On the Security Assessment of the Cloud

Salman Manzoor

School of Computing and Communications
Lancaster University

This dissertation is submitted for the degree of
Doctor of Philosophy

September 2022

Declaration

I hereby declare that the contents of this dissertation are original and have not been submitted for consideration for any other degree or qualification at any other university. I also certify that the presented work is my own, except where specific reference is made to the work of others.

Salman Manzoor
September 2022

Acknowledgements

First of all, I would like to thank my advisor Neeraj Suri for giving me the opportunity to work toward a PhD. Thank you for always being open to new ideas and allowing me to develop and pursue my research interests. I am grateful for your valuable suggestions and guidance regarding my professional and personal life.

I am also very grateful to Stefan Katzenbeisser and Muhammad Khalid for accepting to be my external reviewers and to Yang Lu for being my internal reviewer. I would also like to thank Antonios Gouglidis for chairing the PhD defense.

I want to thank all my colleagues who shared the PhD journey with me both at Lancaster university and TU Darmstadt. Particularly, I would like to thank Tsveti for always helping me whenever I struggled to understand German letters by translating them. I enjoyed our conversations about Bulgaria, Pakistan, and German cultures. I would also like to acknowledge Olli and Habib, who introduced me to the magic of black coffee. I can undoubtedly say that coffee breaks were fun. I would also like to thank Jesus, Ruben, Ahmed, Nico, Habib, Hatem, Kubilay, Heng, and Yiqun for the stimulating research discussions, group BBQs and the occasional breaks at Herngarten. Thank you, Sabine, for your help with all the necessary paperwork, and Ute, for sharing your cake recipes with me.

Finally, I am grateful for the support, love, and encouragement I received from my family. Without their support, I would not have been able to complete this journey. Thank you for always being with me during stressful and less stressful times.

Abstract

Cloud computing is an enabling technology paradigm that provides access to the geo-distributed pool of resources that are rapidly and flexibly provisioned at run-time with minimum management from the user. These benefits have driven the proliferation of the Cloud over the last decade. Many organizations have migrated to the Cloud or have a Cloud-first strategy for their businesses. Despite these benefits, the security of the Cloud has been flagged as among the top concerns by its users.

To address security concerns, Threat Analysis (TA) is often advocated to ascertain a system's exposure to threats. A plethora of TA techniques exist that focus on analyzing threats to targeted assets at the system's level (e.g., components, hardware) or at the user's level (e.g., virtual machine) in the Cloud. These techniques are effective, but their applicability is limited beyond their targeted asset. However, the Cloud is a complex system entailing both the physical and virtual resources. Moreover, these resources can instantiate, migrate across physical hosts, or decommission to provide rapid resource elasticity to the users.

On this background, this thesis aims at assessing the security of the Cloud holistically by considering the interactions among the services/components involved in the operational stack of the Cloud. In this regard, a technology-agnostic information flow model is developed that represents the Cloud's functionality through a set of conditional transitions. Furthermore, threats are added to the model to analyze their impact on the Cloud. This enables the exploration of a threat's behavior and its propagation across the Cloud and supports assessing the security of the Cloud by analyzing the impact of multiple threats across various operational layers/assets. Using public information on threats from the National Vulnerability Database (NVD), actual Cloud attacks were traced and speculatively postulated alternate potential attack paths. Furthermore, the thesis also investigates different threats with similar indicators of compromise (e.g., attack patterns) to be considered in the security assessment along with the specific user's requirements. Finally, the thesis also targets the evaluation of potential violations from the Cloud providers that breach users' requirements.

The results presented in the thesis demonstrate that by ascertaining the attack paths and considering the interplay between threats and security requirements, the security of the Cloud can be comprehensively assessed.

Publications

The following publications have, in parts verbatim, been included in this thesis.

- **S. Manzoor**, A. Gouglidis, M. Bradbury and N. Suri, "ThreatPro: Multi-Layer Threat Analysis in the Cloud", ACM Transactions on Privacy and Security (TOPS) (Under Review).
- **S. Manzoor**, D. Prince and N. Suri, "Ontologies for Vulnerability Terrain Mapping and Attack Reasoning", In Proceedings of the International Conference on Network and System Security (NSS) (Under Review).
- **S. Manzoor**, H. Zhang and N. Suri, "Threat Modeling and Analysis for the Cloud Ecosystem", In Proceedings of the IEEE International Conference on Cloud Engineering (IC2E), IC2E'18, Orlando, USA, April 17-20, 2018, pp. 278-281, doi: 10.1109/IC2E.2018.56.
- **S. Manzoor**, T. Vateva-Gurova, R. Trapero, N. Suri, "Threat Modeling the Cloud: An Ontology based Approach", In: Proc. of the International Workshop on Information and Operational Technology Security Systems (IOSec) IOSec'18, Crete, Greece, September 13, pp. 61-72, doi: 10.1007/978-3-030-12085-6_6.
- **S. Manzoor**, Jesus Luna and Neeraj Suri "AttackDive: Diving Deep into the Cloud Ecosystem to Explore Attack Surfaces", In: Proc. of IEEE Services Computing (SCC), SCC'17, Honolulu, USA, June 25-30 2017, pp. 499-502, doi: 10.1109/SCC.2017.74.
- **S. Manzoor**, A. Taha and N. Suri "Trust Validation of Cloud IaaS: A Customer-centric Approach", In: Proc. of IEEE IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), TrustCom'16, Tianjin, China, Aug 23-26 2016, pp. 97-104, doi: 10.1109/TrustCom.2016.0051.

The following publications are related to different aspects covered in this thesis and have been published during my doctoral studies, but have not been included in the thesis.

- T. Vateva-Gurova, **S. Manzoor**, Y. Huang, and N. Suri. “InfoLeak:Scheduling-Based Information Leakage”. In: Proc. of the 23rd IEEE Pacific Rim International Symposium on Dependable Computing (PRDC) PRDC’18, Taipei, Taiwan, December 4-7, 2018. 2018, pp. 44–53, doi: 10.1109/PRDC.2018.00015.

- T. Vateva-Gurova, **S. Manzoor**, R. Trapero, N. Suri, "Protecting Cloud-Based CIs: Covert Channel Vulnerabilities at the Resource Level", In: Proc. of the International Workshop on Information and Operational Technology Security Systems (IOSec) IOSec’18, Crete, Greece, September 13, pp. 27-38, doi: 10.1007/978-3-030-12085-6_3.

- H Zhang, **S. Manzoor** and N Suri, "Monitoring path discovery for supporting indirect monitoring of cloud services", In: Proc. of the IEEE International Conference on Cloud Engineering (IC2E), IC2E’18, Orlando, USA, pp. 274-277, April 17-20, 2018, doi: 10.1109/IC2E.2018.00055.

- A. Taha, **S. Manzoor** and N. Suri, "SLA-based Service Selection for Multi-Cloud Environments", In: Proc. of IEEE EDGE Computing (EGDE), EDGE’2017, Honolulu, USA, June 25-30, 2017, pp.65-72, doi: 10.1109/IEEE.EDGE.2017.17.

Table of contents

List of Figures

List of Tables

Nomenclature

Acronyms / Abbreviations

AWS	Amazon Web Services
CIA	Confidentiality Integrity Availability
CSC	Cloud Service Customer
CSP	Cloud Service Provider
DSM	Design Structure Matrix
HLPN	High Level Petri Net
IaaS	Infrastructure as a Service
IT	Information Technology
NIST	National Institute of Standards and Technology
NVD	National Vulnerability Database
PaaS	Platform as a Service
SaaS	Software as a Service
SLA	Service Level Agreement
TA	Threat Analysis
VMM	Virtual Machine Monitor
VM	Virtual Machine

Chapter 1

Introduction

Traditional Information Technology (IT) setup consists of installing various parts of the physical hardware (e.g., servers, computers) on the premises of an organization to provide access to the organization's applications and data. Additionally, scaling up an application requires organizations to plan the hardware requirements to support the scaling operation and procure the additional hardware. Conversely, scaling down an application/service means that organizations must decommission the acquired hardware. Thus, both the scaling up and down operations are cumbersome, time-consuming, and costly for many organizations. In contrast to this traditional IT setup, Cloud computing is a technology paradigm that provides on-demand access to the geo-distributed pool of resources (e.g., services, computation, storage, etc.) as utilities over the Internet. These resources can be rapidly and dynamically provisioned according to the requirements of an application without direct management from the users[1].

Moreover, the Cloud utilizes a *pay-as-you-go model* that helps reduce the users' capital and operational expenses. Thus, organizations can rapidly scale up/down their applications in the Cloud without needing to buy/decommission the hardware. Therefore, the Cloud proliferation has increased across a wide range of businesses due to the economic benefits it offers. For example, Amazon, Inc. provides a spectrum of Cloud services (e.g., computing, data storage, networking, etc.) to users, including individual users, companies, or governments, through its proprietary Cloud platform known as Amazon Web Services (AWS) [2]. Similarly, Google delivers Cloud services, e.g., building and deploying artificial intelligence and machine learning models on its Cloud [3], and Microsoft offers more than 200 services in their Cloud suite Azure [4].

Consequently, Amazon, Microsoft, and Google collectively accumulated more than 60 percent of the global market share [5] in 2022, with an increase of 34 percent in Cloud adoption compared to the year 2021. In addition to these market leaders in the Cloud,

numerous small and medium-size Cloud Service Providers (CSPs) offer their services with distinctive features. For example, Dropbox offers particular file hosting and sharing features that make file sharing simpler and easier among its users [6]. In contrast, Github [7] offers specific services for software development. These services include code repositories, version control, and code review to facilitate software development. TechCrunch has reported that the Cloud infrastructure market share has already hit $129B in 2020 [8]. According to Gartner, Inc., the Cloud services industry is projected to increase exponentially over the following years. The growth is estimated to be approximately three times that of the entire IT services [9]. It is evident that Cloud services have become prevalent in a wide range of applications and consequently have penetrated into business organizations and personal lives [10].

1.1 A Brief Overview of the Cloud Environment

This section presents a brief overview of the Cloud environment. Cloud computing is defined as a model to enable convenient, on-demand access to a shared pool of resources (e.g., computing, services, networks, applications, and storage) over the Internet as utilities, according to a definition by the National Institute of Science and Technology (NIST) [1]. Thus, the essential characteristics of the Cloud are flexible billing, resource pooling, on-demand access, and rapid elasticity of resources to the users. The Cloud Service Providers (CSPs) have adopted different service delivery models to provide their services to the users. These are; Software as a Service (SaaS) [11], Platform as a Service (PaaS) [12], and Infrastructure as a Service (IaaS) [13]. These service models are briefly explained below.

- **SaaS:** In SaaS, a CSP offers software applications as a service and allows users to control only the application configurations. Online word processors such as Google docs and Microsoft Office 365 are examples of SaaS that enable users to access their documents across a range of devices seamlessly.
- **PaaS:** PaaS providers host different computing development platforms and offer them as a service to developers. These provisions enable application developers to instantiate and manage the platform without the complexity of building and maintaining the underlying infrastructure. Online games development engines or software development platforms fall under the category of PaaS offerings.
- **IaaS:** IaaS provides computing and storage resources where a user controls and manages its resources. A prominent example of IaaS is the Amazon EC2 which allows users to rent computing resources as a Virtual Machine (VM). The user controls both

the configuration of the VMs and the applications running on these VMs. Moreover, IaaS provides the flexibility to connect multiple VMs to create a coherent view of the resources in the Cloud specific to a user.

Each service delivery model offers specific services to the users with varying degrees of control over the services. As seen from Figure 1.1, IaaS provides the highest control to the user while SaaS offers the least management control.

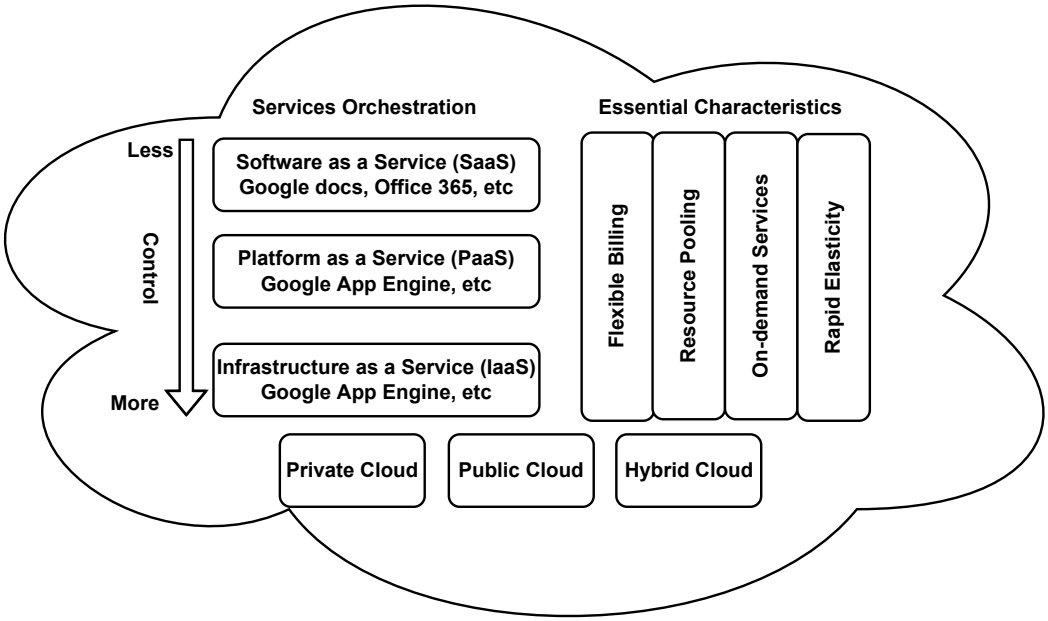

Fig. 1.1 An overview of the Cloud.

The Cloud can be deployed in various settings according to user requirements. The standard deployment models for the Cloud are private, public, and hybrid. A private Cloud deployment entails that the Cloud infrastructure is restricted to an organization that becomes responsible for managing the Cloud. On the other hand, public Cloud deployments are open to the public, i.e., the underlying physical hardware is shared among different users, and therefore, resources belonging to various users/organizations can be hosted on the same physical server. Architecturally, there are very few differences between private and public Cloud deployments. However, the security challenges in a public Cloud are more significant than in a private Cloud [14]. For example, side-channel attacks [15] are more significant in public Cloud deployments. NIST has published guidelines on dealing with security and privacy issues in the Cloud [16]. Hybrid Cloud deployment combines the benefits of both

public and private Cloud deployments. It is particularly beneficial for organizations with an on-premise Cloud deployment to store sensitive data but can use public deployments for business intelligence, for instance.

