction model and the main components.

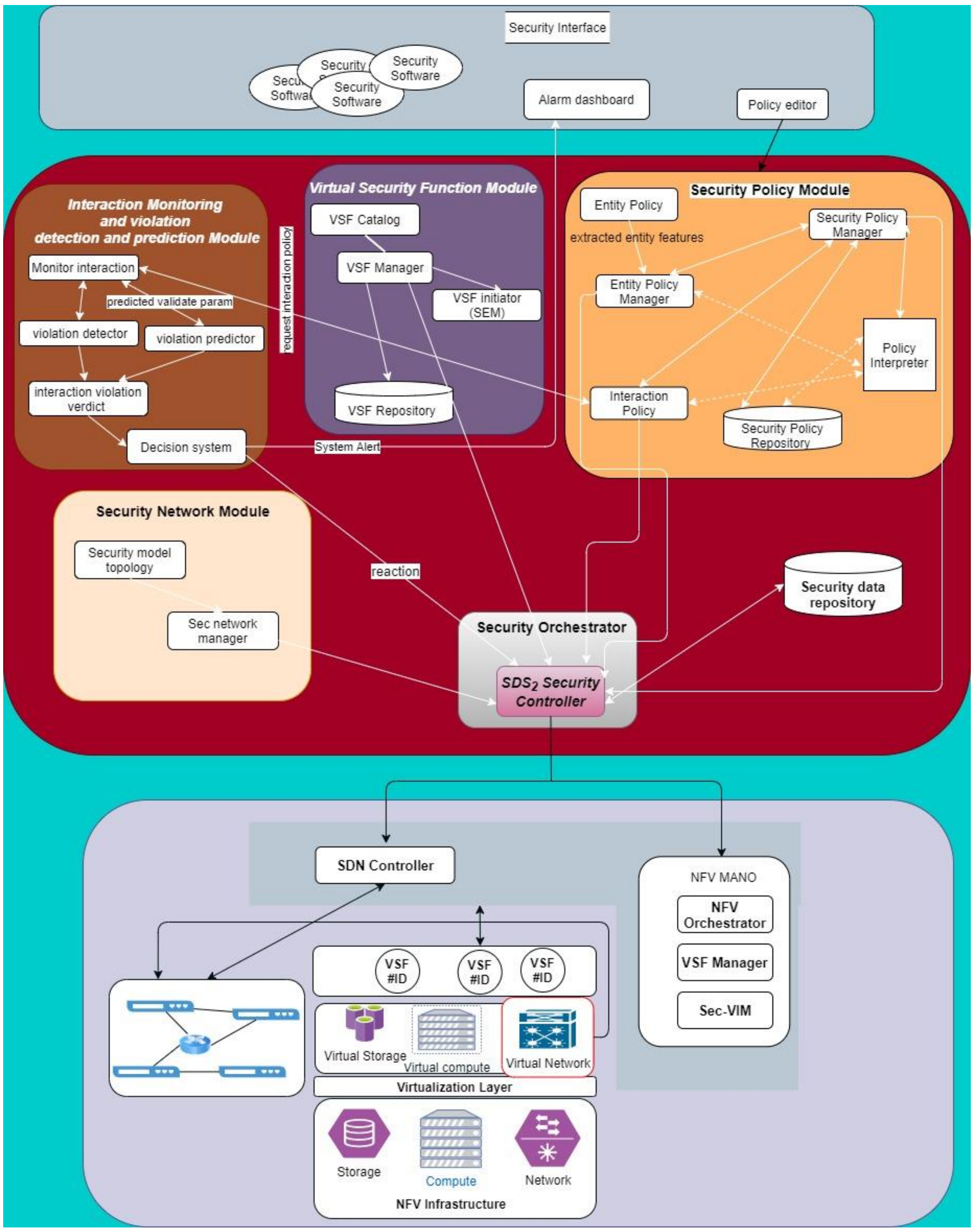


Figure 7.2 Conceptual structure of SDS$_2$ in an integrated Cloud/SDN/NFV

The middle layer mainly focuses on the orchestration of on-demand virtual security functions and other SDS$_2$ modules. This layer consists of a security orchestrator, a security policy module, a security interaction monitoring module, and violation detection and prediction module, a virtual security function module, and a security network module.

Security Orchestrator It houses the SDS_2 security controller. The security controller controls and manages the whole virtual security environment. In chapter 6, we described the security controller and several of its main components. Figure 7.3 shows the workflow of the security controller in our SDS_2 security model.

Security Policy Module This module includes functionality for interpreting high-level policies according to entities (initiator, reactor) and the entity interactions related to security policies. As the name implies, the security policy manager monitors and enforces policies for both entity policy and interaction policy managers. The policy interpreter is responsible for translating high-level policy expressions to allowable interaction rules and constraints.

The entity policy manager extracts policies related to each entity that evolves in an interaction according to its features and nature intrinsic characteristics. There are different constraints on each interaction and its interacting entities, such as time of interaction, and location. This module enforces the policies on entities during the time interval that interaction takes place within the system. The policy repository contains policy expressions which are expressed via a simple language-based template. Chapter 6 contains the foundation of specification of the policy expressions to be used in our security system.

Virtual Security Function Module The module contains a VSF manager, a VSF catalog, a VSF repository, and a VSF initiator. It is in charge of creating VSFs and assigns required resources from the Virtual Infrastructure Manager (VIM) to virtual security functions. It is in control of running VSF scripts start-up defined within the system to monitor an interaction within the system.

The VSF catalog includes simple VSF templates to be run by the VSF manager. The VSF initiator is similar to Element Management (EM) in NFV. The Security Element Management (SEM) is responsible for the functional management of VSFs. Similar to EMS, each SEM can be assigned to one or more VSFs. In our design, we consider each SEM for one specific interaction-based VSF. The VSF repository contains data related to each VSF like ID, location, IP, and MAC addresses.

Security Network Module It consists of two main components: the security model topology and Sec-Net manager. As the name Sec-Net manager implies, it is responsible for constructing logical networking between system components. It is connected to the SDN controller to provide dynamic construction of virtual security networking links specifically between the security controller and VSFs. Through an internal interface, it is connected to security model topology. The Security model topology stores the constructed links between the security controller and its VSF. A unique protocol has been implemented to transfer the required interaction parameters through this network link between the security controller and a VSF. In chapter 5, we described the design and implementation of this communication protocol. The figure 7.3 describes an overview of SDS_2 security platform workflow. The workflow of SecNet is displayed in Figure 7.4.

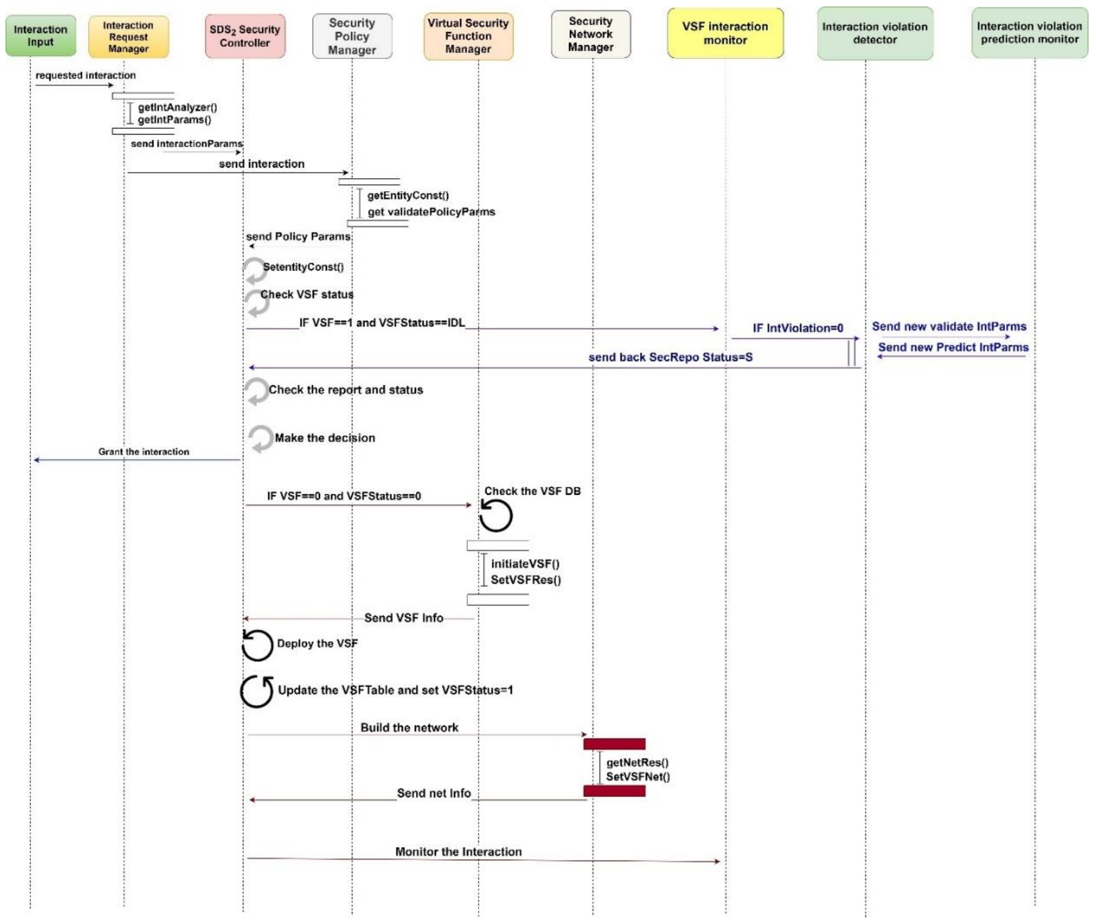


Figure 7.3 Overall SDS_2 security platform workflow

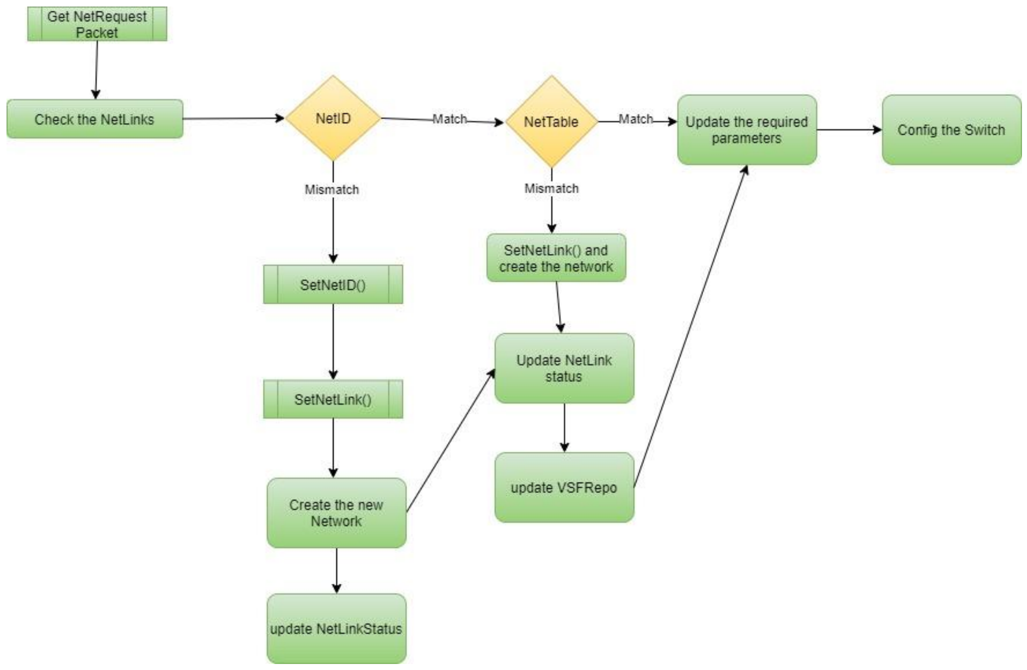


Figure 7.4 Workflow of security Network structure

7.3 SDS$_2$ Platform – Procedure of provisioning on-demand Security Services to Protect Cloud Resources

The process of a cloud security service via an SDS$_2$ model consists of five stages (shown in figure 7.5). At the moment of receiving an interaction, SDS$_2$ analyses the triggered interaction and initiates a virtual security function to monitor the specified interaction. An interaction can be for several reasons, including i) User Interaction- sending an interaction via system API to act on a cloud resource, ii) Triggered interaction- a suspicious interaction occurs within the platform, and this triggers the security controller to activate a VSF requesting it to monitor a specific interaction between suspicious entities, iii) Specific requested Interaction- scheduled or random monitoring an interaction between specific entities requested by an intelligent component of the security controller.

Provisioning Stage At this stage, the SDS$_2$ security controller analyses the triggered interaction and extracts the necessary parameters to provide a monitoring service as needed at different cloud system levels. According to the interaction parameters, the security model protects entities as they evolve via an on-demand virtual security function service. Thus, the model is capable of provisioning on-demand and dynamic virtual security service at the critical points of the system.

Orchestration Stage At this stage, the SDS_2 security controller analyses the interaction parameters and policies and accordingly orchestrates one/more virtual security functions (VSFs) to detect and predict violations.

Programmability Stage At this stage, the SDS_2 security controller orchestrates its virtual security functions to respond to any interaction-based security violation incidents. It automatically allocates required interaction and policy parameters to its VSF monitoring functions to acquire the interaction's security status. The VSFs will send back their monitoring results to the security controller. The security algorithms then use the status results to detect and/or predict security violations. The underlying virtual network function enables automatic programmability based on the interaction-based model.

The SDS_2 security controller configures an on-demand underlying virtual security function to acquire satisfying security monitoring results. The VSFs are initiated accordingly to monitor a triggered interaction. The security controller then automatically configures the VSFs using its Sec-Manage protocol. The VSFs, armed with validated interaction parameters, monitor the requested interaction. The security controller configures the VSFs at their initiation and during the monitoring process.

Security Violation Stage At this stage, the security controller analyses the security monitoring reports sent by each VSF for a specifically requested interaction. Then it performs the interaction-based security detection and prediction algorithms. The results determine if the triggered interaction violated any validated parameters. The results are transferred to the security controller for further decisions.

Completion Stage At this stage, the programmed VSF functions monitoring results and results of interaction-based security detection and prediction methods have been returned to the security controller. At this final stage, the security controller decides the interaction of interest, the VSF state, and predicts future security violations according to interaction parameters. The security controller will finalize and report its security services results in relation to the requested interaction.

Figure 7.5 shows the procedure of provisioning on-demand security services according to described stages.

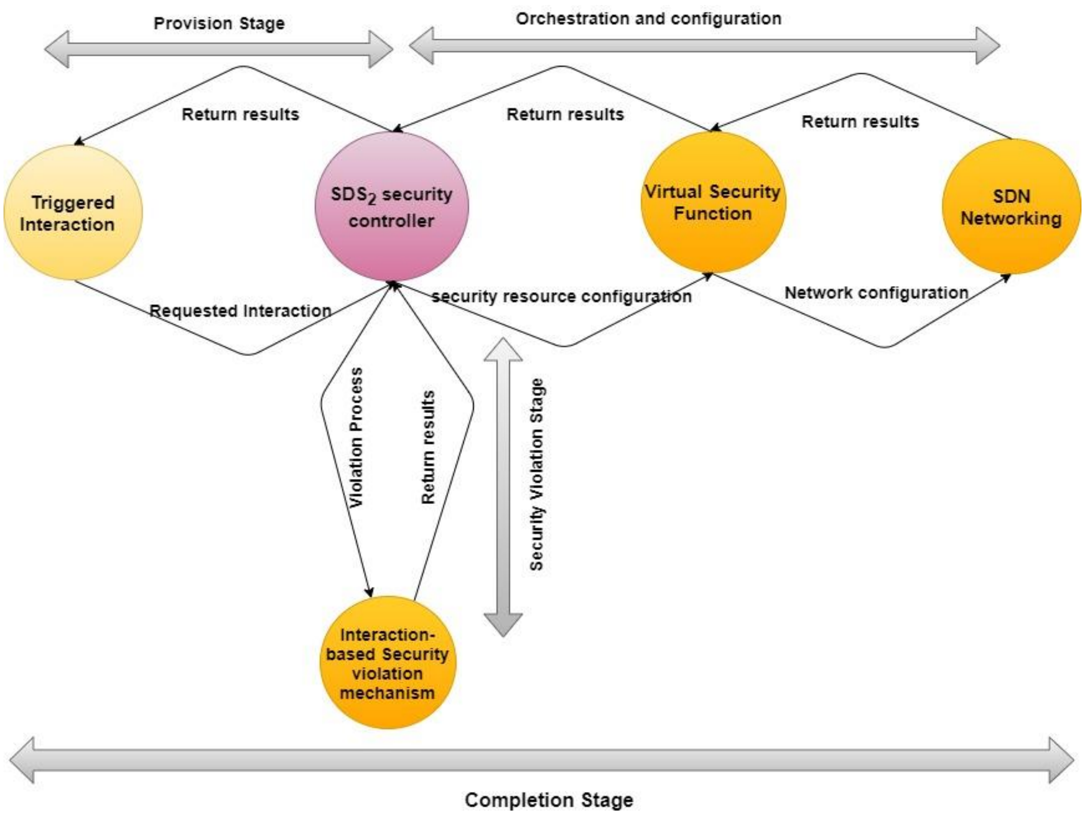


Figure 7.5 Overall process of provisioning on-demand security services via SDS$_2$

The process happens through the SDS$_2$ controller and its VSFs. Details of the SDS$_2$ security controller and VSF workflow are described as in the following.

At SDS$_2$ Security Controller: The security controller analyses triggered interactions and then orchestrates and configures the VSFs following the interaction model and SDS_2 service requirements. The operation of the SDS_2 Security controller is shown in figure 7.6.