Thus, the rapid flexibility in resource management within the Cloud and the availability of various Cloud deployment configurations are the driving forces behind the rapid increase in the Cloud usage. It is evident that many organizations have either migrated to the Cloud or adopted the Cloud-first strategy for their businesses [17].

1.2 Security of the Cloud

The economic benefits of the Cloud have driven many organizations to either migrate their operations to the Cloud or adopt a Cloud-first strategy [17]. On the other hand, the security of the Cloud has been raised as one of the main concerns of the users in a recent survey conducted by Check Point [18]. The survey found that 75% of the users (including organizations) are either extremely concerned or concerned about the security of the Cloud. Broadly, these security concerns relate to the classical Confidentiality, Integrity, Availability (CIA) triad. A brief overview of these properties is presented below with respect to the Cloud.

- **Confidentiality**: The Cloud prevents unauthorized disclosure of the resources/data. The Cloud data should not be leaked either by accident or on purpose to unauthorized recipients, including the Cloud administrators.
- **Integrity**: The Cloud configurations and data cannot be altered improperly, and access to the Cloud is restricted to the authorized entities. This also implies that any incorrect changes to the Cloud are detected. For instance, the software is only updated after its signature has been verified to detect any changes to the code.
- **Availability**: The Cloud is always available to provide services to the users. These services could be the user's resources (e.g., VM) or the Cloud offerings such as applications offered by the Cloud Service Providers (CSPs).

It has been widely known that a violation of any of these properties poses a security threat to an organization. Specifically, 68% of organizations in the survey regard the misconfigurations in the Cloud as a key contributor to the threats facing the Cloud. A misconfigured Cloud could potentially leak information to other users on the same physical machine leading to a violation of the confidentiality property [19]. Furthermore, 58% organizations raised unauthorized access or modification (a violation of integrity) as their primary concern [18]. Additionally, there have been attacks on the Cloud that target the availability of a service.

For example, Amazon reported that they had thwarted one of the biggest Distributed Denial of Service (DDoS) attacks [20]. Similarly, GitHub suffered a DDoS attack, and their services became offline as a result of the attack [21]. On the other hand, Microsoft showed that it is possible to implement malware in a virtual machine [22]. The authors showed that it is possible to run malicious software that gives attackers access to the lower layer of the system, yet the malware remains undetected by the system.

In lieu of the information above, it is evident that a plethora of attack surfaces exists across the abstraction of the Cloud that an attacker can exploit to undermine the security of the Cloud. An attack surface is the sum of different system entry points that can be used to gain access to the system illegitimately [23]. Broadly, the attack surfaces for the Cloud environment are presented in a taxonomy in [24] and their taxonomy is aligned with the proposed Cloud architecture published by NIST [25]. These are the: (i) service layer to the user, (ii) user to the service layer, (iii) service layer to the Cloud management layer, (iv) Cloud management layer to the service layer, (v) Cloud management layer to the user and (vi) user to the Cloud management layer. As Figure 1.2 shows, the users invoke services that subsequently use the Cloud to render their functionality to the users. These services could belong to either SaaS, PaaS, or IaaS. Furthermore, the users manage these services according to the control given by the Cloud providers. Thus, the corresponding attack surfaces belonging to these interfaces are shown in Figure 1.2 and explained in the following.

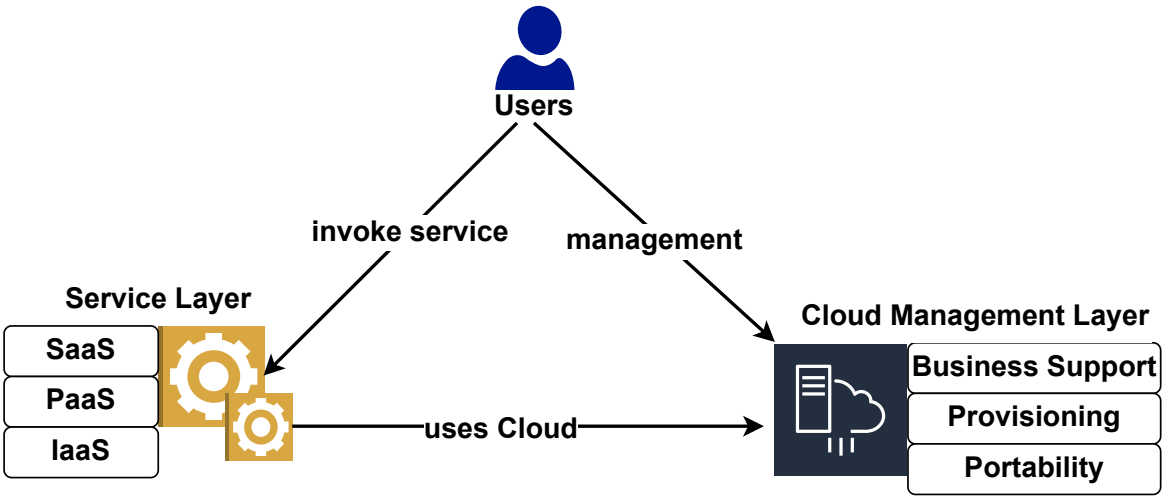

Fig. 1.2 Cloud users and possible attack surfaces

The first attack surface, i.e., a service layer/instance to the user, is the same as a server-to-client interface. Thus, it enables all types of attacks conventionally possible in a client-server architecture. The typical examples of attacks using this attack surface include buffer overflow and SQL injection attacks. Conversely, the second attack surface enables attacks from a

user to the Cloud in the same way a user can target a server in the traditional client-server setup. Prominent classes of attacks include browser-based attacks, such as SSL certificate spoofing [26]. The third attack surface demonstrates the use of service to attack the Cloud. A prominent example of such an attack surface is the use of VM to spread malicious software in the Cloud [22]. On the other hand, the fourth attack surface describes the possibility of using the Cloud management layer as a potential attack surface. An example of an attack might include a malicious Cloud provider that starts decreasing the availability of a service by relieving the assigned resources attached to it. The last two attack surfaces demonstrate the possibility of attacking the Cloud management layer from the user and vice-versa. The Cloud provides management access to the users to control their VMs, and they can use the management interface as a potential attack surface. The set of attacks launched through this attack surface is the same as attacks based on the user to the service attack surface. The final attack surface presents the possibility of using the Cloud management layer to victimize a user. A typical example of using such an attack surface is increasing the bill of the user or a phishing attempt to violate a security requirement of a user.

Analyzing the Cloud Security

A trend of utilizing an attack surface to violate the Confidentiality, Integrity, and Availability (CIA) property of key underlying virtualization technology in the Cloud is shown in Figure 1.3. It is evident from Figure 1.3 that there has been a consistent number of vulnerabilities reported publicly for these hypervisors. Hypervisors are the software that use virtualization technology to make sharing of the same physical machine among different hosts as virtual resources possible. A hypervisor can directly run on top of the hardware or be embedded in the host operating system as part of the OS. A hypervisor running directly on top of the hardware is a Type-I (bare metal) hypervisor, and a popular choice of bare metal hypervisor is the Xen hypervisor [27]. On the other hand, a hypervisor that is a part of the host OS is a hosted or a Type-II hypervisor. An example of such a hypervisor is the Microsoft virtual PC [28]. Figure 1.3 depicts the number of vulnerabilities published for two popular hypervisors Xen [27] and vSphere [29] in the National Vulnerability Database (NVD) [30]. Xen is an open-source virtualization software, whereas vSphere is a propriety technology from VMware. Besides, the Cloud entails a plethora of different services/components (both at the users and management levels), and factoring in these services/components increases the number of vulnerabilities published manifold. Thus, it is critical that the security of the Cloud is assessed considering the holistic view of the operations to minimize the likelihood of a security violation.

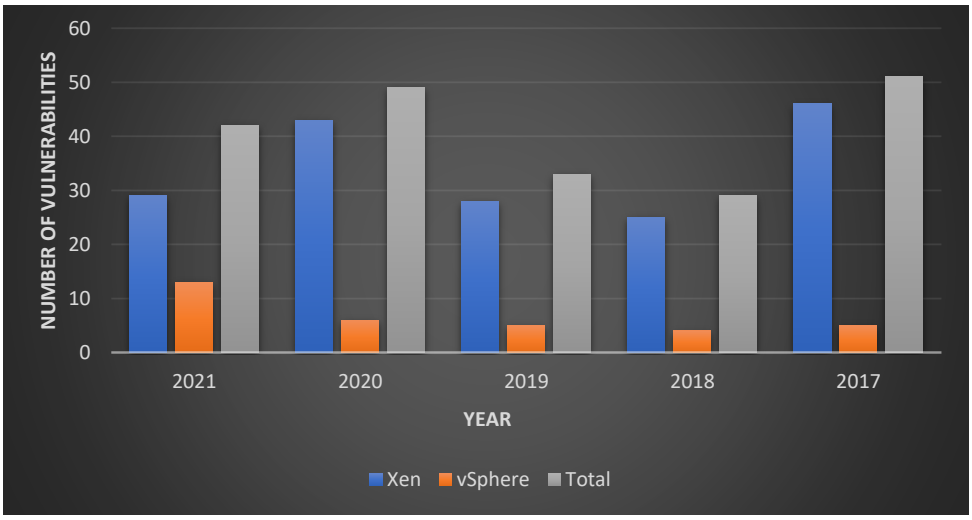

Fig. 1.3 Publicly disclosed vulnerabilities for Xen and vSphere

Among the advocated techniques to address security issues is modeling and analysis of threats targeting a system. Threat analysis is a process to identify threats targeting an organization's assets. These assets could be hardware, services, or an organization's data, as shown in Figure 1.4. Microsoft led initial efforts that pioneered STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of service, Elevation of privilege) classification of threats facing an organization [31, 32]. For instance, a service can be a target of a Denial of Service (DoS) attack or data can be accessed without authorization; hence, identifying threats targeting the assets is critical to evaluate the security of the organization.

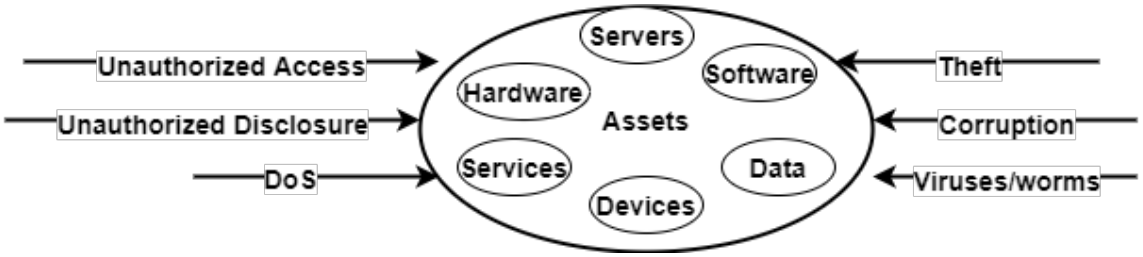

Fig. 1.4 Threat analysis for assets

To address security concerns in complex Cloud environments, multiple threat analysis approaches have been proposed that investigate threats at either a systems level [33, 34], in the context of specific assets/technologies [35], or by exploring potential attack surfaces in the Cloud that attackers could use to violate security requirements [24]. Examples of asset-based schemes include, among others, threat analysis for evaluating cache side-channel attacks [36], analyzing network attacks [37], web attacks [38] or analyzing the impact of

different threats on Cloud storage systems [39]. These schemes evaluate threats to the targeted asset in isolation without considering its operational behavior in the system.

The alternate graphical model-based techniques, e.g., attack trees/graphs, have also been applied to identify attack patterns that could potentially undermine the security of the Cloud. For instance, the authors in [40] developed a model of the Cloud data center and applied attack trees to identify potential paths leading to a security violation. Similarly, in [41], the authors proposed a security assessment methodology targeted specifically at the Cloud clients.

However, performing threat analysis in the Cloud is challenging given the lack of transparency from Cloud providers and the complexity raised by the existence of both the physical and virtual components. Additionally, these resources can migrate, instantiate, and decommission on the run-time. While the previously mentioned approaches provide useful threat analyses, they are either limited to identifying threats in a particular asset or typically assume the interconnections among the assets to be static. This hinders their effective applicability to Cloud environments, which are dynamic over their supporting on-demand adaptive resource provisioning. Furthermore, the limited capabilities of contemporary analysis techniques in incorporating user/service-specific security requirements within the Cloud threat model leads to incomplete security analyses. For example, a content delivery application might prioritize availability, whereas confidentiality may be prioritized instead in a financial or medical information system. Consequently, this thesis investigates assessing the security of the Cloud holistically to ascertain attack paths and organize the security threats according to user's requirements to prioritize threats accordingly. Moreover, the thesis also investigates the propagation of threats in complex and dynamic Cloud environments and the potential violation of users' requirements signed in the Service Level Agreement (SLA).

1.3 Research Questions and Contributions

This thesis is driven by the research questions stated below, and their investigation extends the field of research by making the contributions summarized under the respective research questions. The common theme underlying this work is the **dynamic security assessment of the Cloud at different operational abstractions**, proposing threat analysis techniques independent of the technologies used in the Cloud and assessing violations of users' SLAs. To this end, the first question aims to assess the Cloud's security at different operational abstractions/layers. The second question investigates the interplay between threat propagation and the services interaction. The third question explores variants of threats from the published vulnerability databases to design a requirement-based threat analysis. The fourth

question investigates the validation of a service level agreement between a user and a Cloud Service Provider (CSP).

Research Question 1 (RQ1): How can Cloud Service Providers (CSPs) examine the security of the Cloud at different abstractions of the operations?

A multitude of issues affects the broader adoption of Cloud computing, with security arguably among the most significant. To address security concerns, threat analysis is advocated to assess potential attacks that can undermine security objectives. However, conducting threat analysis for the Cloud is a non-trivial task, given the complexity of the Cloud environment entailing both the physical and virtual resources. Moreover, numerous attack surfaces exist in the multiple layers of the operational stack and the resource/customer interfaces. Consequently, many Threat Analysis (TA) techniques exist that focus on analyzing threats to targeted technology/services to reveal threats pertinent to them. However, these techniques are technology-dependent, and their applicability beyond the targeted technology is limited. Furthermore, these techniques do not contemplate the operational constraints on the technology, such as its interaction with other services/technologies in the Cloud. Thus, this question investigates assessing the security across the operational stack of the Cloud.

Contribution 1 (C1): A Cloud model capable of representing the fundamental operations of a Cloud in a technology agnostic manner.

The feasibility of assessing the security of the Cloud at different abstractions strongly depends on the model of the Cloud used. Generally, state-of-the-art focuses on the services layer of the Cloud, i.e., on modeling an application's behavior for performance optimization or assessing the security issues in the application. However, analyzing an application for security issues does not reveal potential attack surfaces in the underlying operational stack of the Cloud, which is critical to assessing the security holistically. For example, a Cloud facing a denial of service attack might inadvertently limit the availability of an application, although the application is not directly attacked. Thus, to comprehend the threats facing the Cloud, representing the operations of the Cloud is essential to determine the presence of attack surfaces at an attacker's disposal. Furthermore, the model needs to be technology-agnostic and applicable to the spectrum of Cloud offerings. Therefore, the first contribution this makes is by proposing and developing a Cloud model that is agnostic to technologies yet represents the functional operations of the Cloud and is aligned with the existing Cloud deployments. A multi-layer Cloud model is proposed to highlight different abstractions of

the operations involved during the life cycle of a VM. The model's foundation is the set of common services abstracted from multiple open source Cloud computing environments and influential stakeholders in the Cloud market, such as Amazon, Microsoft, and Google.

Furthermore, the model supports capturing the services interaction and the flow of information across the multiple layers. It is achieved by depicting the Cloud's functionality through a set of conditional transitions that are triggered after their respective preconditions are satisfied. Specifically, the model contributes to (a) enabling the exploration of the benign behavior of the Cloud and (b) analyzing the interactions amongst the services across various operational layers of the Cloud. This contribution, detailed in Chapters 3 and 4, is based, partly verbatim, on the material from [42, 43].

Research Question 2 (RQ2): How can the effects of a threat in a service on other interconnected services be identified?

A plethora of highly effective Threat Analysis (TA) techniques exist that focus on analyzing threats to specifically targeted assets (e.g., components, services), typically considering static interconnections among them. However, the Cloud is a dynamic environment where new interconnection occurs at run-time when new resources are instantiated or a resource migrates across physical hosts. In addition, the increasing number of multi-layer/multi-asset attacks highlights the need for threat analysis approaches specifically designed for dynamic and multi-layer Cloud environments. Fundamentally, this question investigates the attack paths that attackers could use considering the holistic view of the Cloud operations (i.e., services interactions across the layers in the Cloud) and the visibility of these services to the attacker. Thus, this question investigates the threat propagation across the Cloud to analyze (a) the spectrum of malicious behaviors stemming from the vulnerable service interactions across the multi-level operational stack and (b) correspondingly enumerate the multi-level attack surface exploitability by the attackers.

Contribution 2 (C2): A path-illustrative approach to profile threats, analyze their impact on targeted services and the propagation of threats across the multiple layers of the Cloud.

The second contribution of the thesis is the development of threat analysis approaches, Threatpro [44], and AttackDive [42], that assist in identifying paths that lead to the violation of the security requirements, i.e., an attack on the system. These approaches propose new ways to track and trace threats in dynamic systems such as the Cloud. The threats are introduced to the Cloud model as additional constraints at different services to assess their

impact on the operations of the Cloud. The inclusion of threats at different layers/services enables the investigation of the cause-effect relationship between a threat and a service. Consequently, it facilitates analyzing the propagation of the threats in the Cloud.

Both ThreatPro and AttackDive simulate (a) the benign Cloud behavior, (b) the impact of a threat on a single service, and (c) the impact of multiple threats to varied services across different abstractions of the Cloud. Furthermore, these techniques can perform speculative analysis using the vulnerabilities reported in the national vulnerability database to identify the corresponding attack scenarios. The speculative analysis assists in identifying potential paths that an attacker could use to undermine a security requirement. This contribution, detailed in Chapter 5, is based, partly verbatim, on the material from [44, 42].

Research Question 3 (RQ3): How can the Cloud service providers gather and organize knowledge concerning the interplay between security threats and a user's requirements?

The high number of disclosed vulnerabilities poses a challenge for system administrators to assess each vulnerability's potential to compromise a system. The current classification of vulnerabilities is based on classifying them into general categories such as denial of service, etc. The sole criterion for this classification is the vulnerability consequence. Consequently, this research question investigates if there exists a relationship between vulnerabilities that go beyond their common consequences. If a relationship exists, to what extent are different vulnerabilities related to each other in terms of their common indicators (e.g., similar attack mechanisms). This helps system administrators to prioritize the vulnerabilities considering their potential variants and according to the security requirements of a user. For instance, a content delivery application might prioritize threats on availability, while an application dealing with financial records might prioritize threats targeting confidentiality. Thus, this question investigates the relationship between security requirements, threats, and the targeted services to perform a requirement-based threat analysis that can (a) prioritize threats according to the requirements and (b) considers both the threats and their variants in the analysis.

Contribution 3 (C3): Development of requirement-based threat analysis to prioritize threats according to the user's requirements.

An exploration into the relationship between different vulnerabilities reveals a similarity between the vulnerabilities that go beyond the common consequence. For instance, the analysis showed that the correlation among the attack mechanisms exploiting the same service is higher than the attack mechanism exploiting different services. Moreover, an attacker

relies on reusing the *similar* attack mechanism to exploit the same service. Therefore, if the attack mechanism is mitigated, it will compel the attacker to find new and innovative ways to compromise the system, which is challenging. Thus, the proposed approach considers vulnerability variants to perform the threat analysis comprehensively.

Moreover, the relationship among different actors involved in the Cloud ecosystem is also investigated using the Design Structure Matrix (DSM), which comprehensively covers requirement specifications, interaction among the Cloud services, and vulnerabilities violating the requirements. The proposed approach obtains security assessments from varied actor perspectives by applying different algorithms to the design structure matrix to identify the most critical/influential, for instance. By systematically identifying the Cloud vulnerabilities and their potential variants, the Cloud can better be protected. This contribution, detailed in Chapter 6, is based, partly verbatim, on the material from [45, 46].

Research Question 4 (RQ4): How can Cloud users assess that their security requirements (qualitative and quantitative) are satisfied by the Cloud Service Provider (CSP)?

A multitude of issues affects the broader adoption of Cloud computing, with the perceived lack of trust in the Cloud Service Providers (CSPs) often listed as a significant concern. To address this, CSPs typically set up Service Level Agreements (SLAs) that contractually list what the CSP is obligated to provide to meet the customer requirements. While SLAs are promising as a concept, the inadequacy of monitoring and validating SLAs' run-time compliance limits the user's capability to evaluate the offered services to assess the actual trust to put in the CSPs. This question investigates the validation of an SLA and evaluates any violations from the CSPs. The SLA consists of a list of service attributes that are required by the user and committed by the CSP. In case of an SLA violation, a user is entitled to compensation.

Contribution 4 (C4): A customer-centric approach to assess the fulfillment of security requirements by the CSP.

Cloud service providers expect their users to simply *trust* the CSP services offered to them but not every user is willing to grant this trust without justification. It should be possible for users to establish that the CSPs actually fulfil their SLA requirements and that services are provisioned accordingly. With this aim, a methodology is proposed to validate SLAs and detect service violations over the life of the service. The SLA consists of a list of service

attributes that are required by the user and committed by the CSP. In order to validate an SLA, each SLA attribute, either qualitatively or quantitatively, is evaluated.

The evaluation provides reproducible assurance to a user for trusting the CSP based on fulfilling the user's requirements. In the case of a requirement violation, the severity of the violation is assessed by calculating an impact factor. The impact factor determines the degree of deviation of the provided value from the required value and consequently dictates the change in the level/state of trust in the CSP. The requirement violations are classified into "trust states" according to user preferences. The CSP can be in varied trust states depending on the severity level of the violations. The customer can periodically apply the proposed methodology to assess the behavior of the CSP over the life of the service. An assessment of the trust state of a CSP based on the services involved in launching and migrating a virtual machine in an IaaS offering is demonstrated. This contribution, detailed in Chapter 7, is based, partly verbatim, on the material from [43].

1.4 Thesis Organization

The remainder of the thesis is organized as follows.

Chapter 2 reviews contemporary threat analysis approaches for the Cloud. Threats affecting individual components/assets at each stage of the VM life-cycle are presented. Since the VM spends more time at the life cycle's execution stage, the threats that impact this stage are specifically focused. Furthermore, the threats are categorized using the STRIDE threat model to signify their impact. Conversely, threat analysis techniques in the Cloud that utilize attack graphs/trees as an underlying approach are also discussed.

Chapter 3 presents the design of the Cloud model that sufficiently encompasses services from the deployment of the Cloud in the wild. For the development of the Cloud functional model, multiple open-source Cloud computing platforms, as well as Cloud deployments by leading companies, were surveyed to abstract the common services and develop a multi-layer model of the Cloud. A sequence diagram is developed highlighting the interactions and the information flow among the services during the operations of the Cloud. The model forms the basis for investigating the interactions among the services during the fundamental operations of the Cloud, i.e., launching a virtual machine at the request of the user.

Chapter 4 translates the functional Cloud model to an information flow model that is agnostic to the underlying technologies used in the Cloud. Furthermore, the model is formally developed using Petri nets to represent the functional behavior of the Cloud. Petri nets are specifically designed for concurrent and dynamic systems as the transitions in Petri nets

could be fired concurrently as soon as the precondition of the transition is satisfied. The functional behavior of the model is simulated to enumerate the correct sequence of operations without the presence of a threat at any service in the Cloud.

Chapter 5 builds on the information flow model presented in Chapter 4 by adding threats as additional constraints to the information flow model. This assists in identifying the changes in the system introduced as a result of a threat. The approach is validated by assessing the capability to trace and analyze real-world attacks. Specifically, two prominent case studies are used to perform post-mortem analysis using the presented approach. Moreover, the threat analysis process includes the approach to performing speculative analysis. The Chapter also discusses the capabilities for predictive analysis, the potential for the plug and play services, and the limitations of this approach.

Chapter 6 extends the threat analysis approach by including both the variants of a threat and the user requirements in the process to explore the relationship between threats, security requirements, and the respective services. A design structure matrix based approach is developed to perform requirements-based threat analysis. For instance, multiple algorithms can be applied to the matrix to extract influential actors. Moreover, the use of natural language processing is presented extract variants of threats that could also potentially pose security risks to the Cloud.

Chapter 7 focuses on validating the SLA signed between a user and a CSP. A methodology is presented that compares the SLA attributes provisioned by the CSP and requested by the user. A violation in any of the attributes results in downgrading the trust state of the Cloud. A user can periodically run the proposed methodology to validate that the CSP is fulfilling SLA.

Chapter 8 concludes the thesis and outlines future works.

Chapter 2

Background & Related Work

This Chapter provides an overview of the contemporary threat analysis approaches for the Cloud. For simplicity, the approaches are broadly categorized into (a) asset-based techniques used to explore potential threats in specific assets/components and (b) graphical security models used to identify potential attack paths leading to a security requirement violation. Furthermore, as mentioned in Chapter 1, the main focus of the thesis is on the IaaS offering of the Cloud as it provides more flexibility to the users. Therefore, the reviewed approaches focus on threats during the life cycle of a VM, which is a fundamental operation in the Cloud IaaS. Before proceeding with the threat analysis, a brief overview of the VM life cycle is presented.

2.1 VM Life Cycle

Among the primary functions of the Cloud IaaS is offering and managing virtual resources as VMs [47, 48]. It is achieved by virtualizing the underlying hardware and sharing it with the users. Therefore, it is essential to investigate threats during the life cycle of a VM. The typical stages involved in the life cycle are the: creation stage, storage stage, deployment stage, execution stage, exit stage, and delete stage. Moreover, there are optional stages that can occur during the execution stage. For instance, a VM can migrate, and a user can suspend the VM or take a snapshot of the existing VM state. Figure 2.1 shows the VM's life cycle stages, briefly described below, before reviewing the threats targeting each stage. The solid line in the Figure depicts the sequence of the stages in launching a VM. However, in certain cases, some stages can be skipped. For instance, a user can delete the VM after the creation stage, as shown in the Figure with dotted lines.

- **Creation stage:** The VM is created, and a host Operating System (OS) is installed on the VM. To expedite the OS installation process, publicly available disk images from the Cloud providers can be used to set up the host OS.
- **Storage stage:** During this stage, persistent storage is assigned to the VM. This stage is optional if a user decides not to have storage assigned.
- **Deployment stage:** During this stage, the Cloud controller selects a potential server to host the newly created VM. The controller node is responsible for managing servers, optimizing load among the servers, and correspondingly identifying the server that could deploy the VM.
- **Execution stage:** The VM gets networking capabilities (e.g., IP/MAC address), a mapping between the virtual and the physical network interfaces, and starts executing on the host server.
- **Exit stage:** This stage occurs when a user shuts down the VM. The VM's data is saved to persistent storage before shutting the VM down. It is a temporary stage, and the user can restart the VM.
- **Deletion stage:** All the contents associated with the VM are permanently deleted. In contrast to the exit stage, the data is not saved to persistent storage; hence, both the data and the VM are not recoverable after this stage.
- **Suspension stage:** This is an optional stage where the VM is suspended, i.e., it relieves its assigned CPU and memory, but the data is saved to a file in case the user restarts the VM.
- **Migration stage:** To optimize the performance of either the host server or the VM, a VM might migrate to another server which can be cold or hot migration [49]. To optimize the performance of either the host server or the VM, a VM might migrate to another server with better performance. The migration can be a cold or hot (live) migration [49]. The VM is shut down before moving to the alternative server during the cold migration. On the other hand, a hot (live) migration does not require the VM to be shut down; thus, it minimizes the cost and interruptions associated with the process of the migration [50, 51].
- **Snapshot stage:** The VM operational state is saved to a file that can be used as a template to create additional VM disk images. A VM snapshot can also be used to roll back the VM to the saved state.

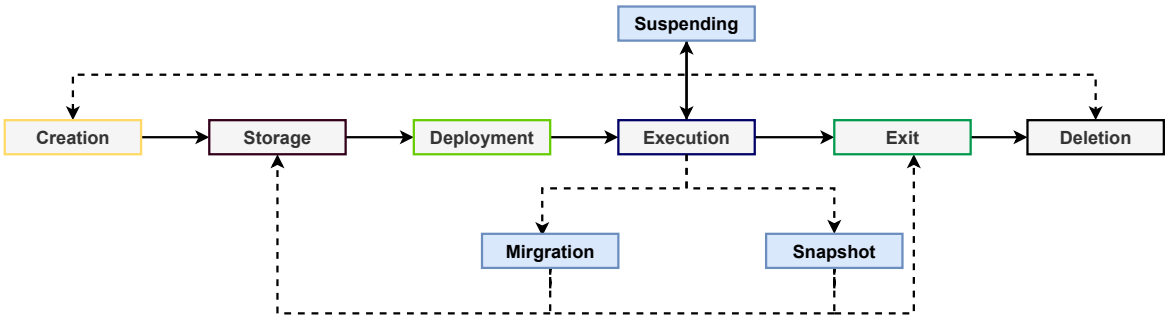

Fig. 2.1 An overview of the VM life cycle in the Cloud.