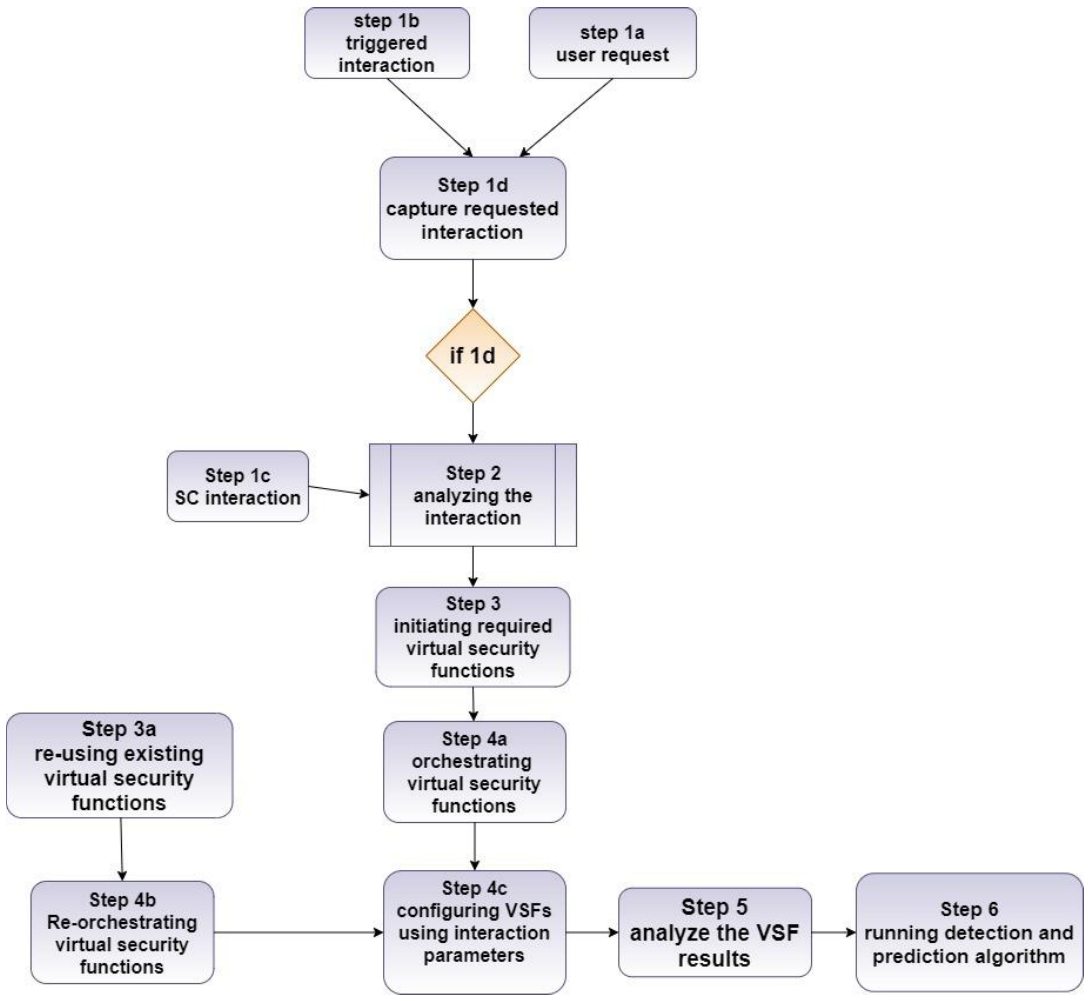

step 1b triggered interaction
step 1a user request
Step 1d capture requested interaction
if 1d
Step 1c SC interaction
Step 2 analyzing the interaction
Step 3 initiating required virtual security functions
Step 3a re-using existing virtual security functions
Step 4a orchestrating virtual security functions
Step 4b Re-orchestrating virtual security functions
Step 4c configuring VSFs using interaction parameters
Step 5 analyze the VSF results
Step 6 running detection and prediction algorithm

Figure 7.6 Overview of SDS_2 security controller workflow

At the virtual security function: upon initiation through the security controller, the virtual security function resources will be orchestrated by the security controller to monitor the specified interaction. The operation of virtual security functions is represented in figure 7.7.

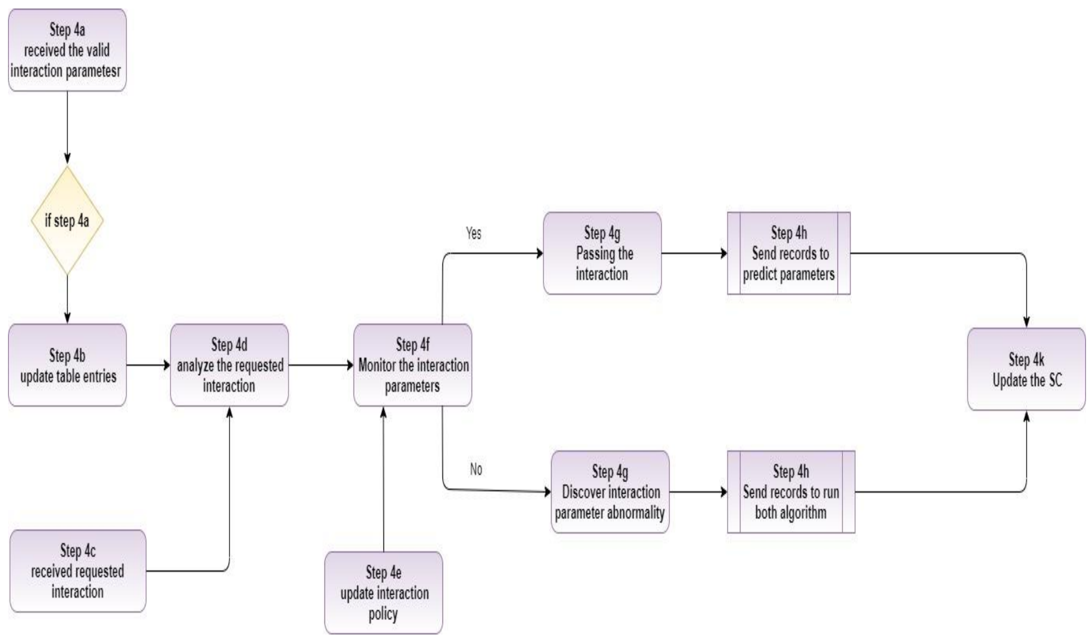


Figure 7.7 Overview of VSF workflow

7.4 SDS$_2$ Platform Implementation

7.4.1 Implemented Platform

In previous sections, we presented the proposed model, described the architecture and its main modules, as well as the design of the main components. In this section, we present the platform that integrates all the main components.

The SDS$_2$ model for the cloud has been designed and implemented. The platform consists of deployments of significant components, as demonstrated in table 7.1. Each component is implemented within our Java program. The overall implementation defines the main cloud objects within the cloud environment. The SDS$_2$ runs as security software on top of the infrastructure using its interfaces to access VIM and SDN controller resources (Figure 7.8).

Table 7.1 Main Implemented Components of The Platform

Integrated Platform components	***Features***
Security controller (SC)	*interaction module, security policy module, interaction detection/prediction module, VSF net module*
VSF	*VSF initiator module, the VSF manager module*
Sec-Manage Protocol	*protocol messages and packet header*

The SDS$_2$ Security controller: We represent the design and implementation of its main components that interpret the policies, analyze the interaction requests, orchestrate and provision on-demand virtual security functions over the cloud system. The internal interfaces enable communication between the components. In chapter 6, we described the security controller and its significant functions.

Sec-Manage protocol: It is responsible for the communication between the security controller and its virtual security functions and the exchange of interaction and policy parameters among them. This protocol was presented in chapter 5.

Virtual Security Function (VSF): A VSF in our usage is created to perform a specific security function and deployed at strategic locations in the cloud that require protection. The VSFs functions are controlled and managed by the SDS$_2$ security controller through the Sec-Manage protocol—the VSF monitors the interaction between entities according to interaction-based model parameters. The VSF catalogs integrate in our SDS$_2$ software platform.

OpenFlow protocol (opensource): The protocol is a well-known programmable network protocol intended to manage and direct traffic between virtual network functions. By taking advantage of the OpenFlow protocol, the SDS$_2$ security controller adapts and reuses the protocol to direct traffic among its VSFs.

OpenStack (open source): OpenStack is a central open-source cloud computing platform that orchestrates and manages shared storage, compute, and network resources using multiple hypervisors based on a set of applications and open-source. The SDS_2 security software runs on a VM in OpenStack compute node.

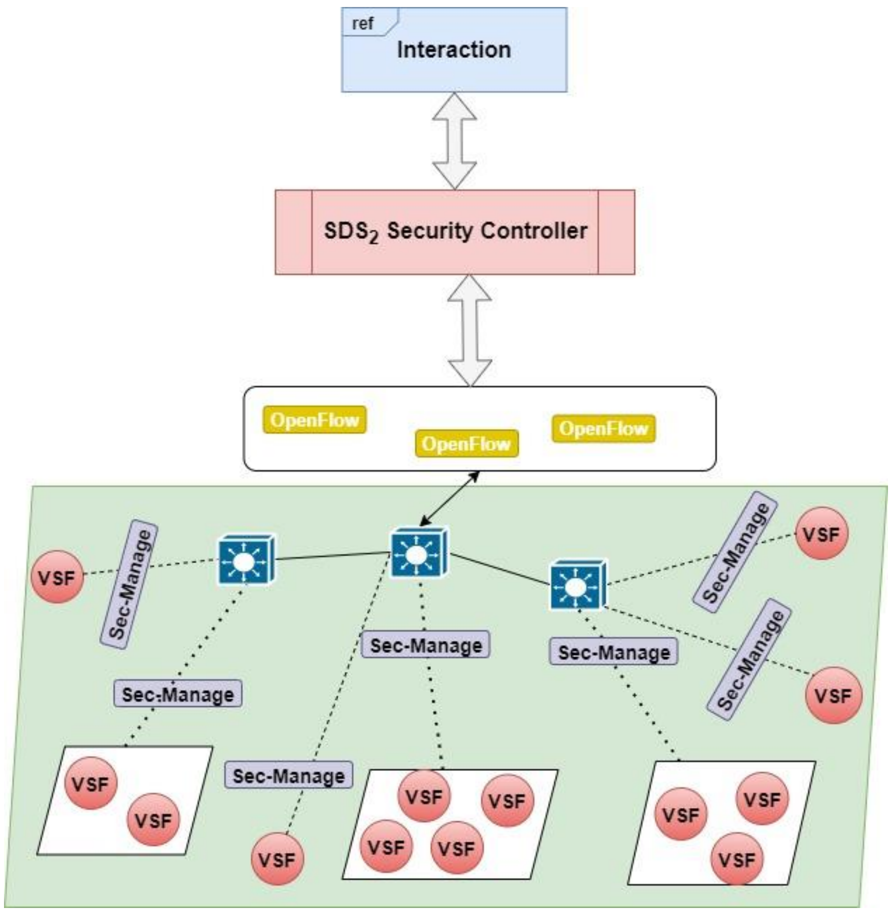


Figure 7.8 overall view of Platform structure

OpenFlow switch (opensource): This is an OpenFlow-enabled data switch that provides communication over an OpenFlow channel to the SDN controller. It enables packet forwarding and lookup based on its routing table entries. Each switch contains essential information such as Port No., IP address, and port name. The SDS_2 controller can update its network and VSF links through the switches. Figure 7.8 shows the structure of the platform.

7.4.2 Implementation Scenarios and Results

To demonstrate the proposed SDS_2 model performance, we run different experiments on our implemented platform covering various interaction scenarios to demonstrate the proposed security model's validity and services. The SDS_2 orchestrates VSFs in a cloud environment to achieve on-demand security service to protect cloud resources. In our security scenario different interactions triggers from different entities. Interactions can occur between different resources at different cloud infrastructure hierarchy levels and from different locations/domains. The cloud object types are defined based on their intrinsic characteristics. The interactions are defined based on different conditions and the environment surrounding the participating objects. The SDS_2 security controller orchestrates the VSF at different locations to monitor the interaction of interest (Figure 7.9). In each interaction, a different set of conditions/policies are applied to test the security controller's efficiency in formulating security violations. The SDS_2 security controller communicates access resources

The test scenarios are based on three main categories of interaction: i) the level of access, ii) the interaction parameters, iii) the interaction types. The security controller analyses the interactions according to their variable parameters (mode, action, and positional relation) extracted from the interaction of interest. The SDS_2 security controller orchestrates and configures the relevant underlying virtual security functions and allocates them at relevant places related to the location of the triggered interaction.

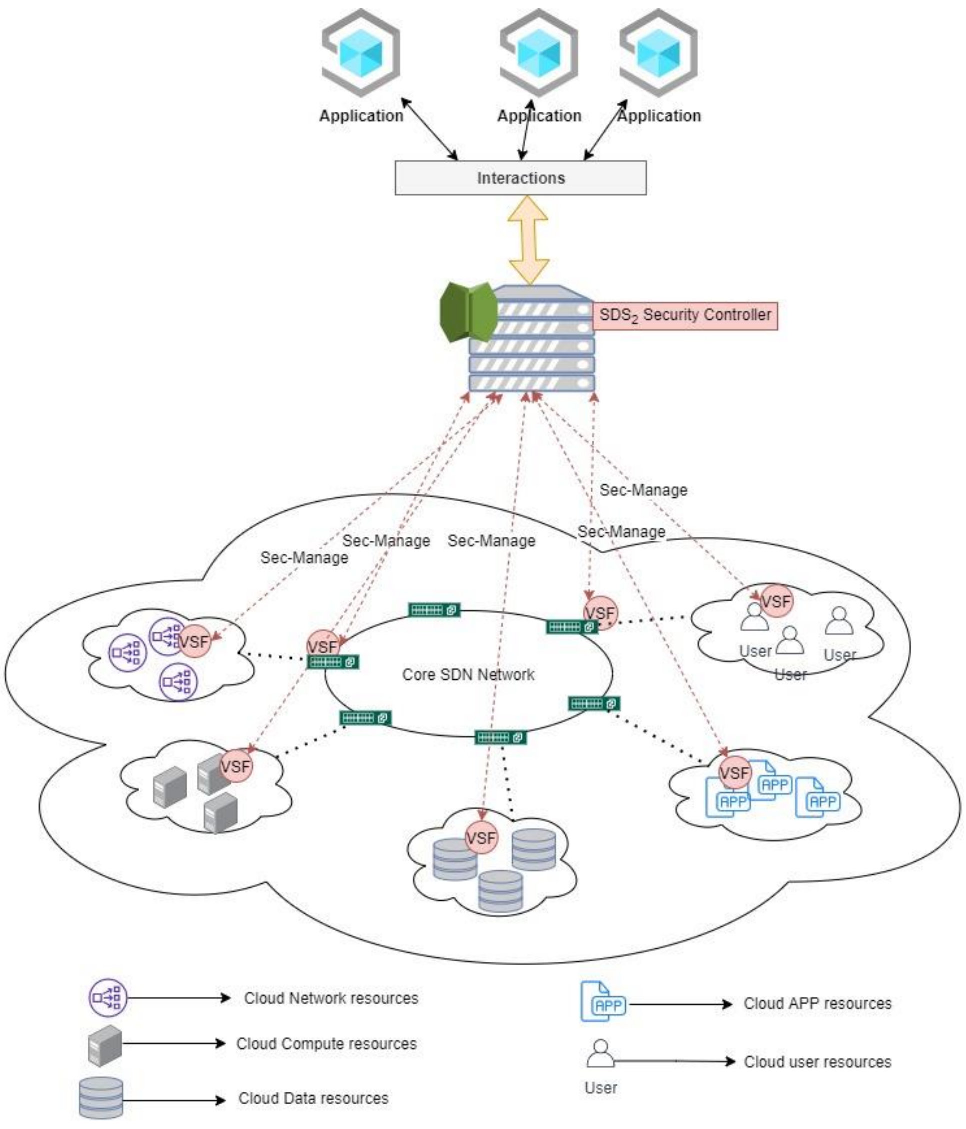


Figure 7.9 SDS$_2$ service Implementation Scenario

Each scenario explores the SDS$_2$ capability in discovering and predicting the security interaction of interest. The SDS$_2$ orchestrates the VSFs accordingly and assigns a random identification id to each. We tested various types of interaction within the testbed running the SDS$_2$ service and algorithms to monitor each interaction between entities. Both *general* and *local policies* are extracted and stored in the policy repository.

Experiment 1: In this scenario, we examine the security model performance through the variation of extracted interaction parameters. We simulate an adversary interaction trying to manipulate cloud resources performing various interactions. This scenario takes into account an interaction modification by an attacker to gain access to the cloud resources triggering different interactions with various interaction parameter conditions (Con*) that consists of variable interaction parameters including mode, action, and positional relation at the same access level. To test the SDS_2 platform in each case one interaction parameter ((M, *, *, *), (*, R, *, *), (*, *, A, *)) is varied at a time. The achieved results demonstrate the SDS_2 security controller's capability to detect security violations via orchestration of its underlying virtual security functions. The scenario prototype is demonstrated in figure 7.10.

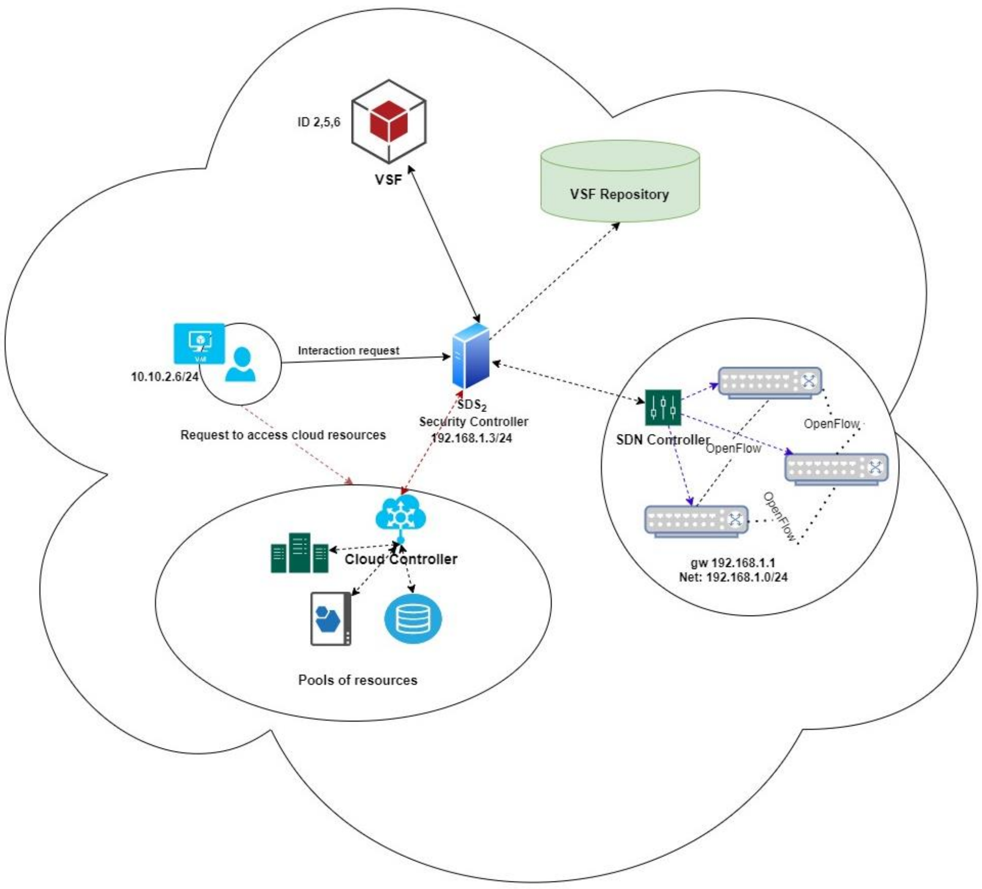


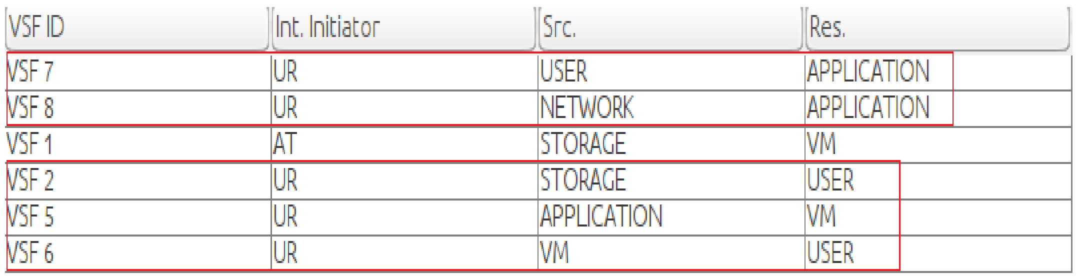
Figure 7.10 Overall state of threat Scenario

Scenario Requirements: In order to carry out the simulation to validate the system operation and performance, we established the following requirements.

- Default policies are applied to both entities involved in the interaction.
- Location policies are stable during the triggered interaction.

To test the efficiency of the system, we run different experiments containing different interaction parameters. Malicious interaction was launched to simulate user interaction attacks against different resources (Figure 7.11). The SDS_2 security service intelligently detects malicious interactions by the discovery of policy violations related to interaction parameters. The security controller intelligently, according to its VSF reports, determines the state of interaction parameters. The ISVD algorithm calls upon the discovery of interaction violation (described in chapter 4) against validated extracted interaction parameters.

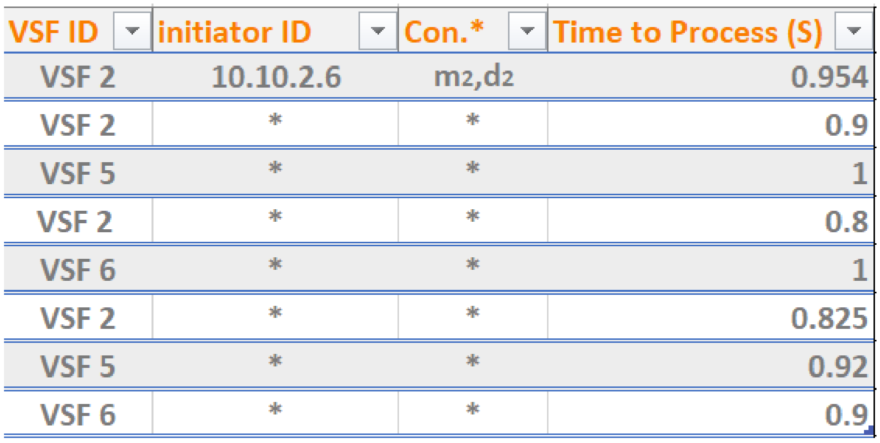
The VSF detects interaction parameter patterns, discovers interaction parameter states (stability of interaction parameters) over a number of interactions, and monitors the behavior of the variable parameters. The discovery of a stable interaction parameter enables the SDS_2 security controller to optimize the detection processing time that the same source triggers. The reason is that after the discovery of the constant interaction parameter, the SDS_2 security controller only focuses on variable interaction parameters. So, it saves time to process all parameters in each step.



VSF ID	Int. Initiator	Src.	Res.
VSF 7	UR	USER	APPLICATION
VSF 8	UR	NETWORK	APPLICATION
VSF 1	AT	STORAGE	VM
VSF 2	UR	STORAGE	USER
VSF 5	UR	APPLICATION	VM
VSF 6	UR	VM	USER

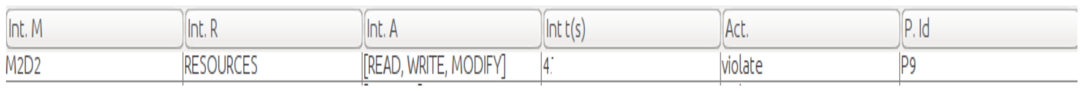
Figure 7.11 Assigned VSF to interactions

Each interaction's processing time is different in each case based on processing the interaction parameters with the condition (Con*) that the interaction parameter mode is the same. So, the time for the system to process the interaction is calculated accordingly. However, we calculate the detection process time for some of these triggered interactions with the condition of stability of interaction mode parameter during interaction period time, depicted in figure 7.12. A single violation detection for Int6 assigned to VSF2 is demonstrated in figure 7.13 as a sample record. The results are stored in the security controller database, and any policy changes are recorded in the policy repository directly. The time to process is defined as the process time the platform requires to detect an interaction violation in the case of the applied condition.



VSF ID	initiator ID	Con.*	Time to Process (S)
VSF 2	10.10.2.6	m2,d2	0.954
VSF 2	*	*	0.9
VSF 5	*	*	1
VSF 2	*	*	0.8
VSF 6	*	*	1
VSF 2	*	*	0.825
VSF 5	*	*	0.92
VSF 6	*	*	0.9

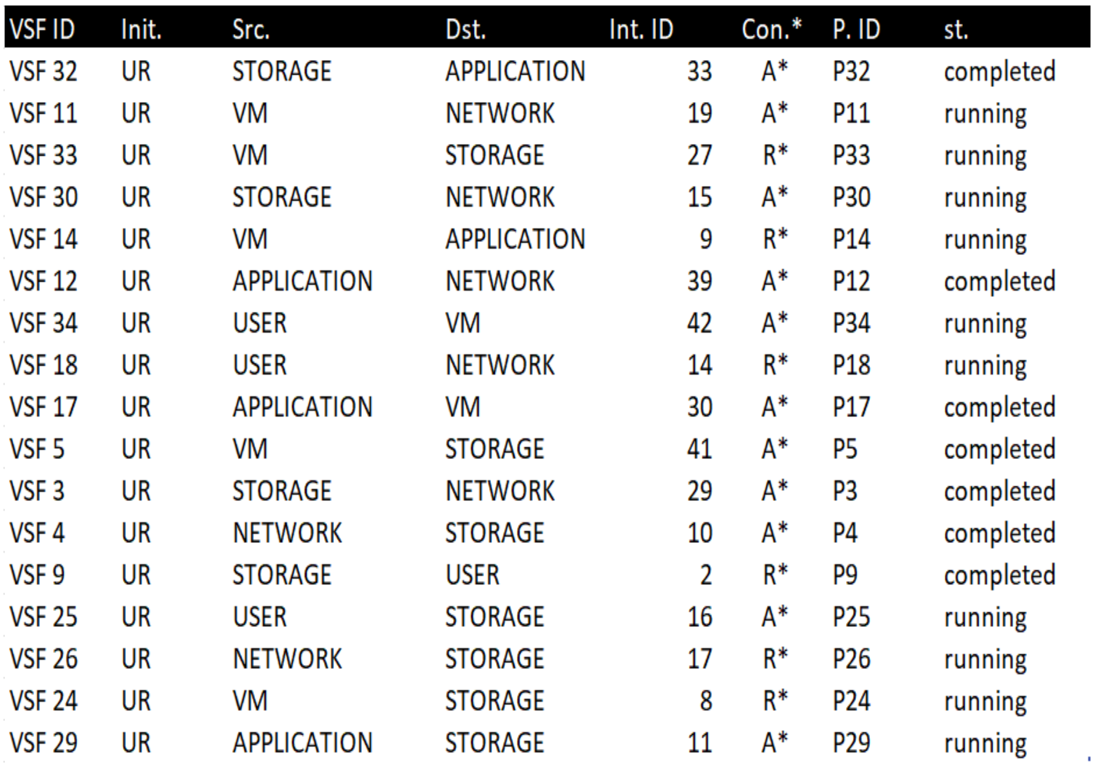
Figure 7.12 Process time by VSFs



Int. M	Int. R	Int. A	Int t(s)	Act.	P. Id
M2D2	RESOURCES	[READ, WRITE, MODIFY]	4	violate	P9

Figure 7.13 Violation detection of Int6

To test the SDS$_2$ security platform against other interaction parameter conditions, we run various interactions between different resources. The interactions are triggered via user requests. The triggered interactions and their status of actions are illustrated in figure 7.14.



VSF ID	Init.	Src.	Dst.	Int. ID	Con.*	P. ID	st.
VSF 32	UR	STORAGE	APPLICATION	33	A*	P32	completed
VSF 11	UR	VM	NETWORK	19	A*	P11	running
VSF 33	UR	VM	STORAGE	27	R*	P33	running
VSF 30	UR	STORAGE	NETWORK	15	A*	P30	running
VSF 14	UR	VM	APPLICATION	9	R*	P14	running
VSF 12	UR	APPLICATION	NETWORK	39	A*	P12	completed
VSF 34	UR	USER	VM	42	A*	P34	running
VSF 18	UR	USER	NETWORK	14	R*	P18	running
VSF 17	UR	APPLICATION	VM	30	A*	P17	completed
VSF 5	UR	VM	STORAGE	41	A*	P5	completed
VSF 3	UR	STORAGE	NETWORK	29	A*	P3	completed
VSF 4	UR	NETWORK	STORAGE	10	A*	P4	completed
VSF 9	UR	STORAGE	USER	2	R*	P9	completed
VSF 25	UR	USER	STORAGE	16	A*	P25	running
VSF 26	UR	NETWORK	STORAGE	17	R*	P26	running
VSF 24	UR	VM	STORAGE	8	R*	P24	running
VSF 29	UR	APPLICATION	STORAGE	11	A*	P29	running

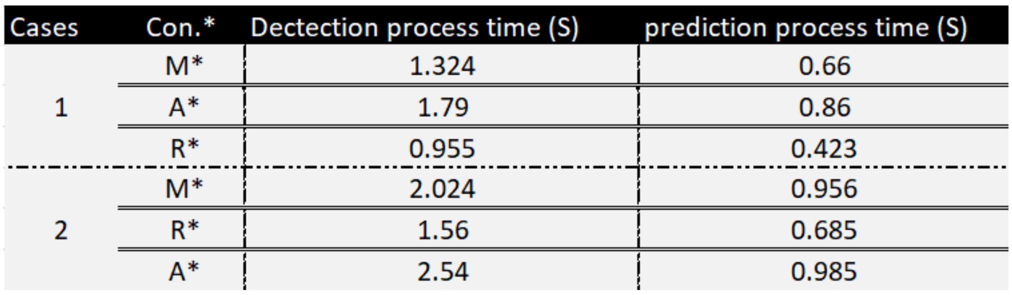
Figure 7.14 performed tests with two sets of interaction conditions (A*, R*)

We have evaluated the SDS_2 service platform's performance in experiment 1 according to two features: the average detection and prediction processing. We consider the time based on the number of interactions that take place. In this experiment, we evaluate the platform performance according to the following cases showed in Table 7.2. each case tested against different conditions related to the stability of the random interaction parameter.

Table 7.2 Tested cases

Experiment	*Number of tested interactions*	*Conditions*
Case 1	***20***	***M*, A*, R****
Case 2	***30***	***M*, A*, R****

Figure 7.15 demonstrates the average time the SDS_2 service requires to detect and predict the security violations based on the different variable parameters. As shown, the predicated processing time is less than detection since the ISVP algorithm uses the current state of validating parameters for prediction.



Cases	Con.*	Dectection process time (S)	prediction process time (S)
1	M*	1.324	0.66
	A*	1.79	0.86
	R*	0.955	0.423
2	M*	2.024	0.956
	R*	1.56	0.685
	A*	2.54	0.985

Figure 7.15 Processing average detection and prediction time for Case 1 and 2

Experiment 2: In this experiment, we focus on evaluating our security model in the face of two scenarios related to security policies i) no-policy, ii) dynamic policies. In the first scenario, there is no defined security policy parsed to triggered interactions. The results demonstrate the platform's capability in dealing with such cases. The next scenarios concentrate on monitoring the platform's behavior against dynamic policy changes related to triggered interactions. In a dynamic cloud environment, the security policies frequently change, which requires platform agility in applying the changes using its security functions. We use the basic set up of experiment 1 presented above but run various interactions *with/without defined security policies*. In our interaction-based software-defined security model, the policy module allows fine granular policy specifications based on various entities and interaction attributes.

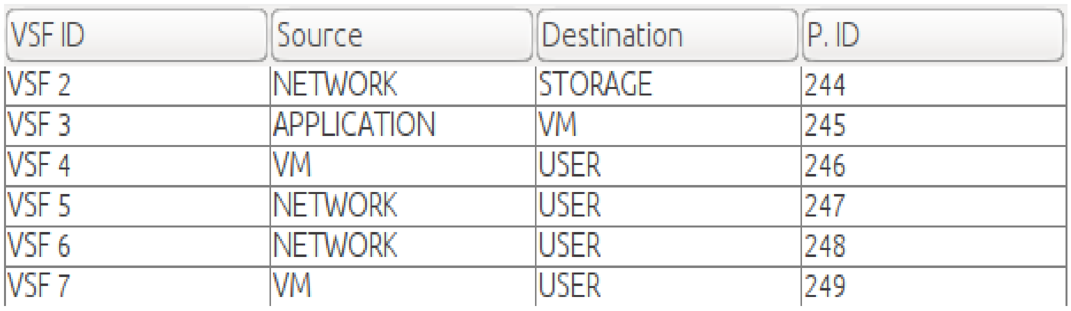
To test the system, we allow changes to the defined policies within the system and monitor the behavior of the security controller and VSFs in handling the situation. We can map this experiment to real-world scenarios where resource locations or policies dynamically change due to factors like resource/data location (one server to another server in same/different geographical location) changes, changes of general policies in companies, and owner changes.

Experiment requirements:

- We change entity policies that affect one interaction parameter at the time of initiation. We change local security policies applied to each interaction according to their entities features and roles. We define various policy files within the system.

- The general system policies stay steady. The system policies refer to policies we defined by default for our cloud data center based on each security domain (described in chapter 4).

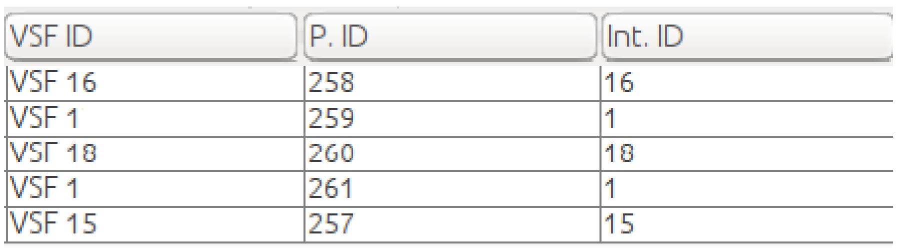
Different policy files are defined within the platform to be used against various interactions. We consider the same general policy file for all interactions while different security policies are applied to entities using defined policy files. We follow a simple policy language to construct the policies to be used. The security policy manager function captures the policies applied to interactions and entities. The policies related to entities are stored in the policy repository. This module enforces the security policies extracted from entity security policy-driven manager. Policies directly affect the interaction parameters. Figure 7.16 shows a few requested interactions captured by the security controller, monitored by the VSFs and theirs assigned policy id (PID).



VSF ID	Source	Destination	P. ID
VSF 2	NETWORK	STORAGE	244
VSF 3	APPLICATION	VM	245
VSF 4	VM	USER	246
VSF 5	NETWORK	USER	247
VSF 6	NETWORK	USER	248
VSF 7	VM	USER	249

Figure 7.16 Captured requested interaction and their policies

Regarding the changes of policies, Figure 7.17 shows that the VSF 1 has been assigned to monitor the interactions with Int ID 1, which is triggered from the same resource but with different policies. The security controller assigned the same VSF to monitor the interaction; however, since VSF detects the policy changes, the security controller updates the policy repository and assigns new changes dynamically.



VSF ID	P. ID	Int. ID
VSF 16	258	16
VSF 1	259	1
VSF 18	260	18
VSF 1	261	1
VSF 15	257	15

Figure 7.17 Changes of policies for

To test the SDS_2 platform capability on the agile discovery of defining security policies and demonstrating its automatic and dynamic configuration of security policies, we conduct tests with no pre-defined policies assigned to sources. So, we initiate interactions where no policies were parsed within the repository. In this case, VSF discovers the lack of policies for any interaction parameter and sends messages through the Sec-Manage protocol to get the updated policy or new policy approved by the security controller. The process is illustrated in the following workflow (Figure 7.18).

We monitor the behavior of VSFs in dealing with two prominent cases. We simulate the interaction between two resources where no policies are set with Con* (M, A*, R, t)). Figure 7-19 shows the interaction, process time, and assigned new policies. According to figure 7-19, the VSF 3 discovers that no-policy has been parsed for the Int. ID 3. After exchanging messages between VSF and the security controller, a new policy (P.ID 245) is assigned according to defined policies explained in chapter 4.

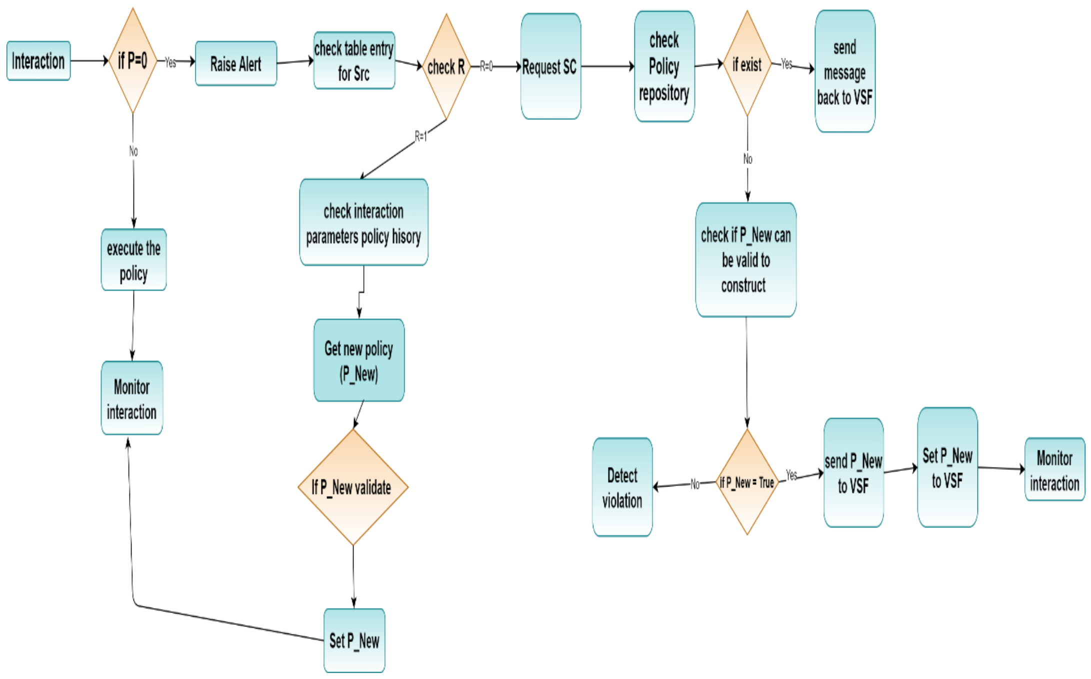
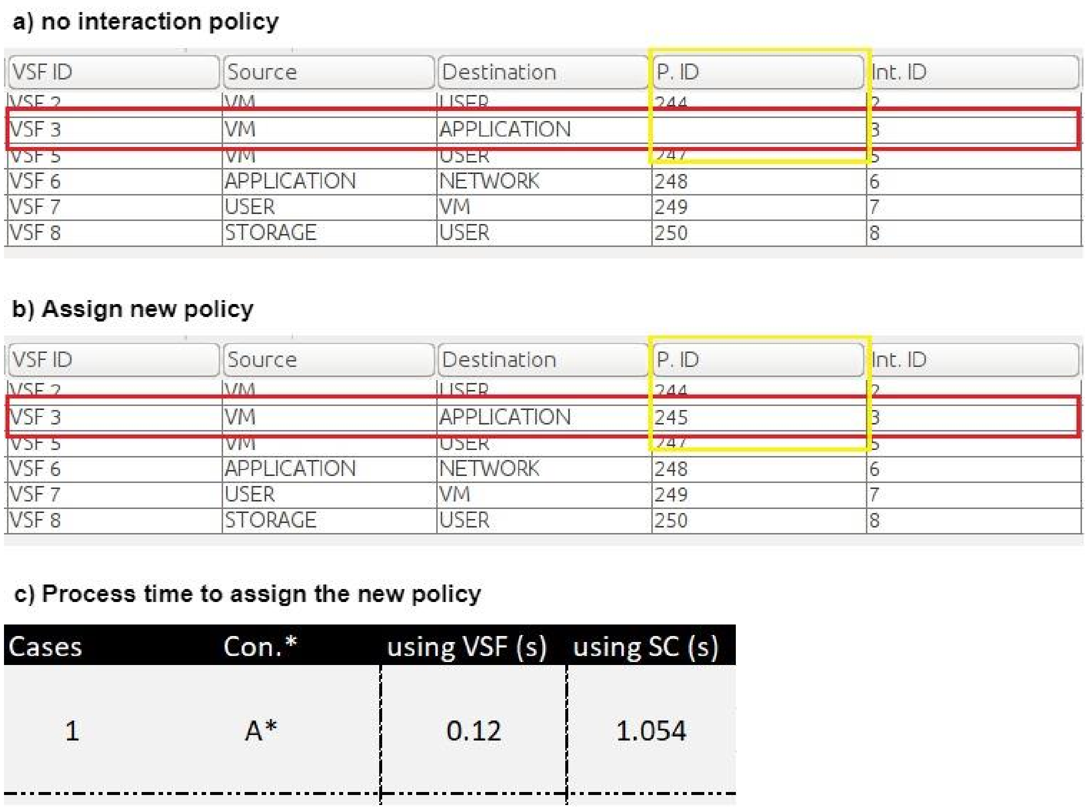