It is an overview of states a VM can go to during its life cycle. The optional stages (e.g., migration) during the life cycle of a VM are also presented for completely covering the life cycle. The following sections present threats against each stage and the corresponding threat analysis approaches.

2.2 Asset-based Threat Analysis

Asset-based threat analysis aims to uncover threats and their impact on discrete assets (e.g., components, services, interfaces, data) without factoring in operational considerations. The initial efforts led by Microsoft introduced a threat modeling approach called STRIDE [52]. It is an attacker-centric approach, applicable to data flow diagrams to find potential weaknesses and security flaws that an attacker can exploit. The STRIDE analysis primarily focuses on the architectural flaws of the system. It classifies threats into spoofing, tampering, repudiation, information disclosure, denial of service attacks, and elevation of privileges. The holistic security assessment of the Cloud necessitates understanding the threats during the entire life cycle of a VM. The following sections review the threats to different stages involved in the life cycle of a VM.

2.2.1 Creation Stage

As mentioned in Section 2.1, the first stage in launching an instance of VM in the Cloud is the creation stage. At this stage, the host OS is selected. Therefore, the asset that is considered is the published disk images of the Cloud provider. However, this could lead to security issues, as Bugiel et al. [53] explored the threats associated with using public disk images. The authors analyzed publicly available virtual machine images in Amazon EC2 [2] repository. Their analysis focused on the public interfaces to extract private information stored in these

machine images. From the extracted data, further attacks can be launched, such as starting a botnet or launching an impersonation attack. In some cases, they could access seven different source code repositories through the extracted credentials. These sources contained highly valuable information, such as hard-coded passwords for administrators. In a few instances, the authors extracted the customer's SSH keys that are used to authenticate the user to the Amazon EC2. Similarly, in [54, 55], the authors qualitatively analyze infrastructure as a code script to investigate security issues within the code repositories. Their investigation also found more than 1200 hard-coded passwords within the code that would allow an attacker to access the application.

2.2.2 Storage Stage

Several TA approaches exist that target specific technologies such as the storage managed by the Cloud provider and assigned to the VM. A typical asset considered during this stage is data at rest and during transmission. The threat facing this stage typically involves data loss or leakage, which could breach the confidentiality of a user. Moreover, data integrity is also critical to ensure that the data is not altered or modified without authorization. For example, the authors analyze the impact of different threats in Cloud brokerage systems in [39]. They developed an attack tree-based approach to elaborate on the paths an attacker could follow to compromise the storage system in the Cloud.

Similarly, the Cloud storage deployment options, respective attacks, and the corresponding countermeasures were surveyed in [56]. Additionally, in [57–59], authors discussed issues concerning the data in the Cloud. They proposed a framework based on the best practices that could serve as a reference architecture for securing data in the Cloud.

2.2.3 Deployment Stage

During this stage, a server is selected by the controller node that can host the VM. Among the preconditions to exploit side/covert channel attacks is the co-residency of both the victim and the attacker on the same physical host. Therefore, the reviewed literature targets exploiting the VM placement algorithm to achieve co-residency. Among these works are from [60, 61] that manipulate the VM placement algorithms to co-locate with the victim and subsequently launch a side-channel attack. The basic premise behind the approach is to use networking tools (e,g., Nmap [62]) to extract the victim's location and subsequently manipulate the controller node to co-locate with the victim.

In a similar effort, the authors show the confidentiality breach of a customer by co-locating tenants [63]. Their approach exploited the placement algorithm of the Amazon EC2

to be co-located with the victim's VM and used side-channel analysis to extract the victim's private key.

2.2.4 Execution Stage

This stage is where VM spends most of its time during the life-cycle; therefore, the number of attacks targeting this stage is manifold. Moreover, this stage involves the co-existence of both the VM and VMM (hypervisor). Thus, protecting the VM and VMM (hypervisor) is critical to isolate the VMs from different users. Therefore, VM and VMM are the assets that are considered during this stage I will discuss both of these aspects of virtualization technology in the following.

In [64], the authors characterized vulnerabilities in the hypervisor according to their impact on the functionality of the hypervisors. A hypervisor provides around 11 functionalities, including assigning virtual CPUs to VMs, emulating Input/Output (I/O), and networking functionality. Moreover, VM exits and hypercalls critical functions of a hypervisor. VM exits are necessary for fulfilling a guest OS request that requires root privileges, and hypercalls are the same as system calls in a traditional OS setup. There are two ways a hypervisor can be used to virtualize the resources.

- Type-I/bare-metal hypervisor: This is the configuration where the hypervisor is directly installed on top of the hardware. A privileged VM (Dom0) manages the user's VM running on the system.
- Type-II/hosted hypervisor: The hypervisor is hosted inside the operating system. Host OS processes are used to manage the resources of the user.

Both of these configurations are shown in Figure 2.2.

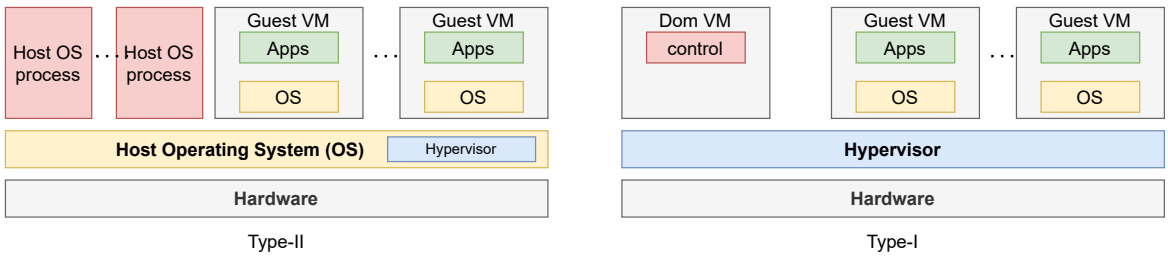

Fig. 2.2 Difference between Type-I and Type-II hypervisors.

The work in [64] analyzed attack vectors that attackers can use to compromise the functionality of a hypervisor. Table 2.1 presents a triggering source of an attack, the attack

vector, and the target that can be achieved using the given attack source and the vector. For instance, if the attack source is OS (marked as ✗ in Table 2.1), a potential attack vector could be VM exits, and the potential targets of such a scenario could be either the OS or the hypervisor. The ✗* in the Table 2.1 represents that an attacker can use the source to launch a second stage of the attack. For example, the attack source *user* could be used to target VM management to compromise Dom0, which enables attackers to ultimately compromise the hypervisor or install malware using the compromised Dom0 [22, 65].

Table 2.1 Xen's (hypervisor) vulnerability map

Attack Source				**Attack Vector**	**Attack Target**		
Network	User	OS	Dom0		OS	Dom0	Hypervisor
	✗		✗	Virtual CPUs	✗	✗	
	✗	✗		VM exits	✗		✗
✗	✗	✗		I/O and Networking	✗	✗	
	✗	✗	✗*	VM Management		✗*	✗
		✗		Hypercalls			✗

Some recent works have demonstrated the value of threat analysis in evaluating cache side-channel attacks [36] to explore the possibility of using the cache to compromise the confidentiality of tenants hosted on the same physical machine. It has been shown that exploiting the VM placement algorithm to co-locate with the victim and using cache as a side-channel could lead to a complete breach of confidentiality [63, 66–68].

The security issues of virtualization at the VM level were analyzed by Tsai et al. [69]. They focused on exploring the security issues on VM hopping, VM mobility, VM diversity, and VM denial of service. They assessed the impact of virtualization (in)security in different Cloud deployments such as Platform as a Service (PaaS), Software as a Service (SaaS), and Infrastructure as a Service (IaaS). Authors in [35] made a similar effort to create a comprehensive survey on threats, their modeling, and the corresponding attacks in a distributed virtualized system.

On the other hand, authors [22] showed that installing malicious software on the VM is possible without being detected. Figure 2.3 shows before and after the infection affects the targeted OS. The fundamental premise behind their approach is to encapsulate the victim's OS into a VM, allowing greater control over the system. Since the security layer or the

security applications are embedded in the OS, creating a VM and moving the entire OS to the VM evades the security layer of the OS. Therefore, inspecting and analyzing malware using the traditional technique becomes challenging, consequently giving attackers more control over the system.

In a similar instance, authors in [65] conducted a survey of possible configurations/settings that would allow attackers to install malicious software on the VM. The authors proposed a framework based on the application of dynamic analysis [70] to detect malware in modern systems, including the Cloud systems that inherently use virtual machines. The proposed framework suggests running dynamic analysis tools both inside the host OS and inside the guest OS to understand the deviation in the behavior of the overall system, which could form the basis for further inspection.

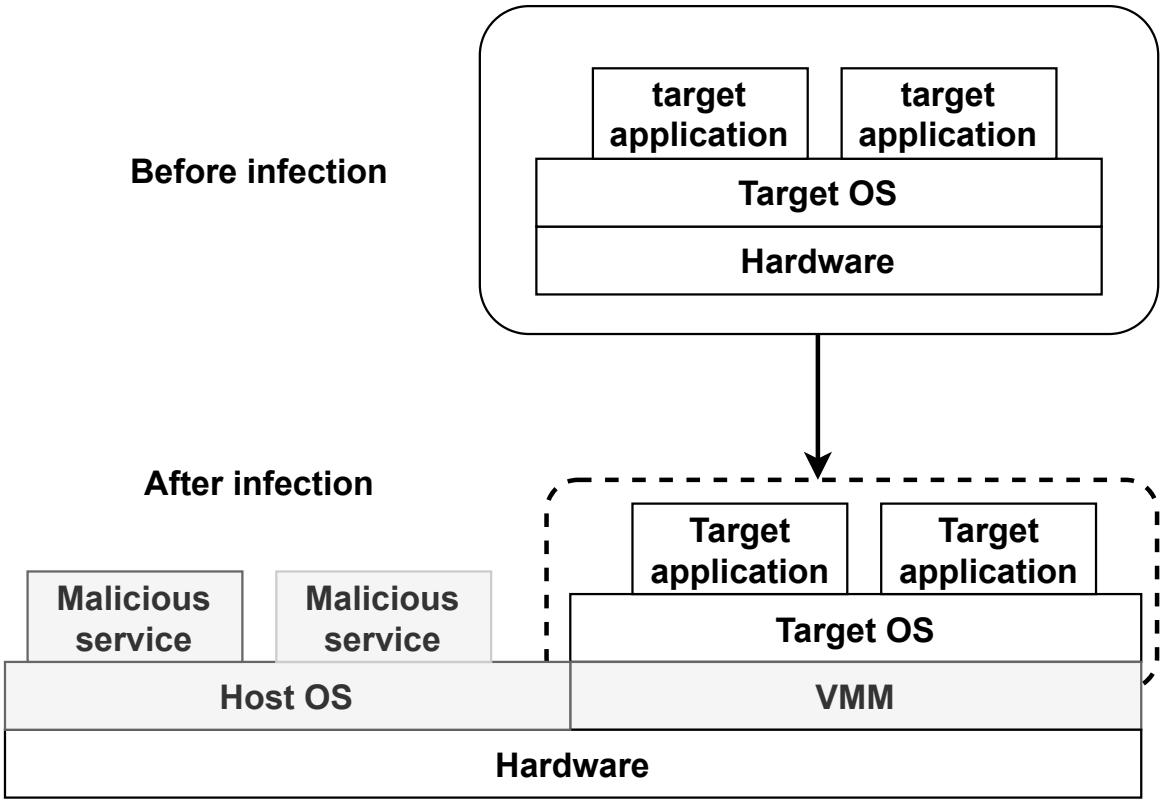

Fig. 2.3 Using virtualization to install malware on a VM.

Additionally, modeling the application's behavior and applying probabilistic model checking to investigate the impact of elasticity on security requirements was investigated in [71]. Furthermore, the outcome of the analysis can be used as feedback to fine-tune the behavior of the Cloud for governing its elasticity.

2.2.5 Exit and Deletion Stages

External attackers to the Cloud are one aspect of the security assessment. However, insider threats also pose security concerns in the Cloud. For instance, a malicious Cloud admin could potentially delete the VM [72] and a user could lose all his data. Furthermore, due to the lack of transparency in the operations of the pubic Cloud deployment, it is challenging to know how many replicas of the content exist, where they exist, and who has access to them [73, 74].

2.2.6 Migration Stage

This stage occurs when a controller node migrates the VM to balance the workload among the servers due to over-provisioning certain servers. Moreover, a VM can also migrate if it requires additional resources that the existing host cannot provide. As mentioned earlier, VM migration could be either cold or hot. The difference between these strategies is that in the latter case, the VM is not shut down before the migration; hence, it minimizes user interruptions.

A VM can be attacked during the migration phase as the network is a shared environment among the users. Therefore, these schemes are focused on the shared network as an asset to be secured. Multiple VMs of a user can be hosted on different hosts that are linked together through networking to provide a coherent view of the resources to the user. However, since the network is a shared environment, it is prone to man-in-the-middle attacks as both the target and attackers share the environment. Moreover, the attacker could also be an insider attacker that manages the Cloud. Thus, it is critical to understand the security concerns raised during this stage, as the migration can happen without a user's intervention. A typical scenario of man-in-the-middle attack is shown in Figure 2.4. As seen from the Figure, during the migration phase, the content of the VM is transferred unencrypted over the network, allowing both passive and attacks to be applied to the data [72, 75, 76]. In a passive attack, an attacker only sniffs the data and does not modify the data. However, in an active attack, an attacker can arbitrarily change the VM OS or application running on top of the VM.

Similarly, authors in [77, 78] created two new zero-day attacks using the network as their primary attack surface to launch their attacks. The basic premise behind the approach is the exploitation of a shared networking environment among both the victims and attackers. The VMs are assigned Virtual Network Interface Cards (VNICs) to manage their network stream, which is then transmitted to the Test Access Point (TAP) device. Typically, the network traffic from a VM is sent from the VNIC to the Internet through the TAP, veth pair, virtual interface (virtual switch), and finally, the physical interface of the host. A veth pair directly

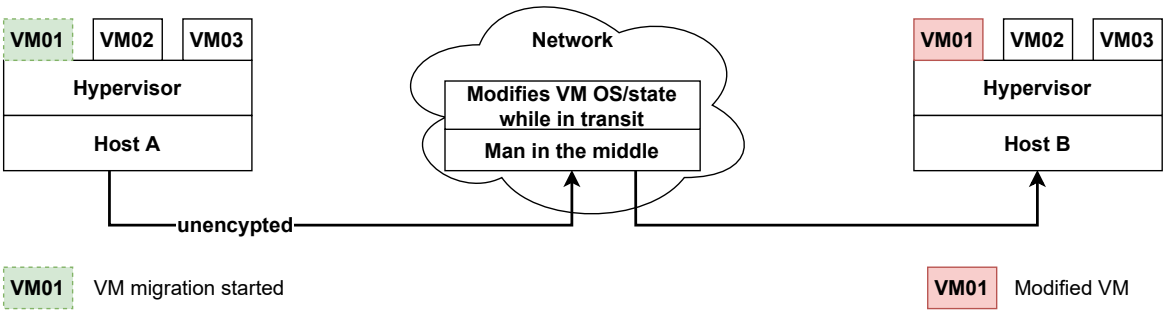

Fig. 2.4 Man-in-the-middle attack during the VM migration

connects to virtual network interfaces, and a bridge that maps VNICs to the respective MAC addresses exists that binds a MAC to the respective IP address. Thus, a TAP offers a path to access the data that is being transmitted through a network. Detailed information on the networking can be found in [79, 80].