Figure 7.18 Workflow of VSF no-policy process

We tested the ability of the SDS_2 system to monitor its behavior in regard to dynamic changes in security policies. In this case, the SDS_2 security controller and its services receive many interactions triggered by the same resource. We consider Con* (M*, A*) where the reactor entity's position is varying within the system. For this purpose, policies related to R* change frequently, and the system is required to adapt to new security policies dynamically. As discussed in chapter 4, each R level is governed by different security policies based on the access level and the assigned role-based policies applied to each entity at that level. The SDS_2 security controller updates the entity's policies using its entity security policy-driven manager function for R* interaction parameters. It updates the VSFs after enforcing the policies and sends back the updated validate parameters. Figure 7.20 demonstrates updated policies for 12 interactions being handle with 4 VSFs. The SDS_2 service uses the VSFs that previously ran the interaction to detect and predict future parameters.



a) no interaction policy

VSF ID	Source	Destination	P. ID	Int. ID
VSF 2	VM	USER	244	2
VSF 3	VM	APPLICATION		3
VSF 5	VM	USER	247	5
VSF 6	APPLICATION	NETWORK	248	6
VSF 7	USER	VM	249	7
VSF 8	STORAGE	USER	250	8

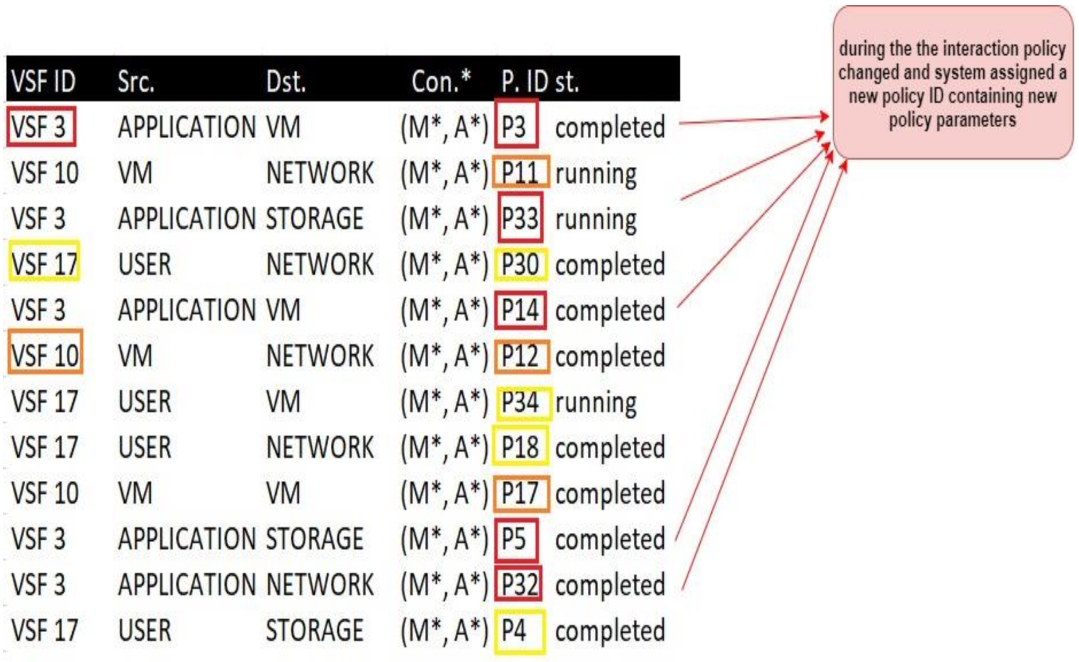
b) Assign new policy

VSF ID	Source	Destination	P. ID	Int. ID
VSF 2	VM	USER	244	2
VSF 3	VM	APPLICATION	245	3
VSF 5	VM	USER	247	5
VSF 6	APPLICATION	NETWORK	248	6
VSF 7	USER	VM	249	7
VSF 8	STORAGE	USER	250	8

c) Process time to assign the new policy

Cases	Con.*	using VSF (s)	using SC (s)
1	A*	0.12	1.054

Figure 7.19 No-policy case- a) demonstrates VSF discovery of interaction with no assigned P.ID, b) presents the new P.ID for interaction, c) shows process time of orchestrating a new policy for particular interaction in two main scenarios



VSF ID	Src.	Dst.	Con.*	P. ID	st.
VSF 3	APPLICATION	VM	(M*, A*)	P3	completed
VSF 10	VM	NETWORK	(M*, A*)	P11	running
VSF 3	APPLICATION	STORAGE	(M*, A*)	P33	running
VSF 17	USER	NETWORK	(M*, A*)	P30	completed
VSF 3	APPLICATION	VM	(M*, A*)	P14	completed
VSF 10	VM	NETWORK	(M*, A*)	P12	completed
VSF 17	USER	VM	(M*, A*)	P34	running
VSF 17	USER	NETWORK	(M*, A*)	P18	completed
VSF 10	VM	VM	(M*, A*)	P17	completed
VSF 3	APPLICATION	STORAGE	(M*, A*)	P5	completed
VSF 3	APPLICATION	NETWORK	(M*, A*)	P32	completed
VSF 17	USER	STORAGE	(M*, A*)	P4	completed

during the the interaction policy changed and system assigned a new policy ID containing new policy parameters

Figure 7.20 The SDS$_2$ platform deals with policy changes during an interaction

Experiment 3: we have tested the system against different types of interaction triggered by different sources. In this experiment, we monitor entities in three main scenarios described in Table 7.3.

Table 7.3 Experiment scenarios

Experiment	*Tested interactions*	*Condition*
Case 1	***Requested (UR)***	***Variable policy, Variable interaction parameters***
Case 2	***Specified (SC)***	***Variable policy, Variable interaction parameters***
Case 3	***Triggered (AT)***	***Variable policy, Variable interaction parameters***

The requested interactions refer to user interactions triggered from a user interface (UR). The specified interactions consider interactions triggered by the security controller to monitor specific interactions (SC). The triggered interactions demonstrate unexpected/abnormal interaction between cloud resources when an undesired interaction is triggered (AT). Figure 7.21 demonstrates the prototype of this experiment.

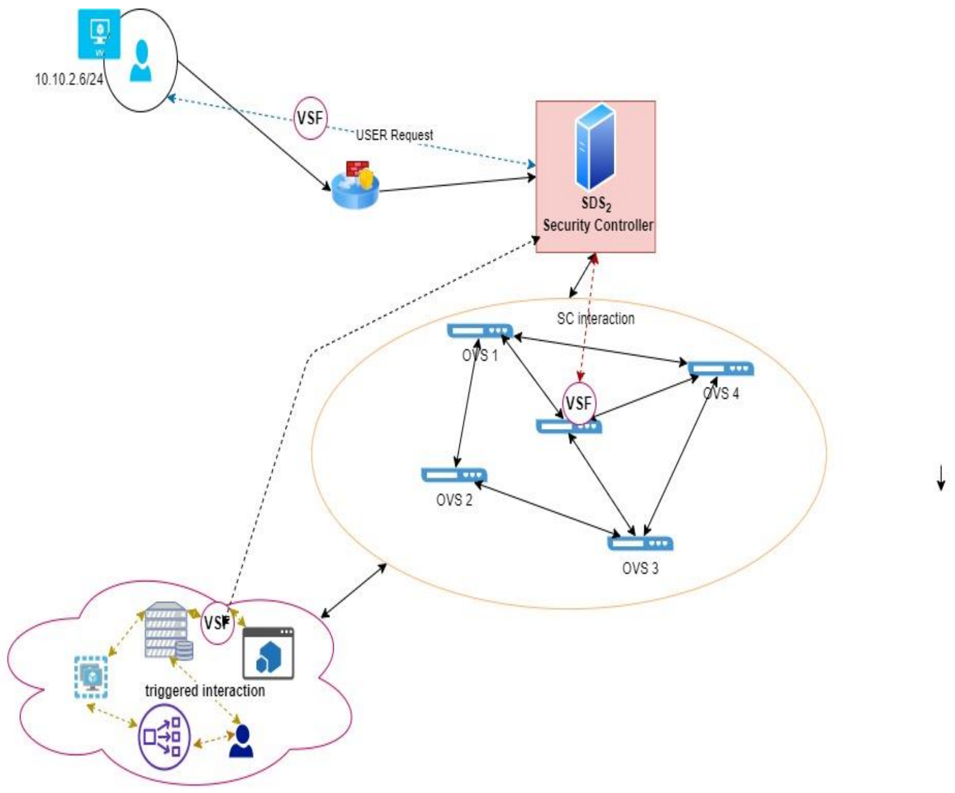
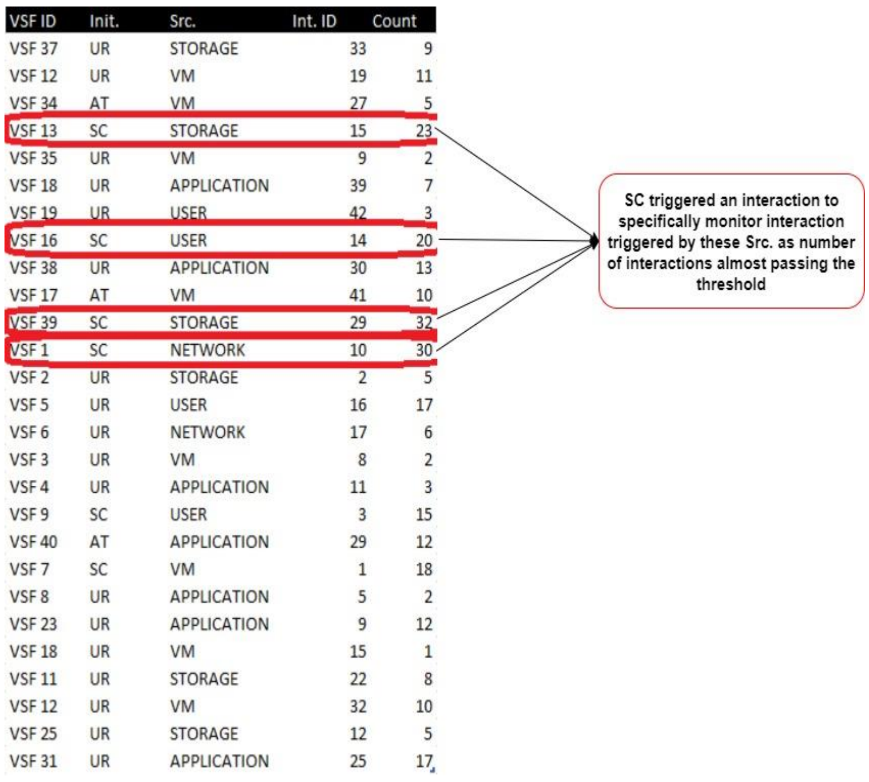


Figure 7.21 Presents the experiment prototype

Each experiment runs a diversity of interactions monitored by the SDS$_2$ security service. The assigned VSF can distinguish the initiator of the interaction and accordingly can speed up the process of monitoring according to previously stored data on the same initiator. The security controller is capable of not only discovering violations from input requests but also monitoring and discovering abnormal interactions between entities. Malicious activity can be considered as a DoS scenario attack scenario which is sending numerous interactions to cloud resources. After receiving a security alarm report from the VSF indicating abnormal interactions, the security controller triggers an internal interaction. In our design, the SDS$_2$ Security dynamically monitors different resources and their interactions. In this case, the security controller sets specific parameters (such as interaction threshold, suspicious behavior on accessing a specific resource, scheduled monitoring) to monitor the interaction of interest. Figure 7.22 demonstrates a list of interactions, assigned interactions, and their initiators.



VSF ID	Init.	Src.	Int. ID	Count
VSF 37	UR	STORAGE	33	9
VSF 12	UR	VM	19	11
VSF 34	AT	VM	27	5
VSF 13	SC	STORAGE	15	23
VSF 35	UR	VM	9	2
VSF 18	UR	APPLICATION	39	7
VSF 19	UR	USER	42	3
VSF 16	SC	USER	14	20
VSF 38	UR	APPLICATION	30	13
VSF 17	AT	VM	41	10
VSF 39	SC	STORAGE	29	32
VSF 1	SC	NETWORK	10	30
VSF 2	UR	STORAGE	2	5
VSF 5	UR	USER	16	17
VSF 6	UR	NETWORK	17	6
VSF 3	UR	VM	8	2
VSF 4	UR	APPLICATION	11	3
VSF 9	SC	USER	3	15
VSF 40	AT	APPLICATION	29	12
VSF 7	SC	VM	1	18
VSF 8	UR	APPLICATION	5	2
VSF 23	UR	APPLICATION	9	12
VSF 18	UR	VM	15	1
VSF 11	UR	STORAGE	22	8
VSF 12	UR	VM	32	10
VSF 25	UR	STORAGE	12	5
VSF 31	UR	APPLICATION	25	17

SC triggered an interaction to specifically monitor interaction triggered by these Src. as number of interactions almost passing the threshold

Figure 7.22 Presents various interaction types

As displayed in figure 7.22, SDS_2 service intelligently and can initiate interactions to protect resources. It can detect the attacks in a dynamic environment when interactions are triggered in different scenarios. In our proposed system, we demonstrate that the SDS_2 security controller not only is capable of monitoring requested interactions triggered by internal/external users (initiator UR) but also can monitor suspicious interactions among resources.

7.4.3 SDS_2 Platform setup

To demonstrate the working of the model, we implement all components of the proposed architecture in software. The SDS_2 service and its components are implemented in Java. In order to establish the SDS_2 security service model, we implement i) the SDS_2 security controller, which is a software written in java with various classes run on VM on top of OpenStack compute node, ii) a network of OpenFlow switches that connect system components, iii) a built-in database within the SDS_2 platform to store the security data including interaction parameters, cloud objects, and the VSFs information, and iv) the SDS_2 classes including components of NFV according to NFV MANO for initiating the VSF.

- All main SDS_2 components are built as software on one PC on Ubuntu 16.04 LTS with the following configuration: RAM: 16GB, CPU: Intel Core i7-8650U, Storage: 100 GB.
- A custom-built user interface is developed for the SDS_2 security controller to demonstrate results, including the device and controller level tables.
- A network of SDN devices is implemented within the system for the purpose of connectivity. Open vSwitches are deployed using OpenFlow protocol for connection.
- The SDS_2 service is a java-based platform developed in Eclipse IDE version: 4.14.0. The platform is deployed in an OpenStack cloud environment and connected to SDN devices. We implement the main components, including the

SDS_2 security controller, the Sec-Manage protocol, and Virtual Security Functions (VSFs).

  - The security controller consists of classes in charge of security controller functionality.
  - The virtual security function representation modules consist of classes to initiate and manage VSFs instances within the SDS_2 platform.
  - The southbound interface module includes classes to construct Sec-Manage messages, configuration, and forwarding tables within the system.

- The NFV MANO is deployed and integrated within the platform as java classes inspired by OpenBaton opensource software.
- The SDS_2 system uses a built-in database using MySQL, version: 5.7.32-0ubuntu0.16.04.1

7.5 SDS_2 Platform Performance Evaluation

In this section, we evaluate our proposed SDS_2 security model and platform based on two aspects: capability of the platform for provisioning on-demand security services for the purpose of security violation detection and prediction, and performance of the platform in relation to orchestration and configuration of virtual security functions- the processing time of security violation detection and prediction.

7.5.1 SDS_2 platform capability

We evaluate the SDS_2 service components and capabilities that contribute to the provision of on-demand security services to protect cloud resources. The following aspects are demonstrated: 1) Orchestration and configuration of on-demand virtual security functions, 2) Formation of interaction security violation detection and prediction. The SDS_2 service capability is divided into: 1) security controller level, 2) virtual security function level. The process is expressed as follow:

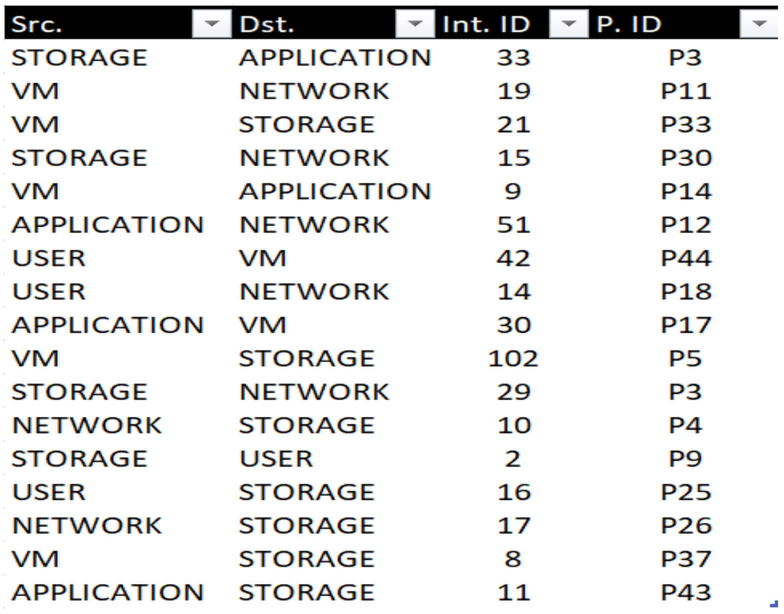
❖ Orchestration and functionalities - the Security Controller level

In this part, we implemented the significant components and functions required for two primary purposes: 1) defining interaction model as well as policy-based interaction model; 2) deploying new interaction proactive mechanisms and its algorithms.