The authors exploited an architecture flaw in the Cloud that allows bridging a TAP interface with no valid private Ethernet interface with the VM and using it to redirect the traffic towards that TAP. The resulting attacks from exploiting the vulnerability were impersonation (sniffing VM content and impersonating it), and privilege escalation attacks on the VM using return-oriented programming [81, 82]. The steps in their attack are detailed below with Figure 2.5 showing the visualization of the attack.

- An attacker creates a dummy network device without an active VNIC. It can be easily achieved by either creating a new networking device or disconnecting an existing device. It is a precondition of the attack to de-link any existing links between the VM and the TAP.

- The newly installed device is impersonated as a TAP device. It can be achieved by removing information about the interface identity and consequently making the network card act like a TAP device. The primary reason for this impersonation is that only TAP devices can be mapped to the veth pair. After the impersonation occurs, connectivity requests are sent to the switch that adds it to the list of the conventional TAP device. This switch also contains the interfaces of all the other VMs located on the host. Furthermore, to evade detection by other users, the identity of the impersonated TAP can be hidden.

- The final step in the attack involves creating a network mirror, and the dummy interface is configured as a destination point by assigning zeros as the IP address. The objective

of such an assignment in a mirror network is to redirect all the traffic to the destination point, i.e., the attacker's VM.

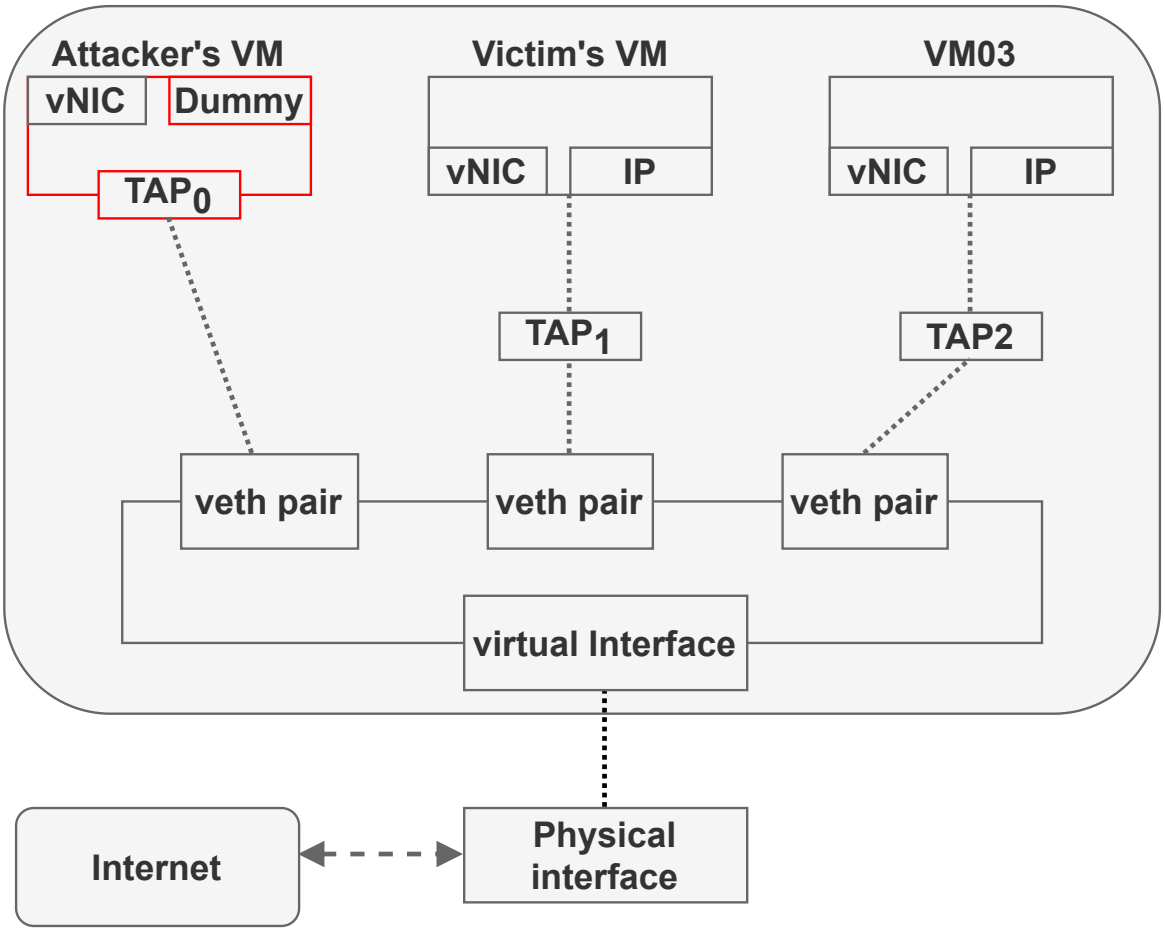

Fig. 2.5 Using network as an attack surface to launch attacks

A shared networking environment can be used to launch attacks against other VMs located on the same host. Much research has been devoted to the virtualization of the hardware, but limited work is done on exploiting networking as a potential attack surface. The application of model checking to verify the violation of security property has been demonstrated in [37]. The primary objective was to analyze network attacks violating the defined security property. The authors developed a formal network model and used model checking tools to generate test cases that would violate the specified security property.

2.2.7 Categorizing Threats using the STRIDE Model

As mentioned earlier, Microsoft developed a threat model called STRIDE [52] to classify threats into categories: spoofing, tampering, repudiation, information disclosure, denial of

service, and elevation of privileges. These categories are briefly explained before utilizing these categories to classify the threats impacting VM life cycle stages.

- **Spoofing:** This threat demonstrates the case when an attacker (or a program) successfully identifies as another user (or a program) by falsifying the data to gain an illegitimate advantage. For example, an attacker can spoof the victim's IP address to receive its network traffic.
- **Tampering:** This threat concerns the active modifications by attackers on the system components. These components could be data, networking communications, or the component's functionality. In contrast to the spoofing attack, tampering attacks are active attacks, and the purpose of such attacks is to directly interfere with the system's operations.
- **Repudiation:** This threat refers to the situation where attackers can deny performing specific actions, and the lack of evidence proving and linking the actions to the attacker is challenging. For example, an attacker can spoof an IP address to avoid network traces; thus, this would fall into the category of a repudiation attack.
- **Information disclosure:** This threat demonstrates the possibility of information leakage to unauthorized persons. The leakage of information could be intentional or accidental. For example, unintentionally leaving default passwords would allow attackers to gain access to the information, or an attacker can manipulate the authentication service to bypass authentication and access the information.
- **Denial of service:** The denial of service threat aims to deny legitimate users from accessing the service, which violates the system's availability. A typical example of such an attack is to overwhelm a service by launching repeated requests using a distributed network of bots which will eventually exhaust system resources such that legitimate users cannot access the system.
- **Elevation of privileges:** These attacks allow privileged access to unauthorized persons. Typically, this is achieved by exploiting a vulnerability/flaw that exists in the system to bypass the authentication system or the access control mechanism used within the system.

The categories presented in the STRIDE cover a wide range of threats. In Table 2.2, I summarize the threats presented in Section 2.2 according to the STRIDE model. Furthermore, the consequence of each threat on confidentiality, integrity, and availability is also demonstrated in the Table. For example, the techniques presented in [53–55] target the VM creation

stage, and the corresponding attacks on the VM creation stage are spoofing, tampering or information disclosure attacks. The respective consequence of these attacks is highlighted under each of the CIA fields in the table.

Table 2.2 Examples of prevalent issues and actions.

Life-cycle stage	Threats	Techniques	C	I	A
Creation	Spoofing	[53] [54] [55]	✓	✓	
	Tampering	[53] [54] [55]		✓	
	Information disclosure	[53] [54] [55]	✓	✓	
Storage	Tampering	[39] [56]		✓	
	Information disclosure	[56] [53]	✓		
Deployment		[60] [61]			
Execution	Information disclosure	[63] [66] [67] [68] [22]	✓	✓	
	Tampering	[22]		✓	
	Denial of service	[72]			✓
	Spoofing	[53]	✓		
Exit & delete	Denial of service	[72]	✓		
	Tampering	[73] [74]		✓	
Migration	Elevation of privileges	[78] [77]		✓	
	Spoofing	[78] [77]	✓	✓	✓
	Tampering	[72] [75] [76]		✓	

This section covered threat analysis techniques focused on the Cloud's components in isolation without considering the operational constraints on the targeted components. Thus, the threats covered targeted each stage of the VM life cycle and the corresponding threat analysis approaches are detailed that demonstrate their applicability in mitigating threats to the respective stage. The second dimension of threat analysis techniques is detailed in the following section. This dimension of the threat analysis uses attack graphs/trees as underlying methods to generate attack paths in the Cloud.

2.3 Graphical Security Models

Multiple graphical security models have been developed for distributed systems in general and for Cloud, in particular, to visually trace and identify attack paths/patterns that could potentially undermine the security of the Cloud. Primarily, these have been in the form of attack trees and graphs.

The attack tree demonstrates the path attackers can take to achieve their objectives [83]. A commonly depicted attack tree is shown in Figure 2.6. The root of the Figure is the attacker's objective, i.e., an attacker would like to get access to the database. As the Figure shows, two paths are available at the disposal of attackers to fulfill their objective.

- **Path 1:** The first path starts with creating a phishing email and then sending the email to the target. The target opens the email, which could lead to accessing the database.
- **Path 2:** Alternatively, attackers can try to access the admin computer, and that would enable them to get access to the database. To access the admin computer, an attacker can breach physical security by accessing the admin's office or by gaining access to the computer remotely.

Either of these paths enables the attacker to access the admin computer and consequently allow access to the database. It should be noted that each action (or a precondition) in the attack tree can be assigned with a likelihood of achieving that action. For instance, setting *Get Office Key* action with a probability of 0.1 indicates the chance of physical security breach. Assigning probabilities to different attack actions enables the system defenders to prioritize the attack paths and, accordingly, focus on these attack paths. These probabilities can be set using policies enforced in the organization.

I have presented a generic attack tree in Figure 2.6, and now I review state-of-the-art schemes that use attack trees as an underlying methodology to identify attack paths in the system. The approach presented in [84] develops an attack tree of the Cloud and then utilizes a "what if" analysis to traverse paths in the tree to find a potential exploit(s). However, they consider only an abstract model of the Cloud and hence fail to reveal attacks pertinent to the interaction of the services across varied levels of the Cloud's operational stack. The authors also introduced a defense tree to reduce the countermeasure cost by selecting the optimal asset from the defense tree concerning the attacker's profile. In [85], the authors formally analyzed different open source Cloud environments to assess the correctness properties. A Petri nets based model was proposed in this work, though, with the limited target of verifying the correctness of these open source Cloud environments. Their approach generated the

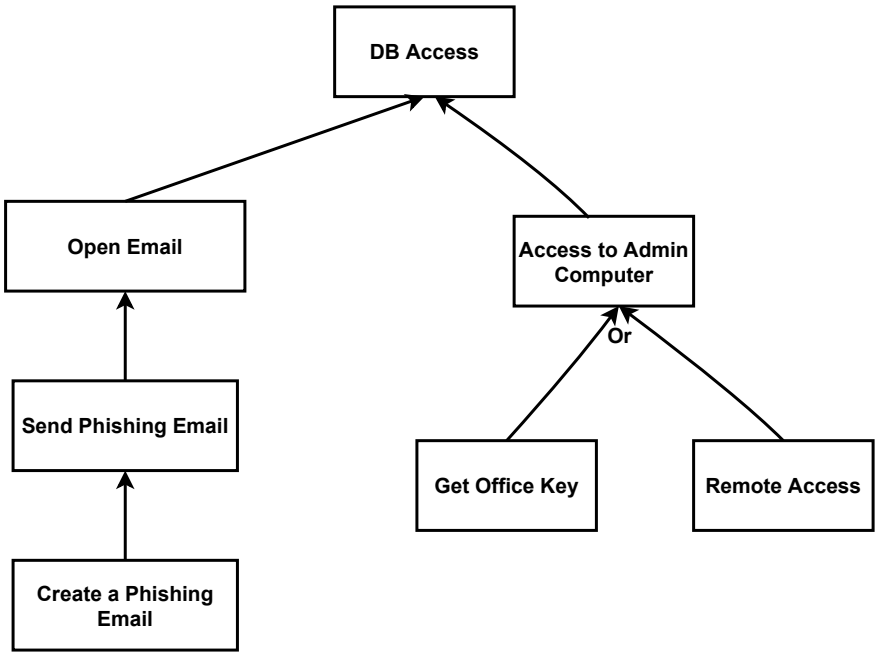

Fig. 2.6 An attack tree example

computational tree for the open-source Cloud environments to verify the property that a valid request from the user should always terminate correctly.

Similarly, a Petri nets based approach [36] modeled the side-channel attacks in the Cloud. Furthermore, based on the modeling, they presented potential mitigating strategies that will work considering the characteristics of the side-channel attack. For instance, if the side-channel attack uses Flush+Reload [15] strategy, an appropriate countermeasure to that was discussed in the paper.

On the other hand, an attack graph is more fine-grained and details the causal relationship among the nodes [86]. An example of using both the attack trees and graphs is presented in [40]. The authors did the modeling of a Cloud data center and applied both the attack trees and attack graphs to identify potential paths leading to an attacker's objective. An illustration of the attack tree from the paper is shown in Figure 2.7. As the Figure shows, there are multiple paths that attackers can take to achieve their goal of getting storage data. These paths are briefly described below.

- **Path 1:** This path can be used by either a Cloud admin or a normal user. The objective for the attackers is to access the storage device that holds users' data and consequently breach the confidentiality of the users. To achieve this, the attacker must first access the VM, which is typically allowed. Therefore, the probability of achieving the action is one and is shown under the corresponding action. As mentioned earlier, each action

can be assigned a probability to denote the likelihood of the action. For instance, the number 1 underneath *Have_http_VM* in the Figure means that the attacker will have access to it. However, the attacker must also access the underlying host Operating System (OS)/hypervisor. This can be achieved by utilizing attacks presented in the execution stage of the VM. After gaining access to the underlying OS/hypervisor, the attacker can control VMs running on the host. Subsequently, the attacker can access the VMs located on a different host as all the VMs belonging to the user are connected via networking. The attacker has to select the VM hosting the database, allowing the attackers to access data stored on the storage device.