At the Controller level, the SDS_2 security controller is capable of:

- Analyzing dynamic security policies according to features like entities, location, interactions, and assigning interaction parameter policies (figure 7.23)
- Handling dynamic changes of security policies (figure 7.24)
- Automating the construction of interaction-based virtual security functions (figure 7.25)
- Handling and detecting interaction security violation threats (figure 7-26a)
- Providing interaction-based security violation predictions for further security threats (figure 7-26b)

The SDS_2 security controller captures the interaction, initiates the VSFs to monitor the interaction, and performs detection and prediction techniques according to VSF result reports. As shown in figure 7.23, the security controller analyzes the security policies aligned with interaction parameters and assigns a policy identification for each triggered interaction to validate each interaction parameter's security rules. The P. ID refers to a table with stored policies for each entity according to their defined features.

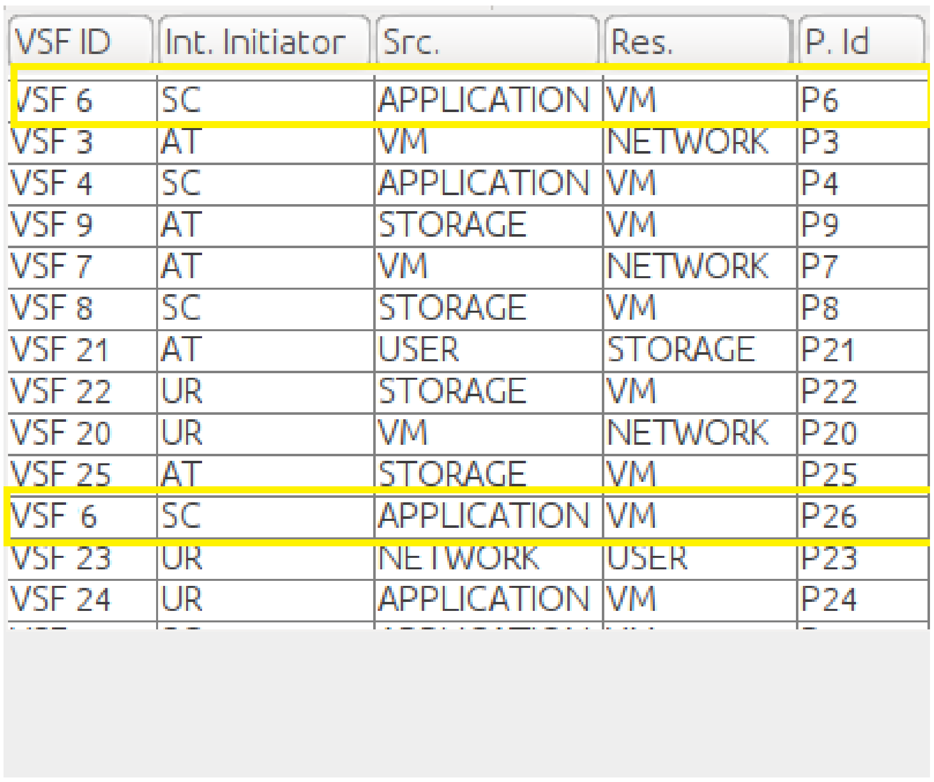


Src.	Dst.	Int. ID	P. ID
STORAGE	APPLICATION	33	P3
VM	NETWORK	19	P11
VM	STORAGE	21	P33
STORAGE	NETWORK	15	P30
VM	APPLICATION	9	P14
APPLICATION	NETWORK	51	P12
USER	VM	42	P44
USER	NETWORK	14	P18
APPLICATION	VM	30	P17
VM	STORAGE	102	P5
STORAGE	NETWORK	29	P3
NETWORK	STORAGE	10	P4
STORAGE	USER	2	P9
USER	STORAGE	16	P25
NETWORK	STORAGE	17	P26
VM	STORAGE	8	P37
APPLICATION	STORAGE	11	P43

Figure 7.23 Presents the assigned PID according to each entity (Src., Dst.)

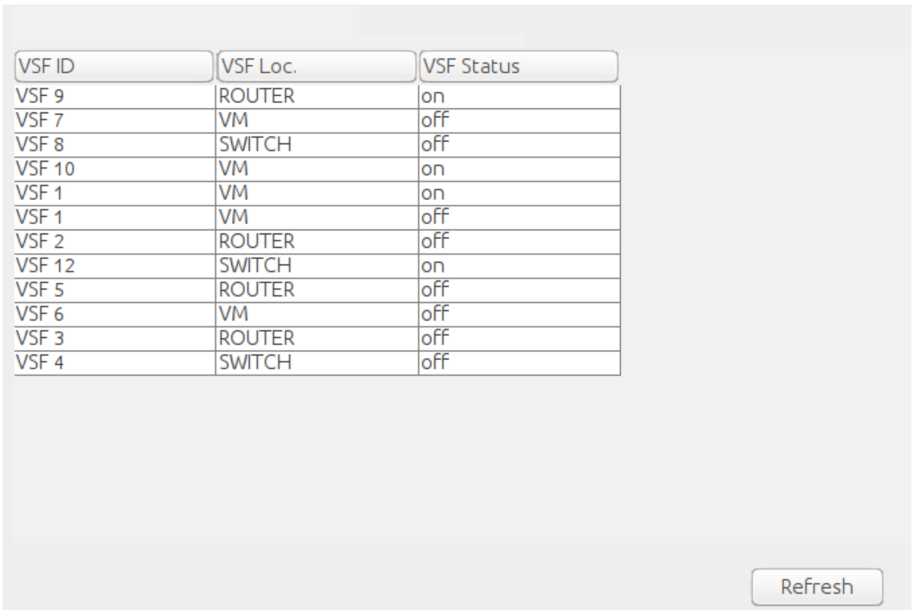
Figure 7.24 represents the ability of the security controller to handle dynamic changes in security policies. The security controller re-orchestrates the assigned security policies for entities with any changes during an interaction based on interaction parameters. Figure 7.25 demonstrates the security controller interface that shows the VSF functions. The primary purpose of the SDS$_2$ platform is to detect and predict interaction-based security violations. The Security controller is in charge of performing ISVDP algorithms to detect and predict the present and possible future attacks. Figures 7.26a and 7.26.b demonstrate captured interactions and detection and prediction results (expressed in chapter 4). The detection and the prediction rely on VSF sec_messages to the security controller reporting on the monitored interaction status.

As you can see in Image 7.26a, the VSF 9 denotes a USER request for NETWORK resource on the TENANT level with extracted READ and MODIFY actions. The request is dynamically analyzed by the security controller and, based on the applied policy, is detected as a violation for requesting beyond what is allowed. The figure shows the detection analysis of various interactions at the security controller level. The results present the state of interaction decided by the security controller after monitoring.



VSF ID	Int. Initiator	Src.	Res.	P. Id
VSF 6	SC	APPLICATION	VM	P6
VSF 3	AT	VM	NETWORK	P3
VSF 4	SC	APPLICATION	VM	P4
VSF 9	AT	STORAGE	VM	P9
VSF 7	AT	VM	NETWORK	P7
VSF 8	SC	STORAGE	VM	P8
VSF 21	AT	USER	STORAGE	P21
VSF 22	UR	STORAGE	VM	P22
VSF 20	UR	VM	NETWORK	P20
VSF 25	AT	STORAGE	VM	P25
VSF 6	SC	APPLICATION	VM	P26
VSF 23	UR	NETWORK	USER	P23
VSF 24	UR	APPLICATION	VM	P24

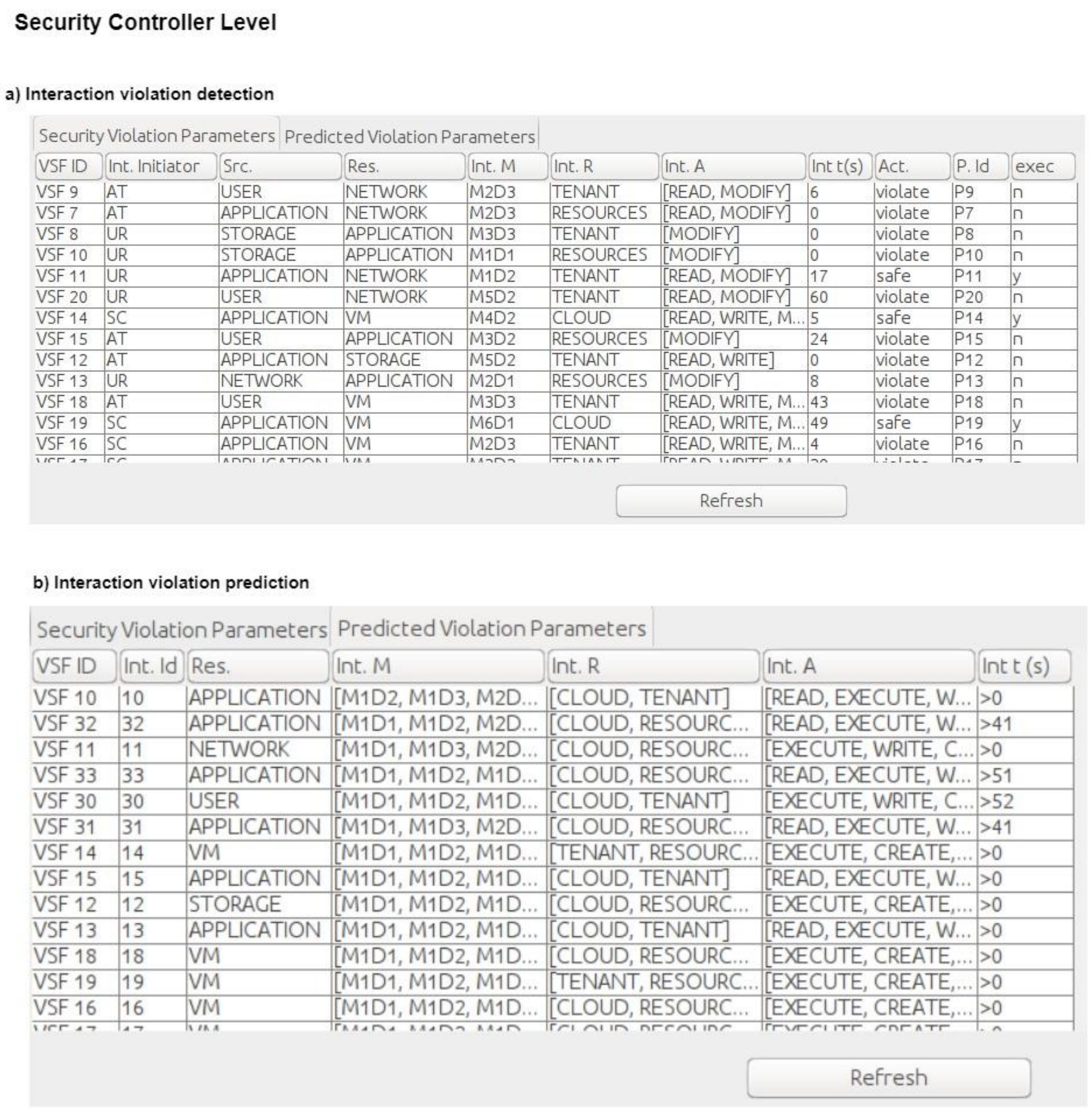
Figure 7.24 Dynamic changes of Policies



VSF ID	VSF Loc.	VSF Status
VSF 9	ROUTER	on
VSF 7	VM	off
VSF 8	SWITCH	off
VSF 10	VM	on
VSF 1	VM	on
VSF 1	VM	off
VSF 2	ROUTER	off
VSF 12	SWITCH	on
VSF 5	ROUTER	off
VSF 6	VM	off
VSF 3	ROUTER	off
VSF 4	SWITCH	off

Figure 7.25 SDS$_2$ automatically initiates VSFs to monitor interactions

As demonstrated in figure 7.26.b, each row defines predicted future violation of presented interactions according to their interaction parameters. Each entry results from an automatic analysis of potential interaction violations concerning the value of interaction parameters. For example, the first row shows potential interaction violation as a set of variables for each interaction parameter against the targeted resource (Application).



Security Controller Level

a) Interaction violation detection

Security Violation Parameters | Predicted Violation Parameters

VSF ID	Int. Initiator	Src.	Res.	Int. M	Int. R	Int. A	Int t(s)	Act.	P. Id	exec
VSF 9	AT	USER	NETWORK	M2D3	TENANT	[READ, MODIFY]	6	violate	P9	n
VSF 7	AT	APPLICATION	NETWORK	M2D3	RESOURCES	[READ, MODIFY]	0	violate	P7	n
VSF 8	UR	STORAGE	APPLICATION	M3D3	TENANT	[MODIFY]	0	violate	P8	n
VSF 10	UR	STORAGE	APPLICATION	M1D1	RESOURCES	[MODIFY]	0	violate	P10	n
VSF 11	UR	APPLICATION	NETWORK	M1D2	TENANT	[READ, MODIFY]	17	safe	P11	y
VSF 20	UR	USER	NETWORK	M5D2	TENANT	[READ, MODIFY]	60	violate	P20	n
VSF 14	SC	APPLICATION	VM	M4D2	CLOUD	[READ, WRITE, M...	5	safe	P14	y
VSF 15	AT	USER	APPLICATION	M3D2	RESOURCES	[MODIFY]	24	violate	P15	n
VSF 12	AT	APPLICATION	STORAGE	M5D2	TENANT	[READ, WRITE]	0	violate	P12	n
VSF 13	UR	NETWORK	APPLICATION	M2D1	RESOURCES	[MODIFY]	8	violate	P13	n
VSF 18	AT	USER	VM	M3D3	TENANT	[READ, WRITE, M...	43	violate	P18	n
VSF 19	SC	APPLICATION	VM	M6D1	CLOUD	[READ, WRITE, M...	49	safe	P19	y
VSF 16	SC	APPLICATION	VM	M2D3	TENANT	[READ, WRITE, M...	4	violate	P16	n

b) Interaction violation prediction

Security Violation Parameters | Predicted Violation Parameters

VSF ID	Int. Id	Res.	Int. M	Int. R	Int. A	Int t (s)
VSF 10	10	APPLICATION	[M1D2, M1D3, M2D...	[CLOUD, TENANT]	[READ, EXECUTE, W...	>0
VSF 32	32	APPLICATION	[M1D1, M1D2, M2D...	[CLOUD, RESOURC...	[READ, EXECUTE, W...	>41
VSF 11	11	NETWORK	[M1D1, M1D3, M2D...	[CLOUD, RESOURC...	[EXECUTE, WRITE, C...	>0
VSF 33	33	APPLICATION	[M1D1, M1D2, M1D...	[CLOUD, RESOURC...	[READ, EXECUTE, W...	>51
VSF 30	30	USER	[M1D1, M1D2, M1D...	[CLOUD, TENANT]	[EXECUTE, WRITE, C...	>52
VSF 31	31	APPLICATION	[M1D1, M1D3, M2D...	[CLOUD, RESOURC...	[READ, EXECUTE, W...	>41
VSF 14	14	VM	[M1D1, M1D2, M1D...	[TENANT, RESOURC...	[EXECUTE, CREATE,...	>0
VSF 15	15	APPLICATION	[M1D1, M1D2, M1D...	[CLOUD, TENANT]	[READ, EXECUTE, W...	>0
VSF 12	12	STORAGE	[M1D1, M1D2, M1D...	[CLOUD, RESOURC...	[EXECUTE, CREATE,...	>0
VSF 13	13	APPLICATION	[M1D1, M1D2, M1D...	[CLOUD, TENANT]	[READ, EXECUTE, W...	>0
VSF 18	18	VM	[M1D1, M1D2, M1D...	[CLOUD, RESOURC...	[EXECUTE, CREATE,...	>0
VSF 19	19	VM	[M1D1, M1D2, M1D...	[TENANT, RESOURC...	[EXECUTE, CREATE,...	>0
VSF 16	16	VM	[M1D1, M1D2, M1D...	[CLOUD, RESOURC...	[EXECUTE, CREATE,...	>0

Figure 7.26 demonstrates result after SDS$_2$ runs ISVDP algorithms: a) shows results of ISVD algorithm for various simulated interactions within the system, b) presents results after running ISVP algorithm

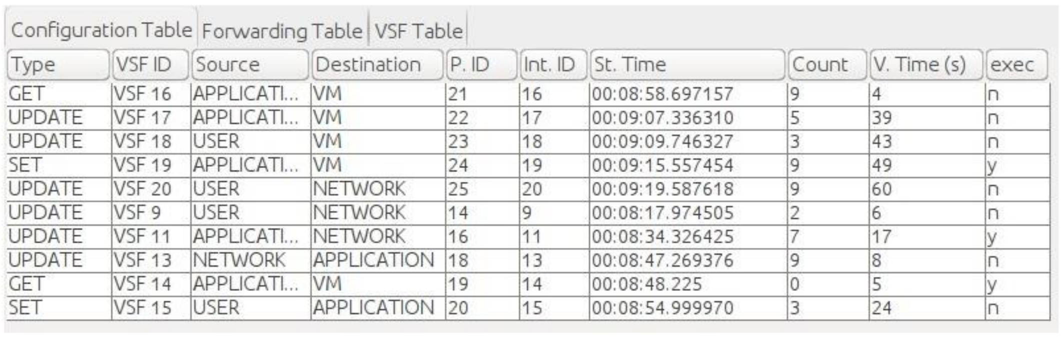
❖ Orchestration and configuration – At Virtual Security Function level

At this level, the SDS$_2$ security controller configures the virtual security functions through Sec-Manage protocol and security messages. The virtual security functions are configured based on interaction parameters. They include information regarding the configuration table, forwarding table, and VSF services. The VSFs can be configured to monitor the interactions (figure 7.27a). It forwards data back to the security controller for further decisions (figure 7.27b). The VSFs table lists VSFs functions and their associated status and location to monitor the interaction (figure 7.27c). The config and forward table contain parameters required to configure a VSF and define forwarding action in relation to an interaction—each parameter is described in detail in chapter five.

As you can see, figure 7.27 shows the security service table entries for each assigned VSF. For example, VSF 7 shows that VSF function is still running with remaining time to live equal to 50s allocated in a VM to monitor interactions.

Virtual Security level

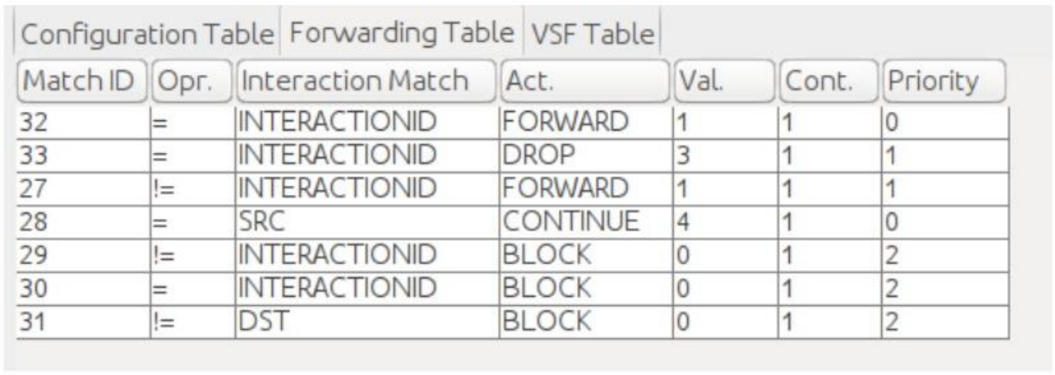
a) Config Table



Configuration Table | Forwarding Table | VSF Table

Type	VSF ID	Source	Destination	P. ID	Int. ID	St. Time	Count	V. Time (s)	exec
GET	VSF 16	APPLICATI...	VM	21	16	00:08:58.697157	9	4	n
UPDATE	VSF 17	APPLICATI...	VM	22	17	00:09:07.336310	5	39	n
UPDATE	VSF 18	USER	VM	23	18	00:09:09.746327	3	43	n
SET	VSF 19	APPLICATI...	VM	24	19	00:09:15.557454	9	49	y
UPDATE	VSF 20	USER	NETWORK	25	20	00:09:19.587618	9	60	n
UPDATE	VSF 9	USER	NETWORK	14	9	00:08:17.974505	2	6	n
UPDATE	VSF 11	APPLICATI...	NETWORK	16	11	00:08:34.326425	7	17	y
UPDATE	VSF 13	NETWORK	APPLICATION	18	13	00:08:47.269376	9	8	n
GET	VSF 14	APPLICATI...	VM	19	14	00:08:48.225	0	5	y
SET	VSF 15	USER	APPLICATION	20	15	00:08:54.999970	3	24	n

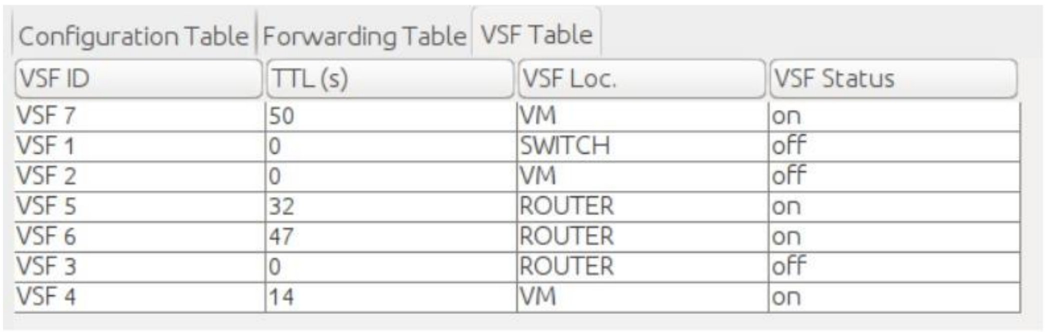
b) Forward Table



Configuration Table | Forwarding Table | VSF Table

Match ID	Opr.	Interaction Match	Act.	Val.	Cont.	Priority
32	=	INTERACTIONID	FORWARD	1	1	0
33	=	INTERACTIONID	DROP	3	1	1
27	!=	INTERACTIONID	FORWARD	1	1	1
28	=	SRC	CONTINUE	4	1	0
29	!=	INTERACTIONID	BLOCK	0	1	2
30	=	INTERACTIONID	BLOCK	0	1	2
31	!=	DST	BLOCK	0	1	2

c) Security Services Table



Configuration Table | Forwarding Table | VSF Table

VSF ID	TTL (s)	VSF Loc.	VSF Status
VSF 7	50	VM	on
VSF 1	0	SWITCH	off
VSF 2	0	VM	off
VSF 5	32	ROUTER	on
VSF 6	47	ROUTER	on
VSF 3	0	ROUTER	off
VSF 4	14	VM	on

Figure 7.27 SDS$_2$ security functions

7.5.2 SDS$_2$ platform – Performance

In this section, we carry out service provisioning tasks and evaluate the proposed SDS$_2$ performance through two performance measures: security orchestration time and security reaction time. Security orchestration time is referred to as the time required for the security controller to orchestrate and configure VSFs according to interaction parameters. Security reaction time is the security action time that measures the time to detect security violations and predict future interaction violation parameters. Figure 7.28 signifies the SDS$_2$ service timing diagram. As demonstrated in the figure, the orchestration time is T3, which is comprised of T1 and T2—the total security reaction time measured based on T4, T5, T6, T7, T8, and T9. The T6 demonstrates the time that SDS$_2$ system requires to detect violations using the algorithms.

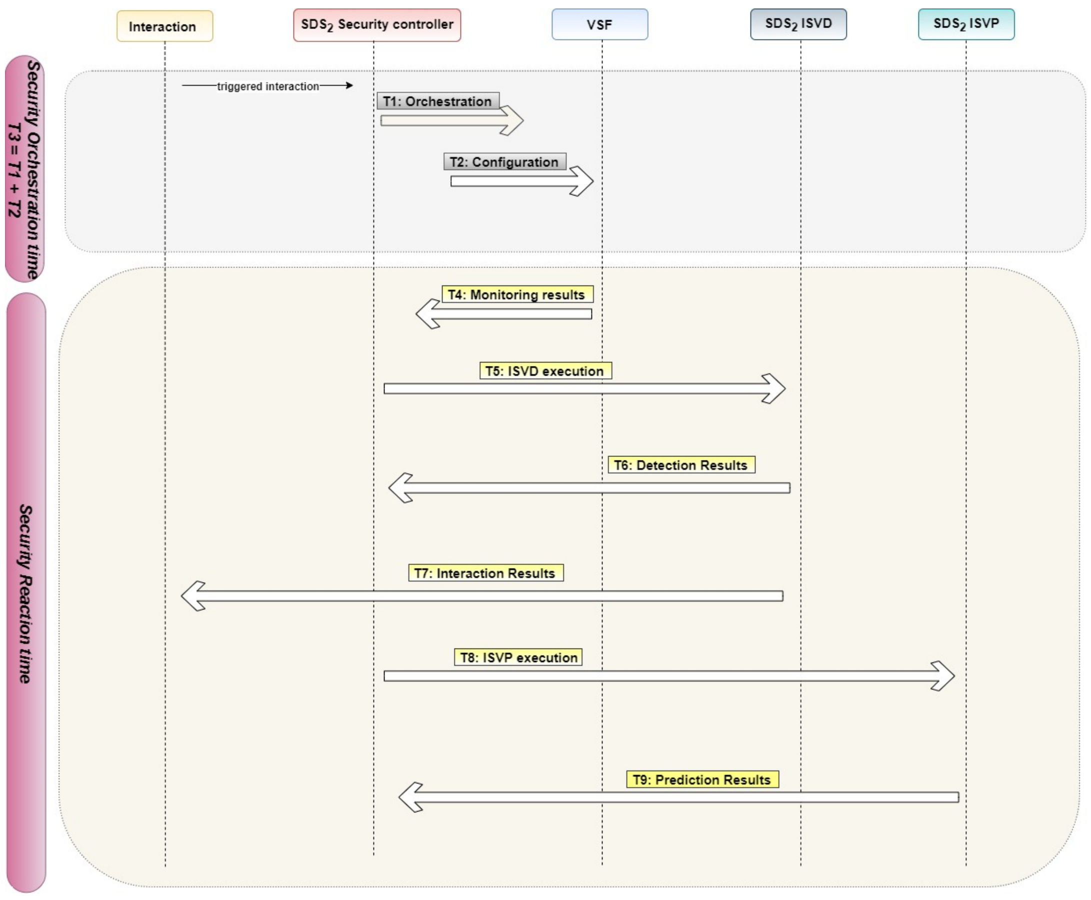


Figure 7.28 SDS$_2$ time diagram

The orchestration times consider both orchestration of VSFs and the time it requires to configure the functions. We examined the efficiency of the SDS_2 service through the parameters mentioned above. During the orchestration phase, the SDS_2 platform analyzes the interaction and triggers new or existing VSFs to monitor the interaction and configure/update the VSFs according to validate interaction parameters. To evaluate it, we consider two prominent cases 1) orchestrate new VSFs, 2) re-orchestrate existing VSFs. In the first case, system orchestration time increased as this requires the initiation of the new VSF. The reason relies on the time the security controller requires to initiate, assign the resources, and run a new VSF. The SDS_2 security controller needs to assign new resources to the VSFs, which takes a long time to communicate with VIM to get the VSF resources. We examine the security orchestration of the system in both cases. A considerable number of interactions have been sent to the security controller. The interaction can be of any type of interaction. As displayed in figure 7.29, the time required for the cases orchestrations increases with a rising number of interactions within the system. Remarkably, the maximum amount of time for responding to 60 concurrent requested interactions is slightly above the 4s while this amount is less for re-orchestrating the existing VSFs. This is because case two 1) reuses the existing VSFs which already have assigned resources, and 2) reduces the time to configure the VSFs for the same interaction. Case one does not consider these factors; hence it requires to orchestrate and configure the new VSF for each incoming interaction.

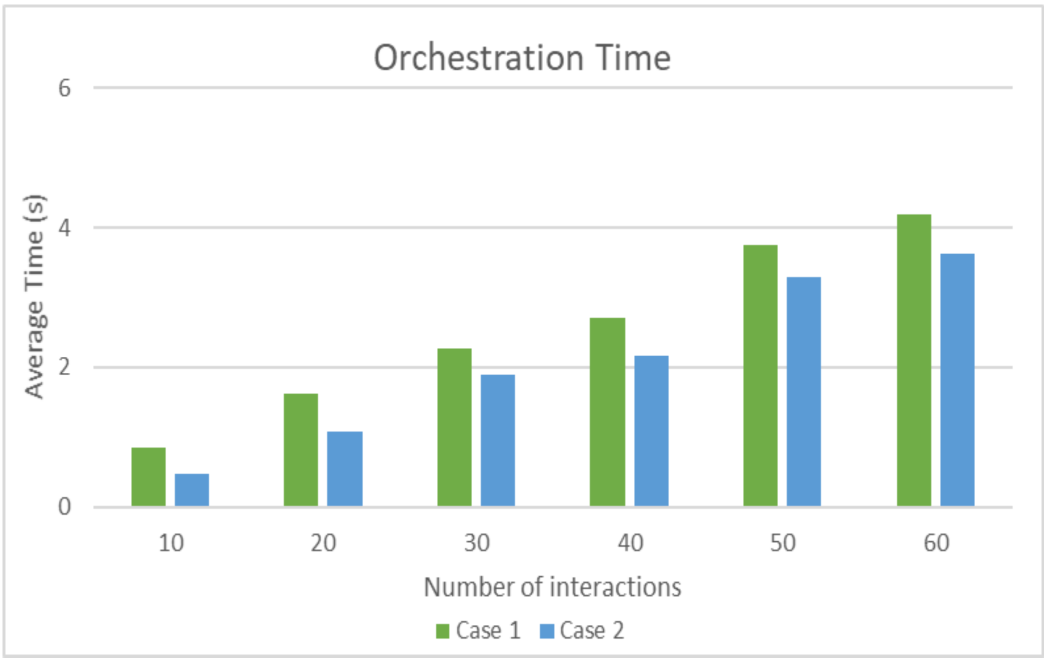


Figure 7.29 Orchestration time Case 1 and Case 2

The security reaction time is expressed as when the SDS_2 service requires to respond to an interaction. The response time measures are based on two main factors: 1) detection time and 2) prediction time. To evaluate the performance of the SDS_2 service, we consider two cases: 1) non-smart SDS_2 service, and 2) smart SDS_2 service. We run numerous interactions for each case. The first case demonstrates the required time to detect the security violations without previous prediction parameters (Figure 7.30).

In this case, the SDS_2 service each time measures the average detection time for various interactions without considering predicted potential interaction parameters violations. The second case enables the SDS_2 service to reduce the time by accessing prediction parameters (Figure 7.31). The detection processing time will be reduced as a result of considering prediction parameters as the first stage of detection for an interaction. The SDS_2 shows a considerable decrease in detection processing time where predicted interaction parameters reflect future threats for an incoming interaction. In such cases, the SDS_2 security services first will deliberate on the detection process considering previously predicted violation interaction parameters. So, it optimizes the processing time during the detection period by considering previously predicted parameters. As demonstrates in Figures 7.30 and 7.31, the detection process time that the SDS_2 requires to detect security violations within the system reaches its peak by an increased number of interactions to 60.

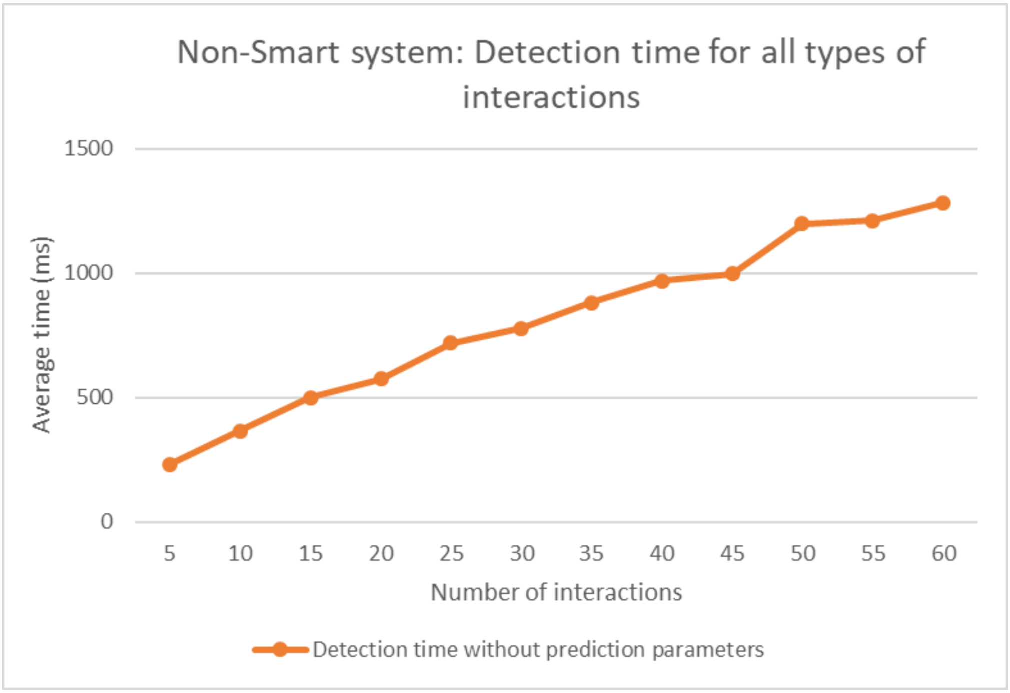


Figure 7.30 Non-Smart SDS_2 service

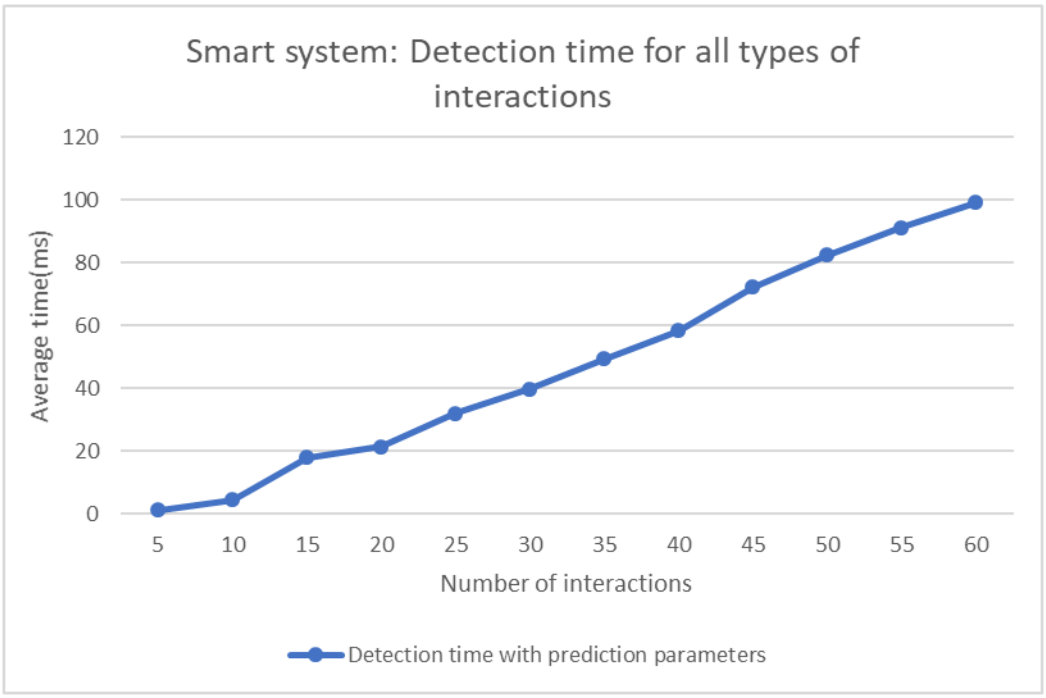


Figure 7.31 Smart SDS$_2$ service

As shown in Figure 7.32, we can observe a performance improvement of upwards of 90%. With the lower number of interactions, the performance improvement is around 99%, and as the number of interactions increase, we can see the improvement stabilize upwards of 90%. This is predictable as with an increase in the number of interactions, so does the average detection time, however using the Smart SDS$_2$ service, we can have a better prediction model and maintain a steady performance improvement over time.

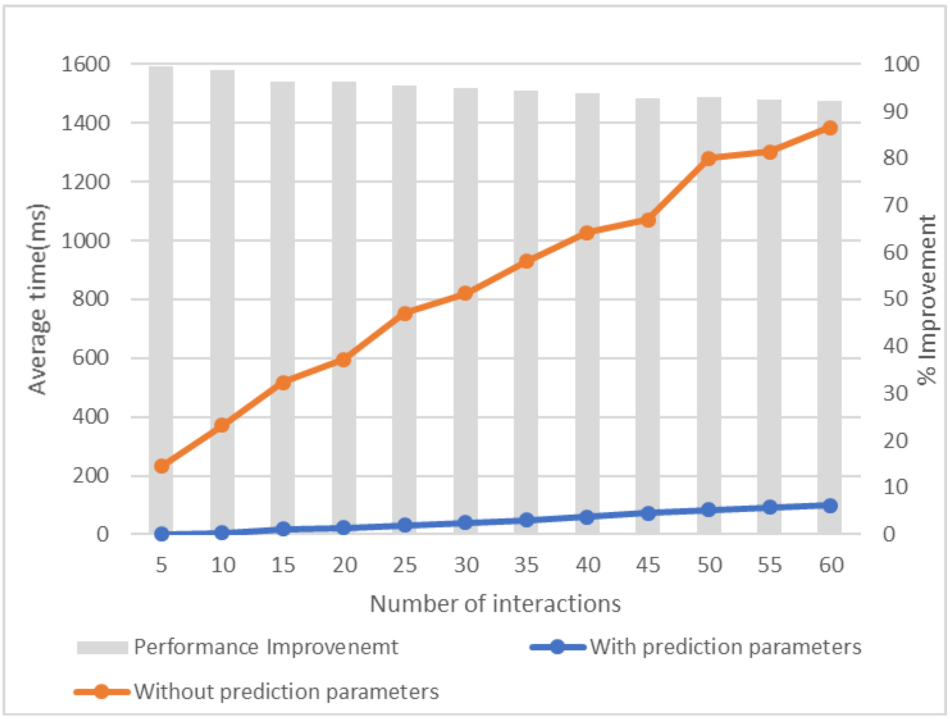


Figure 7.32 SDS$_2$ service - performance improvement

As illustrated in Figure 7.33, the SDS$_2$ service spent more time during the detection discovery phase for interactions triggered by users. The discovery time for SDS$_2$ related to user interactions maximizes as interactions reach 90 concurrent requested interactions compared to other types. In the other cases, this time is less as SDS$_2$ uses existing stored data, its local security DB. The SDS$_2$ service requires more computation time to process a user requests since new data has been presented which there is no record in DB for involved entities.