- **Path 2:** This path covers the possibility of Cloud admins or third-party providers accessing the authentication service. If they get access to the authentication service, they can change the image of the VM, and by modifying the VM image, they might gain access to the storage data. On the other hand, attackers can assess if the VM has default passwords stored in the image used for the VM by analyzing it using the techniques presented in Section 2.2.1 and Section 2.2.4. In this case, the attacker will gain access to the stored data. Alternatively, an attacker can publish a new disk image with a backdoor installed that can later be used to access the user's data.

- **Path 3:** This attack can be used by either a normal user or an attacker to obtain data from the storage device. Similar to the first action in Path 1, the user must first access the VM and must gain root access to this VM. It is the precondition for both attack paths 1 and 3. After gaining root access to the VM, the attacker can connect with other VMs associated with the user, including the VM hosting the database. After that, the attacker must gain root privileges by exploiting vulnerabilities related to the database service hosted on the VM. Once root access is acquired, the attacker can access the user's stored data. However, to access the data of other users, the attacker must compromise the hypervisor, which can be achieved in any of the ways presented in Section 2.2.4.

Subsequently, to explore the attack paths, the authors proposed an attack tree metric to evaluate the likelihood of attack paths that could potentially compromise the system's security. Additionally, Bayesian network metric [87, 88] was proposed for attack graphs to prioritize the paths at the attacker's disposal.

Similarly, the quantification of the user's security requirements is proposed in [89]. A risk assessment framework for a sensor environment deployed in the Cloud was presented in [90]. The objective was to illustrate the cause-effect relationship and apply security measures that correspondingly minimize the attack's impact. On the other hand, concepts from requirement

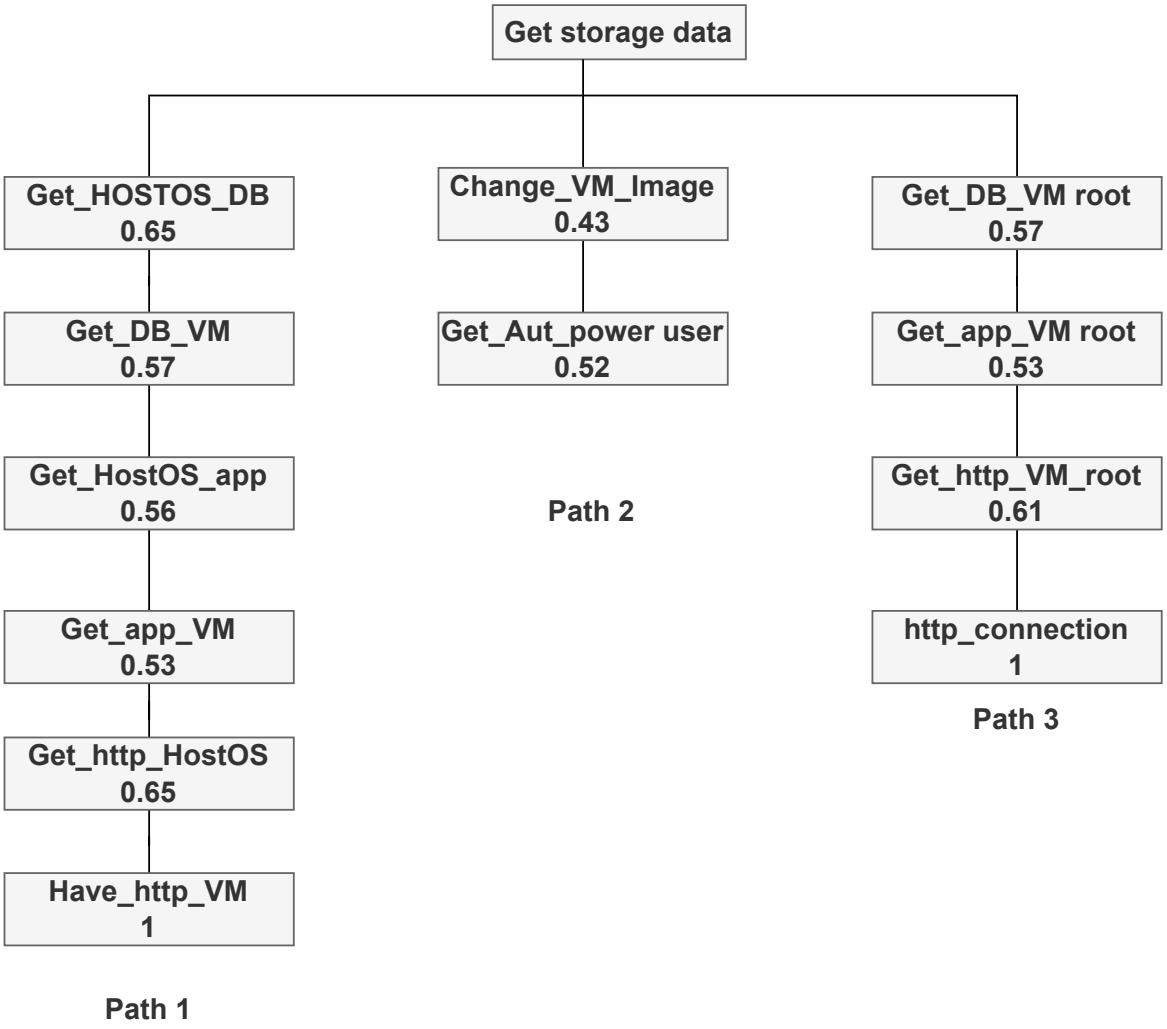

Fig. 2.7 Example of attack paths of the Cloud

engineering have been utilized in [91] to propose a methodological approach to elicit users' security and privacy requirements and select the appropriate Cloud provider. The approach performs a cost-benefit analysis for the users, enabling them to make an informed decision about migrating to the Cloud.

The attack/defense tree application has been detailed in [92]. The approach investigated the interplay between attacks and the respective countermeasures and proposed a framework to assess the associated risks of the applied countermeasures. Similarly, in [93], the authors present a similar approach by combining both the attacks and their respective countermeasures in a single tree. They developed a qualitative analysis to assess the return on investment for the countermeasure and prioritize the countermeasures that give a better return on investments.

The work in [94] proposed a graphical security model using Bayesian attack graphs to quantify the likelihood of the network compromise, which feeds into an attack mitigation plan. This enables system administrators to make an informed decision by considering the trade-off between the attack and the mitigation strategy. A reference model of the Cloud incorporating the security controls and best practices was developed in [95] to assess the security posture of the Cloud offerings for confidentiality and integrity. This was achieved by estimating probabilities of advanced persistent threat infiltration in the Cloud. The underlying technique utilized a Bayesian network model that examines attack paths and assesses their impact on both confidentiality and integrity requirements.

Overall, these schemes leverage attack graphs/trees to explore potential paths that identify a security violation. Furthermore, quantifying the risks associated with each path is fundamental to many of these schemes, which enables system administrators to prioritize the paths and the mitigation strategy accordingly. On the other hand, these schemes assume that the attack paths are static and the functional behavior does not create new interconnections at run-time. This assumption does not hold in the inherently dynamic Cloud environment, where new interconnections might be introduced at run-time through VM migration or by instantiating a new VM.

A Table summarizing the presented techniques and their capabilities is shown in Table 2.3. The fields in the Table depict the method used in the respective technique and whether a Cloud model was developed in the paper. Furthermore, the Table also highlights if the proposed technique was agnostic to the underlying technologies used in the Cloud. The user requirement field determines if specific user requirements were considered for prioritizing the threat analysis accordingly. For example, [40] used both the attack tree and graph in their methodology on the Cloud they developed based on the Cloud deployment adopted by market leaders, e.g., Microsoft and Google. However, they did not consider user requirements in their methodology or the dynamic interactions in the Cloud due to resource migration.

2.4 Conclusion

As can be seen from both Tables 2.2 and 2.3 and identified in Sections 2.2 and 2.3, both asset-based threat analysis and graphical security models are effective techniques to analyze potential threats targeting a system. However, their effectiveness is limited in analyzing threats considering the holistic view of the Cloud's dynamic operations. For instance, asset-based schemes consider assets in isolation without operational factors and reveal threats pertinent to the specific asset. On the other hand, graphical models assume that the interconnection among assets is static and hence, cannot analyze threats in a dynamic

Table 2.3 Application of the graphical security models

Method	Literature	Cloud Model	Technology agnostic	User requirements
Attack tree	[84]	Limited	✗	✗
	[40]	✓	✓	✗
Attack graph	[40]	✓	✓	✗
	[88]	✓	✓	✗
Attack & defend tree	[92]	✗	✗	✗
	[93]	✗	✗	✗
Probabilistic attack graph	[94]	Limited	✗	✗
	[87]	✗	✗	✗
Formal models	[85]	✓	✓	✗
	[36]	✗	✗	✗
Risk based attack tree	[90]	✗	✓	✗
	[91]	✗	✓	✓

environment. Moreover, the techniques based on attack trees/graphs are inherently limited to the selected threats, and any changes in either the threats or the service require restarting the complete process. Thus, in this thesis, I fill this gap by proposing techniques that can incorporate (a) the asset's operational environment, (b) dynamic interconnections across resources/services, and (c) specification of the user's security requirements to provide a comprehensive threat analysis process applicable to the Cloud.

Chapter 3

Designing and Modelling the Cloud

The state-of-art focuses on modeling the behavior of an application in the Cloud for performance optimization. However, to understand the threats facing the Cloud, a model to represent the functional operation of the Cloud is necessary. Moreover, capturing the services' interactions forms the basis for analyzing the potential of threats propagation across the Cloud. Thus, this chapter designs a functional Cloud model that is applicable to a wide range of Cloud offerings and is aligned with the Cloud deployments in the wild.

3.1 Functional Cloud Model

Several delivery models exist for the Cloud, such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). The emphasis is primarily on functionality and performance. Moreover, these models do not reveal the interactions among the services that an attacker can exploit to laterally move in the system. On the other side, a considerable body of research exists for modeling and analyzing the behavior of an application in the Cloud [96–98]. However, ascertaining threat propagation requires modeling the functional behavior of the Cloud to capture the interaction across services and investigating the interplay between the service's interactions and the threat progression. However, work related to modeling the Cloud functionality is very limited. The work that focuses on the development of the Cloud is either technology-centric or too abstract that these models cannot be used to examine the behavior of the Cloud. Among the primary functions of the Cloud IaaS, is offering and managing virtual resources as VMs [47, 48]. These VMs are created through virtualization technology, an enabling technology to share a physical host with the VMs. [99]. Thus, I define an abstract model for the Cloud emphasizing the interactions of services during the life-cycle of a VM [100] as presented in Section 2.1. The service interactions during the life-cycle of a VM are conceptualized after

surveying multiple open-source Cloud computing environments [101, 102] as well as Cloud deployments adopted by market leaders such as Amazon, Google, and Microsoft. The model, depicted in Figure 3.1, exhibits a 3-layer architecture of the Cloud consisting of the Control Layer, Infrastructure Layer, and Storage Layer, where each layer performs distinct functions. The model is flexible and can be extended to include vendor-specific services at each layer. However, for the scope of this thesis, the focus is on modeling the functionality in launching a VM as it is a fundamental offering of the Cloud IaaS.

3.2 Defining the Functional Model of the Cloud

Following the overview in Section 3.1, this section details the representation of the Cloud's functional behavior as a model. The reasons for developing such a model are twofold. Specifically, there is a lack of both (a) a generalized Cloud model applicable to the spectrum of Cloud offerings, and (b) approaches that can analyze the interplay between the functional behavior of the Cloud and the attack paths. To develop such a model, I first extracted common services from multiple open source Cloud computing environments [102, 101] and major stakeholders in the Cloud market, such as Amazon, Microsoft and Google. There are obvious differences in terms of the Cloud architecture and network configurations adopted by each vendor. For instance, the controller node could be distributed across the data center. However, these differences are technology and optimization-driven and therefore fall out of the scope of this thesis. Table 3.1 presents the primary concepts used in the major Cloud providers in the market which is adopted from [40].

Table 3.1 Configurations adopted by Cloud providers

	AWS	**Azure**	**Google Compute**
Multiple layers	✓	✓	✓
Controller node per cluster	✓	✓	✓
Authentication server	✓	✓	✗

It is evident from Table 3.1 that there exist similarities among the configurations of the Cloud in major Cloud providers. For instance, all the presented providers have adopted a multiple layers approach, but each layer's functionality varies from vendor to vendor. Furthermore, the users must be authenticated before accessing their assigned resources or creating additional resources in the Cloud. Depending on the vendor, the authentication service can be centralized or distributed across the Cloud. For example, both Amazon

and Microsoft have a centralized authentication service, and Google adopted a distributed authentication service configuration. Similarly, all these vendors have a controller node for each cluster, and the goal of the controller node is to optimize resources and balance the workload among the servers.

Using these reference models and surveying multiple open-source Cloud computing environments, the Cloud model is developed and presented in Figure 3.1. The model depicts a generalized 3-layered (Control, Infrastructure, and Storage) architecture focusing specifically on the Cloud's functionality to be agnostic to the technologies implementing the functionality. Each demarcated layer performs a specific task in the life cycle of a VM. The role of the control layer is to authenticate users and enables them to request new VMs. The infrastructure layer receives the request, creates the respective VM, and links it with the existing resources of the user. The storage layer provides storage capabilities for the data. The functionality of each layer is detailed in the following sections.