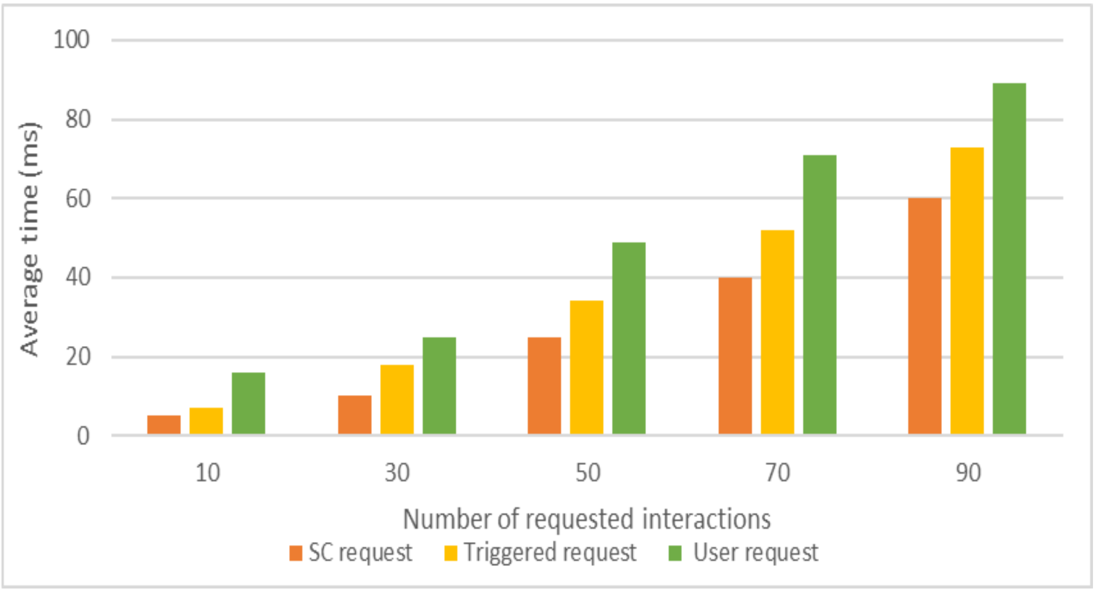


Figure 7.33 SDS$_2$ service performance evaluation - in case of different simulated interactions

7.6 Summary

This chapter has introduced our SDS$_2$ platform to provide on-demand security services to protect cloud resources. The SDS$_2$ model aims to introduce a proactive system to limit virtual environment challenges on provisioning and programming underlying virtual security functions. We attempt to incorporate the SDS$_2$ security model within an integrated cloud/SDN/NFV system for dynamic protection against interaction security violations. We have deployed the proposed model's primary functionality and have enabled the SDS$_2$ service to control and manage virtual security functions according to their purpose. We discuss several experiments to demonstrate SDS$_2$ model functionality and to provide the proposed model operation in provisioning on-demand virtual security functions. We present the feasibility and efficiency of the proposed model through the design and implementation of the SDS$_2$ platform. We evaluate the system performance through numerous simulated interactions.

Chapter 8

Conclusion and Future Work

8.1 Research Remarks

This chapter summarizes this research and outlines the significant contributions. We suggest future work in association with the research achievements.

Cloud computing has evolved into a key structure for IT industries for providing users with on-demand services. Cloud architecture enables users to access cloud services over the Internet at any time regardless of their location through application software like web browsers. Rapidly cloud services and their on-demand virtual functions have become an indispensable technology involved in many aspects of human life, educational system, healthcare, industry, government, and social enterprises. However, the cloud environment adoption and its services have been slowly moving forward as it becomes more vulnerable to traditional and new security threats related to its structure and elements. Moreover, the integration of new emerging technologies like software-defined networking and Network Function Virtualization provoked security cloud services and their virtual functions. According to the National Institute of Standards and Technologies (NIST), **security**, portability, and interoperability have been declared the main obstacles to adopting the cloud environment.

Cloud, SDN, and NFV technologies and their associated software-defined infrastructures rely on virtualization technology to provide their virtual resources and offer them as services to users. However, the complexity of security issues in virtual cloud infrastructure is more complicated than traditional infrastructure since resources/functions are shared and virtualized between numerous cloud users. In multi-tenant cloud architecture, isolation is introduced as a crucial concept for both security and infrastructure management. Isolation should be considered at functional entity levels and appropriate abstraction levels of the infrastructure. However, virtual boundaries amongst cloud virtual functions/components are not always well defined and rather often undefined, and hence they are not visible/controllable by security mechanisms or cloud providers.

In this research, we discovered a number of significant security challenges of the current cloud and its integrated technologies. The first challenge is finding effective mechanisms for constructing dynamic isolation boundaries for securing cloud assets at different cloud infrastructure levels. This challenge prompted the need to provide overall visibility on virtual boundaries within a cloud infrastructure. The second challenge is to deliver a competent, proactive security technique to detect and predict security violations to protect cloud resources. The third issue is to automate and provision virtual security functions whenever they are required.

To address the challenges, we propose a software-defined security service model and its associated techniques to provision on-demand security service to protect cloud resources.

In particular, on dynamic connectivity and networking among virtual security functions, software-defined networking (SDN) has transformed the physical underlying network infrastructure into programmable and virtualized networks. The SDN controller enables automatic connectivity among virtual network functions through its logically centralized overview of network function dynamics. However, using SDN in cloud security is still not common as it is still developing and facing security challenges. This research aims to use SDN to enable connectivity among virtual security functions through a direct connection between the security controller and VSFs.

We introduce a new technique to provide dynamic visibility on security boundaries for cloud entities during a triggered request on the dynamic construction of security isolation. For this purpose, we investigate various algorithms and mechanisms for security isolation.

On security violation detection and prediction mechanism, we try to deliver an exceptional, innovative technique when an interaction occurs within the system. To achieve that goal, we explore security breaches and methods in different domains for security violation services.

As for communication and management protocol, it is worth noting that virtual security functions (VSFs) are not routing network functions where heavy protocols for programming network flows in virtual network functions are not entirely applicable to virtual security functions. In a security model less, an effort has been made to address these challenges. This study investigates the deployment of a new simple protocol to transfer essential required security parameters.

Regarding the creation and orchestration of virtual functions, Network function virtualization (NFV) aims to virtualize an entire class of network component functions using virtualization technologies. NFV enables network functions to be realized and executed as software instances in a VM on single or multiple hosts instead of customized hardware appliances. However, efforts to utilize virtual functions in creating efficient but straightforward virtual security functions are limited. We explore network function virtualization mechanism to deploy virtual security functions with simple functionality and capability to be used in a security model in a cloud environment.

The contribution of this research has been summarized as follows.

We proposed a software-defined security service model that enables the provision of on-demand security services via orchestration of virtual security functions over cloud infrastructure. This model enables the programmability of numerous virtual security functions for provisioning on-demand security services. The model consists of novel security violation techniques to construct dynamic security boundaries related to interaction among cloud entities.

We introduced an innovative policy-based interaction model that enables the dynamic construction of security boundaries for cloud entities involved during a real-time interaction. The model governs interaction security among entities in a virtual cloud environment. The model provides a framework for incorporating system security policies and entity constraints in constructing interaction boundaries and defining a security dictionary of expected/unexpected cloud entities expected/unexpected behavior when they access resources in the cloud environment.

We presented new algorithms and techniques to detect and predict security violations during a triggered interaction. We deploy an automatic detection and prediction algorithm called ISVDP to identify security breaches related to interaction parameters. The algorithm also maps out possible future attacks based on expected violations of the currently defined interaction parameters.

We proposed a novel control and management protocol for programming virtual security functions. The proposed protocol enables direct communication between the SDS_2 security controller and its VSFs. The main purpose is to transfer the parameters pertinent to the security aspects of objects' interaction, between a VSF and the security controller, to monitor an interaction's parameters to detect and predict security violations.

We proposed a software-defined security service system that can be a part of cloud infrastructure protecting virtual resources/functions. The proposed architecture via its centralized security controller enables dynamic programmability of the underlying virtual security function. The virtual security functions can be controlled, orchestrated, managed, and configured by centralizing security control with overall visibility on entities security boundaries.

We design and implement the software-defined security platform to present its capability and performance in provisioning on-demand virtual security functions to protect the cloud resources. The proposed platform demonstrates a new software-defined security system for integrating cloud, SDN, and NFV concepts in cloud security and a specific method to detect and predict security violations.

On security connectivity between the security controller and virtual security functions, we adopted SDN functionalities to provide connectivity. We demonstrated the use of a software-defined networking domain in security through the SDS_2 model.

On virtual security function, we introduce a simple VSF to monitor interactions between cloud resources. The security function enhanced on-demand security monitoring within cloud infrastructure due to the dynamicity of interactions.

On control and management protocol, we designed and deployed a novel, simple protocol as Sec-Manage protocol, specifically designed to transfer interaction parameters and policies between the security controller and virtual security functions.

On security isolation, we introduced a new isolation domain via interaction. A novel interaction model governs the protection of cloud resources through dynamic security interaction boundaries. We presented major interaction parameters for the construction of security boundaries during triggered interactions.

We design and deploy an innovative policy-based interaction model and its associated techniques and an algorithm to construct a proactive security mechanism. We introduced an original security violation detection and prediction method according to interaction parameters and policies governing the interaction and entities.

In brief, we trust that this research thesis delivers an affirmative response to the posed question in chapter 1, "*How to secure and protect cloud resources against security isolation breaches using new technologies based on SDN/NFV, and can the* *proposed model be realized in the practical environment?*". Our security model and its innovative elements and algorithms can be used to provide on-demand security service to protect cloud's resources.

This research's novelty lies in its novel software-defined security service platform for provisioning on-demand security services, its unique mechanisms in constructing dynamic security boundaries, and its innovative method in detecting and predicting security violations in relation to the proposed policy-based interaction model. The proposal includes a novel software-defined security architecture that is constructed using SDN and NFV technologies. The proposed architecture introduces a logically centralized security controller with overall visibility of security boundaries.

The significance of this study is that it allows the orchestration and programmability of virtual security functions in provisioning on-demand security service to protect cloud resources. This research enables i) cloud providers to dynamically provide security interaction isolation to protect cloud resources, ii) cloud providers to develop innovative 0n-demand virtual security functions to improve dynamicity of security monitoring over virtual cloud environments iii) clouds and tenants security developers to enhance security methods in detecting and predicting security violation over virtual functions, iv) security admins to dynamically program and orchestrate underlying virtual security function against any types of interaction violation breaches.

8.2 Future Work

Usage of SDN and NFV in cloud security is still a developing research area. In this section, we outline some potential future works and research directions for cloud security.

This research investigates technologies, security architectures, virtual function capabilities, security protocols, security isolation mechanisms, and programmable and orchestration of virtual security functions within a cloud infrastructure. This thesis opens up new research directions in integrating SDN and NFV in the cloud environment to provide on-demand security services. Even though this research presented significant outcomes, there remain several limitations.

The SDS_2 security platform can be considered a crucial platform for future investigation on new mechanisms associated with QoS-driven network among virtual security functions and security platform elements. This research mainly focuses on constructing a dynamic but straightforward network security connection between VSF and SC. In the future, billions of virtual security functions can transfer security data and messages, which may cause a heavy load on network transportation. In the future, we consider the QoS and design QoS techniques to prevent the above-mentioned issue and enhance software-defined security service orchestration in provisioning on-demand virtual security functions.

Currently, the proposed SDS_2 security model focuses on constructing VSFs according to network function virtualization concepts. The VSFs are implemented from pre-defined scripts/templates written in Jason/TOSCA templates. The security controller places them according to the placement of triggered interaction. In future work, we plan to work on allocation optimization of VSFs within a cloud infrastructure. We plan to investigate intelligent location discovery algorithms and integrate them within the security controller.

The proposed model includes the Sec-Manage protocol to control and manage virtual security functions. The Sec-Manage transfers specific interaction parameters between the security controller and VSFs. Currently, protocol mainly focuses on forwarding a limited number of specific parameters and functioning the behavior of VSFs to monitor the interactions. We can develop this further to includes QoS-specific parameters considering the limitation of virtual security functions and security controllers.

Currently, the SDS_2 security model concentrates mostly on validating the security model's efficiency within a single cloud node. However, we plan to test the model in an integrated cloud infrastructure, including various nodes and domains.

An interaction can be considered as a simple/complex interaction. As previously discussed in chapter 4, an interaction can involve numerous internal interactions before achieving its goal, which can involve a hierarchy of dependent resources. However, in this research, the proposed SDS_2 security model emphasizes only simple interaction among two resources. The security system bypasses the complexity of the interaction in terms of nested internal interactions and dependable resources. We are currently investigating interaction complexity in terms of nested interactions and plan to enhance our policy-based interaction mechanism which will satisfy such interactions within the virtual cloud environment.

In our proposed interaction model, the relational position parameter is entangled with role-based security policies. Role-based policies are considered as significant access policies in each security domain. However, they are not easy to be calculated in a complex and dynamic environment such as a cloud. In our study, we mainly focus on three main role-based policy levels. We consider improving the complexity of our security model by deploying complex dynamic role-based security policies. We aim to use nested role-based security policies to enhance the discovery time in our security model.

Bibliography

[1] S. Jeuk, G. Salgueiro, and F. J. Baker, "Cloud provider, service, and tenant classification in cloud computing," ed: Google Patents, 2017.

[2] L. Schubert, "ADVANCES in CLOUDS Report from the CLOUD Computing Expert Working Group," 2012.

[3] F. Liu *et al.*, "NIST cloud computing reference architecture," *NIST special publication,* vol. 500, no. 2011, pp. 1-28, 2011.

[4] (2018). *AUSTCYBER CYBER SECURITY SECTOR – COMPETITIVENESS PLAN*. [Online] Available: file:///C:/Users/11822400/Downloads/CYB7900_SCP_digital_and_print_Complete_V12_FILM_web.pdf

[5] H. M. K. Al Nasseri and I. M. M. Duncan, "Investigation of Virtual Network Isolation security in Cloud computing: data leakage issues," 2016.

[6] B. P. Rimal and M. Maier, "Workflow scheduling in multi-tenant cloud computing environments," *IEEE Transactions on parallel and distributed systems,* vol. 28, no. 1, pp. 290-304, 2016.

[7] M. Almorsy, J. Grundy, and I. Müller, "An analysis of the cloud computing security problem," *arXiv preprint arXiv:1609.01107,* 2016.

[8] M.-M. Bazm, M. Lacoste, M. Südholt, and J.-M. Menaud, "Isolation in cloud computing infrastructures: new security challenges," *Annals of Telecommunications,* vol. 74, no. 3, pp. 197-209, 2019.

[9] K. Z. Bijon, R. Krishnan, and R. Sandhu, "A formal model for isolation management in cloud infrastructure-as-a-service," in *International Conference on Network and System Security*, 2015: Springer, pp. 41-53.

[10] T. Madi *et al.*, "ISOTOP: auditing virtual networks isolation across cloud layers in OpenStack," *ACM Transactions on Privacy and Security (TOPS),* vol. 22, no. 1, pp. 1-35, 2018.

[11] M. A. Babar and B. Ramsey, "Understanding container isolation mechanisms for building security-sensitive private cloud," *The University of Adelaide, Australia,* 2017.

[12] J.-M. Kang, T. Lin, H. Bannazadeh, and A. Leon-Garcia, "Software-defined infrastructure and the SAVI testbed," in *International Conference on Testbeds and Research Infrastructures*, 2014: Springer, pp. 3-13.

[13] J. C. Patni, S. Banerjee, and D. Tiwari, "Infrastructure as a Code (IaC) to Software Defined Infrastructure using Azure Resource Manager (ARM)," in *2020 International Conference on Computational Performance Evaluation (ComPE)*, 2020: IEEE, pp. 575-578.

[14] Research, "The Future of Infrastructure and the Software-Defined Organization," 451 Research, 2018. [Online]. Available: https://www.suse.com/media/white-paper/future%20of%20infrastructure%20and%20software%20defined%20organization.pdf

[15] (2015). *Software Defines the Infrastructure of a Future-Ready Enterprise*. [Online] Available: https://i.dell.com/sites/csdocuments/Shared-Content_data-Sheets_Documents/en/Software_defines_the_future_infrastructure.pdf

[16] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE,* vol. 103, no. 1, pp. 14-76, 2014.

[17] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications surveys & tutorials,* vol. 18, no. 1, pp. 236-262, 2015.

[18] M. Compastié, R. Badonnel, O. Festor, R. He, and M. Kassi-Lahlou, "A software-defined security strategy for supporting autonomic security enforcement in distributed cloud," in *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2016: IEEE, pp. 464-467.

[19] M. Al-Ayyoub, Y. Jararweh, E. Benkhelifa, M. Vouk, and A. Rindos, "Sdsecurity: A software defined security experimental framework," in *2015 IEEE international conference on communication workshop (ICCW)*, 2015: IEEE, pp. 1871-1876.

[20] M. Vizardl, "What software-defined security could mean for the channel," *ed,* 2013.

[21] K. Walker, "Cloud security alliance announces software defined perimeter (sdp) initiative," *online] https://cloudsecurityalliance. org/media/news/csa-*

announcessoftware-defined-perimeter-sdp-initiative/(accessed October 2014), 2013.

[22] (2014). *Catbird® 6.0: Private Cloud Security*.

[23] N. v, "vArmour distributed security system: protecting assets in the world without perimeters," 2015.

[24] V. Team, "VMware vCloud networking and security overview, White Paper, VMware," *ed: Inc,* 2013.

[25] N. Team, "NetCitadel's one control platform the key to intelligent, adaptive network security, White Paper, NetCitadel," *ed: Inc,* 2012.

[26] C. R. Kothari, *Research methodology: Methods and techniques*. New Age International, 2004.

[27] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and openflow: From concept to implementation," *IEEE Communications Surveys & Tutorials,* vol. 16, no. 4, pp. 2181-2206, 2014.

[28] A. Markelov, *Certified OpenStack Administrator Study Guide*. Springer, 2016.

[29] J. Son, T. He, and R. Buyya, "CloudSimSDN-NFV: Modeling and simulation of network function virtualization and service function chaining in edge computing environments," *Software: Practice and Experience,* 2019.

[30] D. C. Marinescu, *Cloud computing: theory and practice*. Morgan Kaufmann, 2017.

[31] J. Sahoo, S. Mohapatra, and R. Lath, "Virtualization: A survey on concepts, taxonomy and associated security issues," in *2010 Second International Conference on Computer and Network Technology*, 2010: IEEE, pp. 222-226.

[32] Y. Xing and Y. Zhan, "Virtualization and cloud computing," in *Future Wireless Networks and Information Systems*: Springer, 2012, pp. 305-312.

[33] C.-J. Chung, *SDN-based Proactive Defense Mechanism in a Cloud System*. Arizona State University, 2015.

[34] C. C. A. SM–CSA, "Security guidance for critical areas of focus in cloud computing V3. 0," ed, 2011.

[35] M. Compastié, R. Badonnel, O. Festor, and R. He, "From Virtualization Security Issues to Cloud Protection Opportunities: An In-Depth Analysis of System Virtualization Models," *Computers & Security,* p. 101905, 2020.