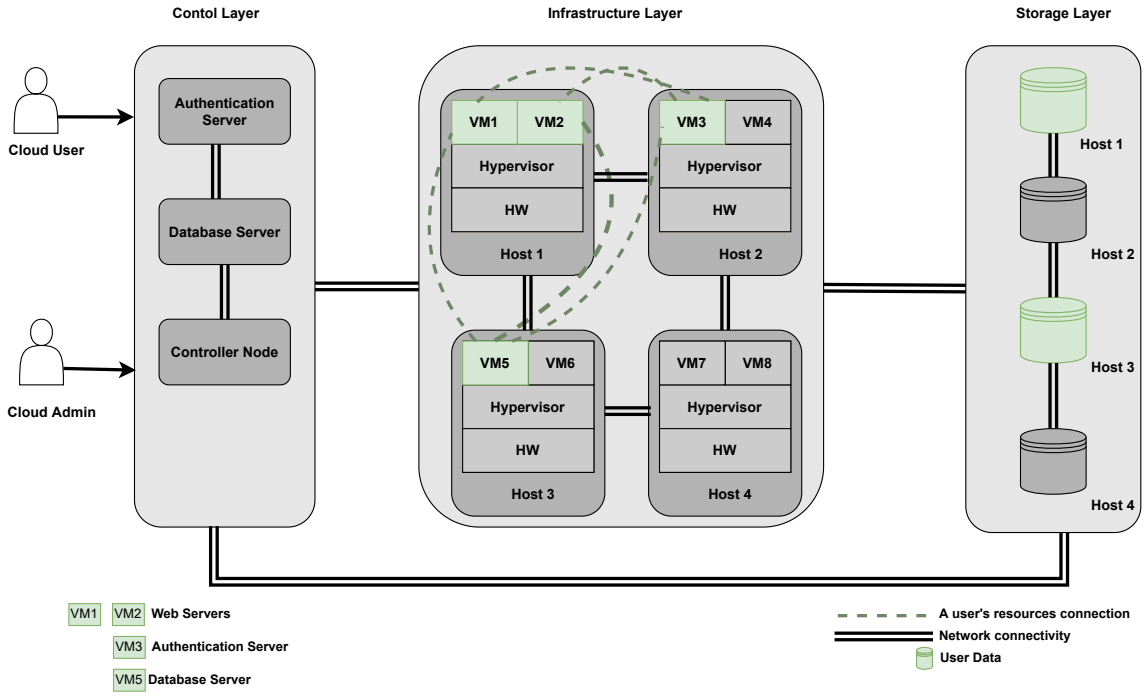

Fig. 3.1 Multi-layer architecture of the Cloud

3.2.1 Control Layer

The control layer, consisting of an authentication server, database server, and a controller node, orchestrates the managing and scheduling of the Cloud resources — physical and services — for the Cloud administrator and the users. For a user requesting Cloud access, the authentication service authenticates and redirects the user to a resource dashboard. From the dashboard, a user can request a new VM instance or start an existing VM. The database server is responsible for maintaining a list of VMs allocated to the user. The controller node, under the control of the Cloud administrator, allocates the resources to a data center and migrates them in case of over-provisioning. Overall, the control layer is responsible for allocating and managing a user's resources that are scattered across the data center to create a coherent view of the resources.

3.2.2 Infrastructure Layer

As the name suggests, this layer represents the actual physical hardware of the Cloud for binding the VM's to physical hosts. The core functionality of the layer is provided by a hypervisor [103] that runs on top of the hardware/OS along with other VM management tools. The hypervisor is the fundamental element in the virtualization technology that enables sharing the same physical host among multiple users. A request to launch a VM is transferred from the control layer to the infrastructure and after a successful instantiation of it, the VM is linked with other resources of the user. As shown in Figure 3.1, a user's resources can be dispersed across different servers/hosts. In this example, VM1 and VM2 are located on host 1 of the data center while VM3 and VM5 are respectively located on host 2 and host 3 of the data center.

3.2.3 Storage Layer

This layer provides storage capacity and delivers data when requested. This layer is also responsible for providing consistency among different data backups. As the placement of the VMs across different hosts is permitted, the data could also be distributed across different hosts. Additionally, the data can also migrate from one host to another similar to a VM.

These 3 layers collectively outline the operations of any generalized Cloud system. As VM management (creation, migrations, and deletion cf. Section 2.1) is the basic Cloud functionality, This thesis utilizes a VM-centric approach for threat propagation and analysis. In the following, I focus on the operations involved in creating a VM to illustrate the information flows across the operational layers of the Cloud.

The role of each layer in the Cloud is defined to determine the interactions among the layers and services in fulfilling users' requests, such as creating a new VM. The creation of a VM is among the core functionalities of the Cloud and, therefore, is primarily focused. Even in the case of VM migration, the destination VM having the same configuration as the source VM is created before the migration process is initiated. Thus, I focus on the operations involved in creating a VM and present how the information flows between the layers when launching a VM.

As noted earlier, there are architectural differences in the Cloud model adopted by each vendor. The model can be adapted to represent each vendor by adding the necessary services to each functional model's layers. For example, if the authentication service is distributed, then each layer will have an instance of the authentication service, and the functional model would become similar to that of AWS and Azure.

3.2.4 Information Flow in Launching a VM

As mentioned, the authentication service is the user's interface to the Cloud. A user can only launch or request a VM after being successfully authenticated. The details of subsequent transitions at each layer are as follows:

- *Control layer transitions:* Once authenticated, a user is transferred to a dashboard presenting the allocated VMs and the possibility of requesting additional VMs. If the user decides to launch a new VM, the requested VM configurations (e.g., CPU, RAM) are compared with the assigned quota. A valid request leads to the invocation of the scheduler service that determines a potential host for the requested VM. The VM configuration and the selected host are then passed to the infrastructure layer.

- *Infrastructure layer transitions:* The infrastructure layer receives the VM request and invokes the image repository service for the operating system and the network service for the networking capabilities (e.g., virtual Network Interface Card (NIC), IP addresses). Furthermore, the infrastructure layer interfaces with the storage service for allocating storage for the VM.

- *Storage layer transitions:* The primary responsibilities of the storage service are assigning storage to the VM and keeping the data among the backups consistent. This step is optional in case the user does not select the storage capacity for the VM.

- *VM:* After the configuration is finalized, the hypervisor instantiates the VM and it is added to the database against the corresponding user.

Utilizing the above information, a sequence diagram representing the information flow among different services is presented in Figure 3.2. The sequence of operations in the Cloud following a user's request to launch a VM is briefly explained in the following.

- The Cloud interface to the user is the authentication service that validates (or invalidates) the user based on the provided credentials. Users provide the credentials which are checked against the stored credentials. Following successful authentication, the users get access to the resources attached to their accounts and the respective access privileges are granted to the users.
- In case a user requests a new VM, the configuration of the VM (*reqVM* in Figure 3.2) is checked against the quota of the user, and the VM request is sent to the controller node.
- The controller node manages and maintains physical servers. The scheduler service probes different servers that can host the new VM and selects an appropriate server. Through the scheduler service, the controller node selects the host server that can launch the requested VM.
- After the host is selected, networking capabilities are provided to the VM, and the virtual machine disk image is installed. In some instances, the disk image can be selected when launching a VM request.
- The final configurations (*createVM(config, account, MAC/IP, DI)* in Figure 3.2) is then pushed onto the hypversior (VMM) which launches the VM.
- The VM starts executing and is added to the list of the user that requested the VM.

The presented sequence of operations relates to creating a new VM. However, it should be noted that the Cloud provider can initiate the VM instantiation or migration to optimize the workload without the user's input but in compliance with the Service Level Agreement (SLA) signed between the user and the Cloud Service Provider (CSP). The migration of a VM entails creating a new VM with the same configurations as the old VM and migrating the content of the old VM to the newly created VM. Thus, VM migration also contains the VM creation sequence of operations. Therefore, it is critical to understand the service's interaction and flow of information that leads to the VM creation. For completeness, the service's interactions during the VM migration are shown in Figure 3.3 and explained in the following.

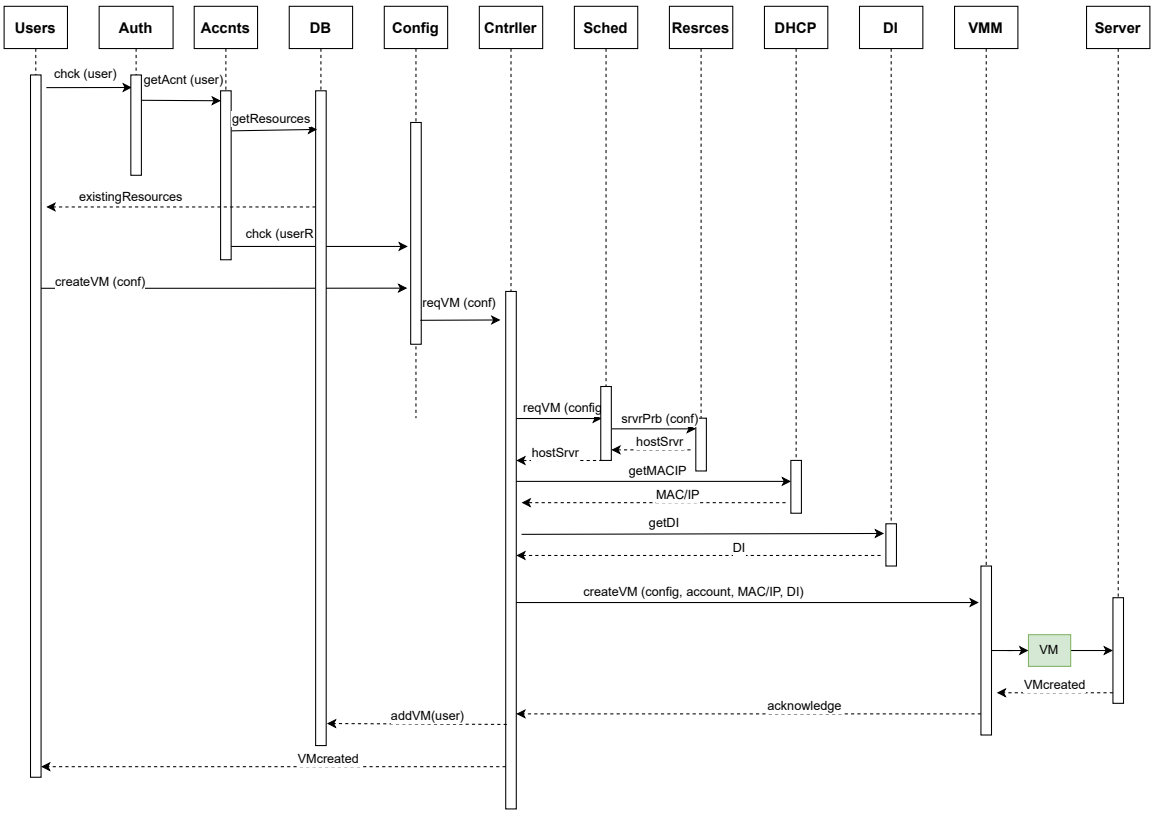

Fig. 3.2 Communication among the services to launch a VM

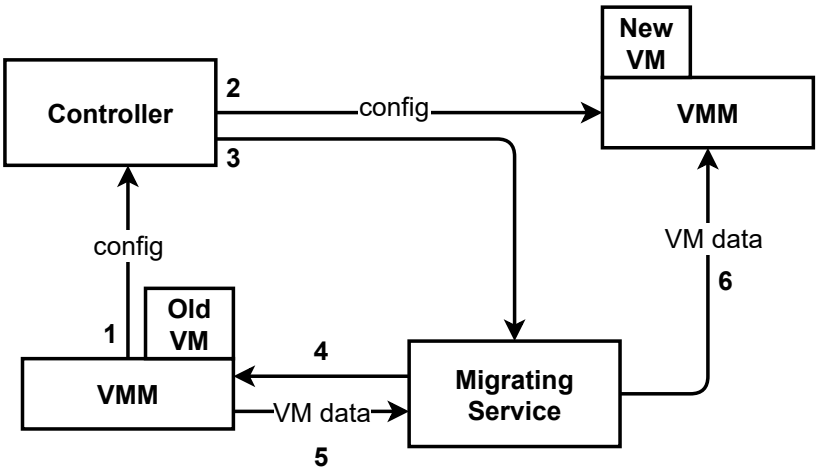

Fig. 3.3 Communication among the services in migrating a VM

- The Cloud admin can trigger the VM migration to optimize load among the servers or it can happen dynamically to fulfill a specific user's requirements. The VMM forwards the configurations of the VM to the controller node.
- The controller node selects a new physical server that can host the VM. This is shown as transition 2 in the Figure 3.3.
- The controller node initiates the migrating service to start the migration process.
- The migrating service takes data from the old VM and transmits the data to the new VM over the network. The data transmission over the network is usually unencrypted and VM is susceptible to certain attacks that are explained in Section 2.2.4.

3.3 Conclusion

The Chapter presented a Cloud model that represents the functionality of the Cloud in launching a VM. The functional model is a multi-layer aligned with Cloud deployments in the wild. The control layer manages and maintains the Cloud and authenticates users. The infrastructure layer schedules resources on the hardware and manages the workload amongst the servers. The storage layer provides the storage capabilities to the users/VM. The Chapter also introduced a sequence diagram to identify the flow of information and the services interactions in the Cloud. Moreover, the Chapter also presented the communication among the services during the migration of the VM. Understanding these interactions is critical to evaluating the security of the Cloud at different functional abstractions.

Chapter 4

Information Flow Model

This Chapter builds on the Cloud functional model and the sequence diagram presented in Chapter 3 to create a technology-agnostic information flow model of the Cloud operations. Information flow models [104] have been applied to various areas of computer science. This chapter explains the thesis's first contribution, i.e., developing an information flow model independent of the underpinning technologies used in the Cloud. The development of such a model entails abstracting the technology and vendor-specific characteristics to create a transition system governed by rules that trigger transitions following the fulfillment of the respective preconditions. For example, the authentication credentials provided by the user are a precondition to trigger different transitions depending on the validity of the credentials irrespective of the underlying authentication technology used to check these credentials. In the case of valid credentials, a user is directed to a dashboard/interface to access their VMs. On the other hand, invalid credentials lead to an error message, and the user is requested to reenter credentials. Thus, defining the pre-conditions and rules that govern the triggering of transitions and passing of the information among the services represent the functional behavior of the Cloud. Furthermore, the security requirements of the users are incorporated into the information flow model to support the prioritization of threats that violate specific requirements. It is argued that a security requirement of an application varies depending on the functionality of the application. For example, a content delivery application might set the availability of the data as a high priority while an application dealing with financial records might consider confidentiality as its primary requirement. Therefore, considering such security requirements is critical since it helps to identify threats that may lead to their violation.

Following the overview, the rest of the Chapter details the development of an information flow model of the Cloud. The requirements for the information flow model are: (a) that the model should support expressing the functional behavior of the Cloud as well as the threats

in a technology-agnostic style, and (b) that there should be the ability to identify violations from the sequence of events by determining the modifications in the operations of the Cloud caused by spurious input to the system. These specifications are achieved by defining rules and constraints that determine the triggering of transitions after their respective preconditions have been fulfilled. Consequently, I begin with a basic transition system representing a functional behavior and rules that determine the transitioning among the states. Subsequently, I leverage the rule-based transition system to represent a login system for users authentication and eventually represent the Cloud functional behavior. Furthermore, a threat's behavior is expressed as an instantiation of the rule-based transition system to use as a spurious input to the system.