[36] N. M. Almutairy, K. H. Al-Shqeerat, and H. A. Al Hamad, "A taxonomy of virtualization security issues in cloud computing environments," *Indian Journal of Science and Technology,* vol. 12, no. 3, 2019.

[37] D. B. Hoang and S. Farahmandian, "Security of Software-Defined Infrastructures with SDN, NFV, and Cloud Computing Technologies," in *Guide to Security in SDN and NFV*: Springer, 2017, pp. 3-32.

[38] K. Jeffery and L. Schubert, "Advances in Clouds Research in Future Cloud Computing," *London,* pp. 34-45, 2016.

[39] P. Mell and T. Grance, "The NIST Definition of Cloud Computing, National Institute of Standards and Technology, Gaithersburg, MD, 2011," *URL http://nvlpubs. nist. gov/nistpubs/Legacy/SP/nistspecialpublication800-145. pdf,* vol. 905, 2014.

[40] S. Bele, "A Comprehensive Study on Cloud Computing," *International Journal of Information Research and Review,* vol. 5, pp. 5310-5313, 2018.

[41] C. N. Modi and K. Acha, "Virtualization layer security challenges and intrusion detection/prevention systems in cloud computing: a comprehensive review," *the Journal of Supercomputing,* vol. 73, no. 3, pp. 1192-1234, 2017.

[42] H. Tabrizchi and M. K. Rafsanjani, "A survey on security challenges in cloud computing: issues, threats, and solutions," *The Journal of Supercomputing,* pp. 1-40, 2020.

[43] R. Kumar and R. Goyal, "On cloud security requirements, threats, vulnerabilities and countermeasures: A survey," *Computer Science Review,* vol. 33, pp. 1-48, 2019.

[44] A. Singh and K. Chatterjee, "Cloud security issues and challenges: A survey," *Journal of Network and Computer Applications,* vol. 79, pp. 88-115, 2017.

[45] M. Hawedi, C. Talhi, and H. Boucheneb, "Security as a service for public cloud tenants (SaaS)," *Procedia computer science,* vol. 130, pp. 1025-1030, 2018.

[46] S. Iqbal *et al.*, "On cloud security attacks: A taxonomy and intrusion detection and prevention as a service," *Journal of Network and Computer Applications,* vol. 74, pp. 98-120, 2016.

[47] S. Prakash, "Role of virtualization techniques in cloud computing environment," in *Advances in Computer Communication and Computational Sciences*: Springer, 2019, pp. 439-450.

[48] M. Saraswathi and T. Bhuvaneswari, "Multitenancy in cloud software as a service application," *International Journal of Advanced Research in Computer Science and Software Engineering,* vol. 3, no. 11, pp. 159-162, 2013.

[49] T. T. W. Group, "The treacherous 12: cloud computing top threats in 2016," *Cloud Security Alliance,* 2016.

[50] K. Greene, "TR10: Software-defined networking," *Technology Review (MIT),* 2009.

[51] M. Pham and D. B. Hoang, "SDN applications-The intent-based Northbound Interface realisation for extended applications," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, 2016: IEEE, pp. 372-377.

[52] P. Berde *et al.*, "ONOS: towards an open, distributed SDN OS," in *Proceedings of the third workshop on Hot topics in software defined networking*, 2014, pp. 1-6.

[53] D. B. Hoang, "Software Defined Networking? Shaping up for the next disruptive step?," *Australian Journal of Telecommunications and the Digital Economy,* vol. 3, no. 4, 2015.

[54] P. Goransson, C. Black, and T. Culver, *Software defined networks: a comprehensive approach*. Morgan Kaufmann, 2016.

[55] K. Benzekki, A. El Fergougui, and A. Elbelrhiti Elalaoui, "Software-defined networking (SDN): a survey," *Security and communication networks,* vol. 9, no. 18, pp. 5803-5833, 2016.

[56] P. Floodlight. Floodlight [Online] Available: https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/

[57] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a model-driven sdn controller architecture," in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, 2014: IEEE, pp. 1-6.

[58] M. Fernandez, "Evaluating OpenFlow controller paradigms," in *ICN 2013, The Twelfth International Conference on Networks*, 2013, pp. 151-157.

[59] K. Kaur, S. Kaur, and V. Gupta, "Performance analysis of python based openflow controllers," 2016.

[60] Z. Latif, K. Sharif, F. Li, M. M. Karim, S. Biswas, and Y. Wang, "A comprehensive survey of interface protocols for software defined networks," *Journal of Network and Computer Applications,* vol. 156, p. 102563, 2020.

[61] N. McKeown *et al.*, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review,* vol. 38, no. 2, pp. 69-74, 2008.

[62] A. Doria *et al.*, "Forwarding and Control Element Separation (ForCES) Protocol Specification," *RFC,* vol. 5810, pp. 1-124, 2010.

[63] B. Pfaff and B. Davie, "The open vswitch database management protocol," *Internet Requests for Comments, RFC Editor, RFC,* vol. 7047, 2013.

[64] H. Song, "Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013, pp. 127-132.

[65] M. Smith, M. Dvorkin, Y. Laribi, V. Pandey, P. Garg, and N. Weidenbache, "OpFlex Control Protocol, Internet Draft, Internet Engineering Task Force," ed: April, 2014.

[66] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, "Openstate: Programming platform-independent stateful openflow applications inside the switch," *ACM SIGCOMM Computer Communication Review,* vol. 44, no. 2, pp. 44-51, 2014.

[67] N. V. R. Gupta and M. Ramakrishna, "A road map for SDN-open flow networks," *International Journal of Modern Communication Technologies and Research,* vol. 3, no. 6, p. 265725, 2015.

[68] A. Gupta, H. Bhadauria, A. Singh, and J. C. Patni, "A theoretical comparison of job scheduling algorithms in cloud computing environment," in *2015 1st International Conference on Next Generation Computing Technologies (NGCT)*, 2015: IEEE, pp. 16-20.

[69] S. Rowshanrad, S. Namvarasl, V. Abdi, M. Hajizadeh, and M. Keshtgary, "A survey on SDN, the future of networking," *Journal of Advanced Computer Science & Technology,* vol. 3, no. 2, pp. 232-248, 2014.

[70] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Communications Surveys & Tutorials,* vol. 17, no. 1, pp. 27-51, 2014.

[71] C. Ching-Hao and Y.-D. Lin, "OpenFlow Version Roadmap," tech. rep, 2015. http://speed. cis. nctu. edu. tw/~ ydlin/miscpub …, 2015.

[72] P. T. Congdon, P. Mohapatra, M. Farrens, and V. Akella, "Simultaneously reducing latency and power consumption in openflow switches," *IEEE/ACM Transactions On Networking,* vol. 22, no. 3, pp. 1007-1020, 2013.

[73] A. Khan and N. Dave, "Enabling hardware exploration in software-defined networking: A flexible, portable openflow switch," in *2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines*, 2013: IEEE, pp. 145-148.

[74] W. Braun and M. Menth, "Software-defined networking using OpenFlow: Protocols, applications and architectural design choices," *Future Internet,* vol. 6, no. 2, pp. 302-336, 2014.

[75] O. S. S. V. ONF, "1.5. 1, Open Networking Foundation, 2015," ed.

[76] B. Salisbury, "The northbound api-a big little problem," ed: June, 2012.

[77] R. Chua, "OpenFlow northbound API: A new olympic sport," ed: July, 2012.

[78] T. Koponen *et al.*, "Onix: A distributed control platform for large-scale production networks," in *OSDI*, 2010, vol. 10, pp. 1-6.

[79] Z. Wang, T. Tsou, J. Huang, X. Shi, and X. Yin, "Analysis of comparisons between OpenFlow and ForCES," *ForCES, IETF,* 2012.

[80] F. A. Botelho, F. M. V. Ramos, D. Kreutz, and A. N. Bessani, "On the feasibility of a consistent and fault-tolerant data store for SDNs," in *2013 Second european workshop on software defined networks*, 2013: IEEE, pp. 38-43.

[81] K. Govindarajan, K. C. Meng, and H. Ong, "A literature review on software-defined networking (SDN) research topics, challenges and solutions," in *2013 Fifth International Conference on Advanced Computing (ICoAC)*, 2013: IEEE, pp. 293-299.

[82] S. Scott-Hayward and T. Arumugam, "OFMTL-SEC: State-based Security for Software Defined Networks," in *2018 IEEE Conference on Network Function*

Virtualization and Software Defined Networks (NFV-SDN), 2018: IEEE, pp. 1-7.

[83] S. Hong, L. Xu, H. Wang, and G. Gu, "Poisoning network visibility in software-defined networks: New attacks and countermeasures," in *NDSS*, 2015, vol. 15, pp. 8-11.

[84] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A survey of security in software defined networks," *IEEE Communications Surveys & Tutorials,* vol. 18, no. 1, pp. 623-654, 2015.

[85] N. ETSI, "Network functions virtualisation (NFV); terminology for main concepts in NFV," *Group Specification,* vol. 3, pp. 1-10, 2014.

[86] G. ETSI, "011: Network functions virtualisation (NFV) release 2; management and orchestration," *Os-Ma-Nfvo reference point-interface and information model specification. v3,* vol. 1, 2018.

[87] R. Mijumbi, J. Serrat, J.-L. Gorricho, S. Latré, M. Charalambides, and D. Lopez, "Management and orchestration challenges in network functions virtualization," *IEEE Communications Magazine,* vol. 54, no. 1, pp. 98-105, 2016.

[88] O. D. C. Alliance, "Master Usage Model: Software-Defined Networking," ed: Rev, 2013.

[89] J. M. AMA. The 2016 guide to SDN and NFV – part 4: Network Functions

Virtualization (NFV) a status update

[90] A. Milenkoski *et al.*, "Security position paper network function virtualization," *Cloud Security Alliance-Virtualization Working Group,* 2016.

[91] N. ETSI, "Network Functions Virtualisation (NFV); NFV Security; Problem Statement," *ETSI GS NFV-SEC,* vol. 1, 2014.

[92] S. Lal, T. Taleb, and A. Dutta, "NFV: Security threats and best practices," *IEEE Communications Magazine,* vol. 55, no. 8, pp. 211-217, 2017.

[93] M. Sloman and E. Lupu, "Security and management policy specification," *IEEE network,* vol. 16, no. 2, pp. 10-19, 2002.

[94] A. Hassan and W. Bahgat, "A framework for translating a high level security policy into low level security mechanisms," *Journal of Electrical Engineering,* vol. 61, no. 1, pp. 20-28, 2010.

[95] D. Kosiur, *Understanding policy-based networking*. John Wiley & Sons, 2001.

[96] F. Cuppens, N. Cuppens-Boulahia, T. Sans, and A. Miège, "A formal approach to specify and deploy a network security policy," in *IFIP World Computer Congress, TC 1*, 2004: Springer, pp. 203-218.

[97] K. Karmakar, "Techniques for securing software defined networks and survices," *Ph. D. dissertation,* 2019.

[98] J. B. Bernabé *et al.*, "Security policy specification," in *Network and Traffic Engineering in Emerging Distributed Computing Applications*: IGI Global, 2013, pp. 66-93.

[99] J. Liu *et al.*, "Leveraging software-defined networking for security policy enforcement," *Information Sciences,* vol. 327, pp. 288-299, 2016.

[100] S. Engram and J. Ligatti, "Through the lens of code granularity: A unified approach to security policy enforcement," in *2020 IEEE Conference on Application, Information and Network Security (AINS)*, 2020: IEEE, pp. 41-46.

[101] S. Cabuk *et al.*, "Towards automated security policy enforcement in multi-tenant virtual data centers," *Journal of Computer Security,* vol. 18, no. 1, pp. 89-121, 2010.

[102] M. B. Baig, C. Fitzsimons, S. Balasubramanian, R. Sion, and D. E. Porter, "CloudFlow: Cloud-wide policy enforcement using fast VM introspection," in *2014 IEEE International Conference on Cloud Engineering*, 2014: IEEE, pp. 159-164.

[103] A. Tabiban, S. Majumdar, L. Wang, and M. Debbabi, "Permon: An openstack middleware for runtime security policy enforcement in clouds," in *2018 IEEE Conference on Communications and Network Security (CNS)*, 2018: IEEE, pp. 1-7.

[104] K. K. Karmakar, V. Varadharajan, U. Tupakula, and M. Hitchens, "Policy based security architecture for software defined networks," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, 2016: ACM, pp. 658-663.

[105] R. Fernando, R. Ranchal, B. Bhargava, and P. Angin, "A monitoring approach for policy enforcement in cloud services," in *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, 2017: IEEE, pp. 600-607.

[106] X. Wang, W. Shi, Y. Xiang, and J. Li, "Efficient network security policy enforcement with policy space analysis," *IEEE/ACM Transactions on Networking,* vol. 24, no. 5, pp. 2926-2938, 2015.

[107] V. Varadharajan, K. Karmakar, U. Tupakula, and M. Hitchens, "A policy-based security architecture for software-defined networks," *IEEE Transactions on Information Forensics and Security,* vol. 14, no. 4, pp. 897-912, 2018.

[108] C. Basile, F. Valenza, A. Lioy, D. R. Lopez, and A. P. Perales, "Adding Support for Automatic Enforcement of Security Policies in NFV Networks," *IEEE/ACM Transactions on Networking,* vol. 27, no. 2, pp. 707-720, 2019.

[109] F. Li *et al.*, "Cyberspace-Oriented Access Control: A Cyberspace Characteristics-Based Model and its Policies," *IEEE Internet of Things Journal,* vol. 6, no. 2, pp. 1471-1483, 2018.

[110] F. Roesner, T. Kohno, A. Moshchuk, B. Parno, H. J. Wang, and C. Cowan, "User-driven access control: Rethinking permission granting in modern operating systems," in *2012 IEEE Symposium on Security and Privacy*, 2012: IEEE, pp. 224-238.

[111] R. S. Sandhu and P. Samarati, "Access control: principle and practice," *IEEE communications magazine,* vol. 32, no. 9, pp. 40-48, 1994.

[112] P. Samarati and S. C. de Vimercati, "Access control: Policies, models, and mechanisms," in *International School on Foundations of Security Analysis and Design*, 2000: Springer, pp. 137-196.

[113] F. Sifou, A. Hammouch, and A. Kartit, "Ensuring security in cloud computing using access control: A survey," in *Proceedings of the Mediterranean Symposium on Smart City Applications*, 2017: Springer, pp. 255-264.

[114] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *2010 Proceedings IEEE INFOCOM*, 2010: Ieee, pp. 1-9.

[115] A. Almutairi, M. Sarfraz, S. Basalamah, W. Aref, and A. Ghafoor, "A distributed access control architecture for cloud computing," *IEEE software,* vol. 29, no. 2, pp. 36-44, 2011.

[116] Y. A. Younis, K. Kifayat, and M. Merabti, "An access control model for cloud computing," *Journal of Information Security and Applications,* vol. 19, no. 1, pp. 45-60, 2014.

[117] F. Cai, N. Zhu, J. He, P. Mu, W. Li, and Y. Yu, "Survey of access control models and technologies for cloud computing," *Cluster Computing,* vol. 22, no. 3, pp. 6111-6122, 2019.

[118] M. L. Damiani, E. Bertino, B. Catania, and P. Perlasca, "GEO-RBAC: a spatially aware RBAC," *ACM Transactions on Information and System Security (TISSEC),* vol. 10, no. 1, p. 2, 2007.

[119] G.-Y. Lin, S. He, H. Huang, J.-Y. Wu, and W. Chen, "Access control security model based on behavior in cloud computing environment," *Journal of China Institute of Communications,* vol. 33, no. 3, pp. 59-66, 2012.

[120] R. PV and R. Sandhu, "POSTER: security enhanced administrative role based access control models," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016: ACM, pp. 1802-1804.

[121] I. Tarkhanov, "Extension of access control policy in secure role-based workflow model," in *2016 IEEE 10th International Conference on Application of Information and Communication Technologies (AICT)*, 2016: IEEE, pp. 1-4.

[122] A. Ranjbar, M. Antikainen, and T. Aura, "Domain isolation in a multi-tenant software-defined network," in *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*, 2015: IEEE, pp. 16-25.

[123] Y. Jararweh, M. Al-Ayyoub, E. Benkhelifa, M. Vouk, and A. Rindos, "Software defined cloud: Survey, system and evaluation," *Future Generation Computer Systems,* vol. 58, pp. 56-74, 2016.

[124] A. Viswanathan and B. Neuman, "A survey of isolation techniques," *Information Sciences Institute, University of Southern California,* 2009.

[125] I. Mavridis and H. Karatza, "Combining containers and virtual machines to enhance isolation and extend functionality on cloud computing," *Future Generation Computer Systems,* vol. 94, pp. 674-696, 2019.

[126] Y. Mundada, A. Ramachandran, and N. Feamster, "SilverLine: Data and Network Isolation for Cloud Services," in *HotCloud*, 2011.

[127] M. Factor *et al.*, "Secure logical isolation for multi-tenancy in cloud storage," in *2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST)*, 2013: IEEE, pp. 1-5.

[128] M. Pfeiffer, M. Rossberg, S. Buttgereit, and G. Schaefer, "Strong Tenant Separation in Cloud Computing Platforms," in *Proceedings of the 14th International Conference on Availability, Reliability and Security*, 2019, pp. 1-10.

[129] V. Del Piccolo, A. Amamou, K. Haddadou, and G. Pujolle, "A survey of network isolation solutions for multi-tenant data centers," *IEEE Communications Surveys & Tutorials,* vol. 18, no. 4, pp. 2787-2821, 2016.

[130] C. Chen, D. Li, J. Li, and K. Zhu, "SVDC: A Highly Scalable Isolation Architecture for Virtualized Layer-2 Data Center Networks," *IEEE Transactions on Cloud Computing,* vol. 6, no. 4, pp. 1178-1190, 2016.

[131] O. Accessed, "Open Networking Foundation," ed, 2016.

[132] N. Etsi, "Etsi gs nfv 002 v1. 1.1 network functions virtualization (nfv)," *Architectural Framework. sl: ETSI,* 2013.

[133] SDxCentral, "SDN security challenges in SDN environments," 2017. [Online]. Available: https://www.sdxcentral.com/security/definitions/security-challenges-sdn-software-definednetworks/.

[134] S. Farahmandian and D. B. Hoang, "SDS 2: A novel software-defined security service for protecting cloud computing infrastructure," in *2017 IEEE 16th International Symposium on Network Computing and Applications (NCA)*, 2017: IEEE, pp. 1-8.

[135] G. N. Stone, B. Lundy, and G. G. Xie, "Network policy languages: a survey and a new approach," *IEEE network,* vol. 15, no. 1, pp. 10-21, 2001.

[136] X. Yin, X. Chen, L. Chen, G. Shao, H. Li, and S. Tao, "Research of Security as a Service for VMs in IaaS Platform," *IEEE Access,* vol. 6, pp. 29158-29172, 2018.

[137] M. Zhou, R. Zhang, D. Zeng, and W. Qian, "Services in the cloud computing era: A survey," in *2010 4th International Universal Communication Symposium*, 2010: IEEE, pp. 40-46.

[138] A. Manzalini and N. Crespi, "SDN and NFV for network cloud computing: a universal operating system for SD infrastructures," in *2015 IEEE Fourth Symposium on Network Cloud Computing and Applications (NCCA)*, 2015: IEEE, pp. 1-6.