4.1 A Transition System

Figure 4.1 presents an example transition system which demonstrates how a system whose functionality is represented as inputs causing a transition to a different state can be represented. The transition system forms the basis for analyzing the proper functioning of the system and

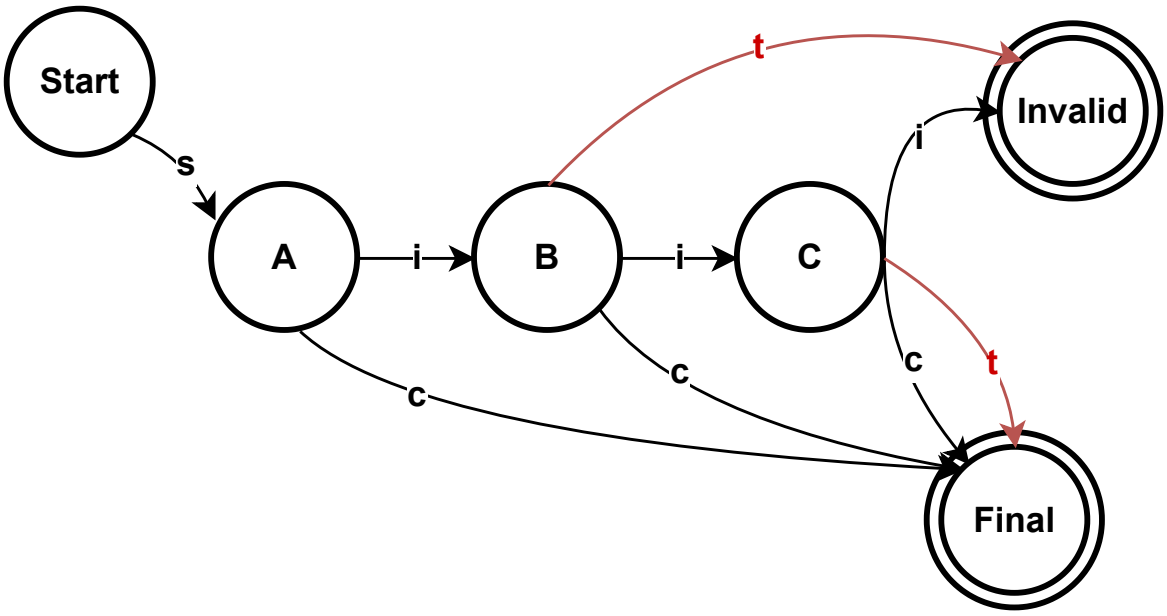

Fig. 4.1 An abstract example of a transition system

provides the capability to identify modifications in system actions caused by spurious inputs. I now describe the rules governing the transitions between states which eventually lead to a terminal state (`Final` or `Invalid` state).

4.1.1 Normal Behavior

There are multiple paths that represent the normal operation of the system. Any modification in these paths might be considered a threat to the system.

- Path 1: $\texttt{Start} \xrightarrow{\texttt{s}} \texttt{A} \xrightarrow{\texttt{c}} \texttt{Final}$
- Path 2: $\texttt{Start} \xrightarrow{\texttt{s}} \texttt{A} \xrightarrow{\texttt{i}} \texttt{B} \xrightarrow{\texttt{c}} \texttt{Final}$
- Path 3: $\texttt{Start} \xrightarrow{s} \texttt{A} \xrightarrow{\texttt{i}} \texttt{B} \xrightarrow{\texttt{i}} \texttt{C} \xrightarrow{\texttt{c}} \texttt{Final}$
- Path 4: $\texttt{Start} \xrightarrow{s} \texttt{A} \xrightarrow{\texttt{i}} \texttt{B} \xrightarrow{\texttt{i}} \texttt{C} \xrightarrow{\texttt{i}} \texttt{Invalid}$

Paths 1, 2, 3 and 4 demonstrate the correct functional behavior of the transition system, i.e., the paths start from the state `Start` and terminate to either the `Invalid` or the `Final` state. The inputs start, invalid, and correct are respectively denoted by $\{\texttt{s}, \texttt{i}, \texttt{c}\}$ and are used to trigger different paths depending on the input provided to the system. For instance, in path 1, an input triggers the state `Start` which passes on `s` as information to state `A`. The received input initiates multiple paths from state `A`, for instance, the input corresponding to a correct value `c` leads to the `Final` state. Conversely, an invalid input `i` at state `A` moves the system to state `B` and the similar process is followed at state `B`. However, at state `C`, an invalid input `i` terminates the system at the invalid state instead.

4.1.2 Incorporating Malicious Inputs to the System

The rules determine the functional behavior despite the different underlying technologies. The rules can be added (or removed) to incorporate new (or speculative) specifications or constraints from users/systems. In Figure 4.1, additional inputs are added to both states `B` and `C` to analyze their corresponding impacts on the behavior of the system. For example, at state `B`, an input `t` modifies the behavior and terminates the system at the invalid state instead of transitioning the system to either state `C` or the `Final` state. Thus, a rule-based transition system highlights manipulation in the system caused by malicious inputs and consequently enables the speculative (what-if) analysis. The complete paths for both the malicious input are given below.

- Path M1: $\texttt{Start} \xrightarrow{\texttt{s}} \texttt{A} \xrightarrow{\texttt{i}} \texttt{B} \xrightarrow{\texttt{t}} \texttt{Invalid}$
- Path M2: $\texttt{Start} \xrightarrow{\texttt{s}} \texttt{A} \xrightarrow{\texttt{i}} \texttt{B} \xrightarrow{\texttt{i}} \texttt{C} \xrightarrow{\texttt{t}} \texttt{Final}$

4.1.3 Representing a Transition System

I have demonstrated the benefits of using a rule-based transition system to enumerate the behavior of a system and speculate on the behavior by adding spurious constraints. I leverage this rule-based transition system concept to develop an information flow model of the Cloud depicting its functionality. There exist multiple methods to model the functionality of a system. In the following, two prominent alternatives of labeled transition system and Petri nets are detailed.

Labelled Transition System (LTS)

LTS has been extensively applied to model the Cloud operations, including modeling Client-Cloud interactions [105–107]. The benefit of using such models is to elaborate the behavior of a system and identify a potential violation of the specified property using a model checker. To this end, the complete model and the property specification are provided to a model checker that generates a counterexample identifying the property violation. The specified property is often a safety/liveness property, but the process can be replicated for certain security properties. On the other hand, LTS becomes cumbersome for concurrent systems due to the state explosion problem [108]. Further, the states and the associated actions in LTS are global, i.e., the complete state information is required to recognize the firing of a transition. A state cannot be distributed into multiple local states with different preconditions to trigger a transition locally if a certain precondition is satisfied. Moreover, these models are deterministic, while modeling the Cloud requires triggering of transitions at certain time intervals to replicate e.g., VM migration.

Petri Nets

An alternative to an LTS is a Petri nets, which can describe the functional behavior of distributed systems. Petri nets have been used to model workflow of concurrent systems [109], resource management in the Cloud [110], and fault detection in distributed systems [111]. A difference between Petri nets and labeled transition systems is the distribution of states as places in the former and enabling places to hold information to enable a transition. Moreover, the transitions are fired locally and non-deterministically without requiring a global view of the system. Furthermore, the Petri nets supports time-driven firing of the transitions, i.e., firing the transition at a specific time instance. Similar to LTS, Petri nets also encounter the issue of state explosion [108].

4.1.4 Information Flow Model Requirements

I have described the possible options for modeling the behavior of a system, and now I proceed to elicit the specific requirements for modeling the Cloud. The Cloud is a distributed and concurrent system, and modeling its functional behavior entails assigning information to each state and passing on either a complete or a subset of information according to the triggering event. Furthermore, certain events might create an impact both locally and globally. For example, a threat targeting a service affects that service, but can also progressively target the interlinked services. On the other hand, performing a speculative analysis requires assigning constraints (threat preconditions) to different services to analyze their consequence on the benign operation of the Cloud. An additional requirement is the capability to model time-driven events. For instance, a VM can instantiate, decommission or migrate at run-time according to the workload. These requirements favor the use of Petri nets for the development of the information flow model. A brief overview of Petri nets is presented before demonstrating its use in developing the information flow model of the Cloud.

A typical Petri nets has two elements, places, and transitions[1], depicted as circles and bars respectively, as shown in Figure 4.2. A transition signifies the occurrence of an event and the place holds the token (information) that enables the transition. The conditions that govern the flow of tokens are represented on the arcs between input and output places. The pre-conditions are represented on the arcs that connect places to transitions and the output flow (post-condition) from a transition governs the flow of token (information). A transition is fired only if both pre- and post-conditions are satisfied. A token from an input place is transferred onto the respective output place after the transition is triggered.

In this thesis, a variant of Petri nets called High-Level Petri nets (HLPN) [112] is used, which provides further flexibility in assigning multiple tokens of different data types to a place. Moreover, in HLPN, a subset of the token (information) can be passed onto the next state depending on the triggering condition. For example, the authentication service holds both usernames and passwords and passes on only the username to the next state that provides a list of the user's existing VMs. Furthermore, the constraint can be time-driven. For instance, after a certain time interval, a VM migration process can start requiring a new VM instance creation and the model needs to capture such dynamic interconnections. These dynamic interconnections are captured in the model through the time-driven firing of the transition. Moreover, the transitions are fired locally without contemplating the global state of the system. This enables the model to capture new VM instance requests during the VM run state or even when another VM request is at a later stage of the execution.

[1]Three different fonts are used in the thesis to make it clear what type of item within a Petri nets is being referred to. These are: a `Place` in the Petri nets, an Input provided, and a TRANSITION that can be taken.

Formally a HLPN is defined as:

Definition 1 *A HLPN is a 7-tuple $N = (P, T, F, \psi, R, L, M_0)$, where,*

- *P is a set of finite places*
- *T is a set of finite transitions such that $P \cup T = \emptyset$.*
- *F is a flow relation between a place and a transition such that $F \subseteq (P \times T) \cup (T \times P)$.*
- *ψ is a mapping function that maps places to data types such that $\psi : P \rightarrow Type$.*
- *R define rules that governs enabling of the transition $R : T \rightarrow Formula$;*
- *L is a label that maps F to labels such that $L : F \rightarrow Label$*
- *M_0 is the initial marking of the generated tokens where $M : P \rightarrow Tokens$*

The first three variables (P, T, F) provide the structural information of the Petri nets while the variables (ψ, R, L) provide the behavioral semantics of the Petri Nets. The working of Petri nets is elaborated using an example shown in figure Figure 4.2. There are three places `P1, P2, P3` and two transitions `T1`, `T2`. The data type of the places `P1`, `P2`, and `P3` are *integer*, *string* and (*integer*×*string*) (product of *integer* and *string*) respectively. The transition `T1` is enabled only if `P1` has a string token of value "`abc"` and `P2` has an integer token with a value greater than `a`. These are the preconditions represented on the arcs that connects `P1` and `P2` with `T1`. For completeness, an output condition on is also placed `T1` that governs the output flow. Thus `T1` is enabled and fired if both preconditions and post conditions are satisfied. Therefore, the rule for enabling and firing `T1` can be written as: $R(\texttt{T1}) = (X = \text{“}\texttt{abc"}) \wedge (\texttt{Y>a}) \wedge (X = \text{“}abc\text{”} \wedge Y > a + b)$. The rules are written following first-order logic [113].

The transition `T1` is enabled and fired as its rule is satisfied and therefore, the respective tokens from input places `P1` and `P2` are placed on the output place `P3`. This behavior depicts the flow of information between the places. Since the token `a` is placed on the place `P3` and thus, the rule for transition `T2` is satisfied and this in turn fires the transition `T2`.

I have presented a generic example to demonstrate the use of Petri nets in modeling a system. In the following sections, the application of Petri nets is extended to model interactions in the Cloud. Initially, a login system to authenticate users is modeled which is later extended to cover the operations during the life-cycle of a virtual machine in the Cloud.

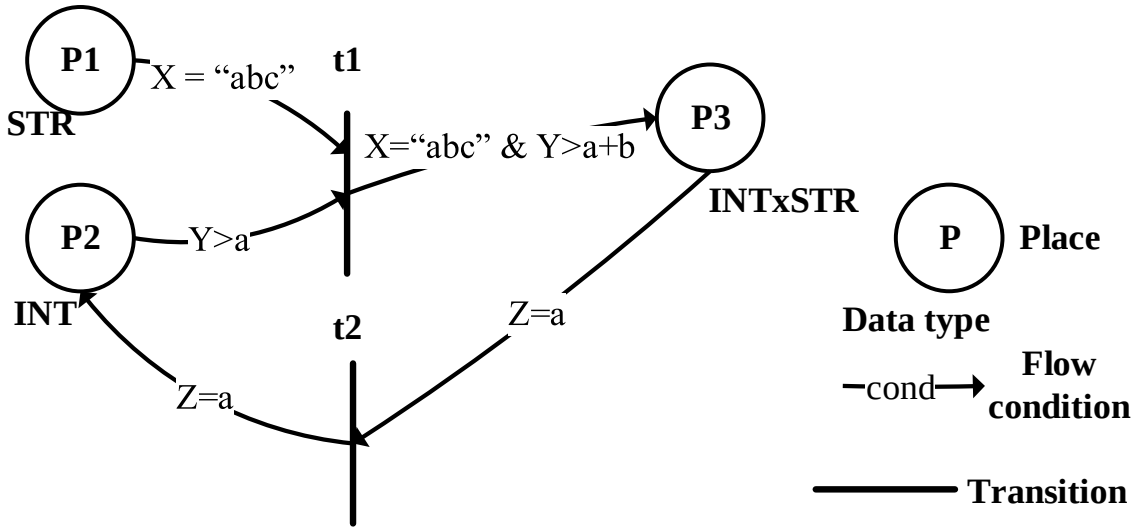

Fig. 4.2 An example of a Petri net

4.2 Modelling the Cloud operations

In the previous section, I have explained a basic transition system and rules that determine the functional behavior of the system through the flow of information among the states. Moreover, the advantages of using HLPN for the development of the information flow model are also explained. This section leverages the rule-based transition system to create an authentication system for the Cloud before translating the complete Cloud model. This authentication system is shown in Figure 4.3.

In Figure 4.3, there are three places (`Log_Reqs`, `Usr_Accns` and `On_Usrs`) and two transitions (AUTH_F, AUTH_S). The transition AUTH_F represents failed authentication due to invalid credentials, while AUTH_S depicts a successful authentication. The firing of these transitions follow rules described in Equations (4.1) and (4.2) while, the description, mapping function and data type of the places are shown in Table 4.1. For instance, the type of the place `Log_Reqs` is $(Str \times Str)$ (product of *string* and *string*) to contain usernames and passwords respectively.

The transition AUTH_S in Figure 4.3 is fired if the necessary preconditions are fulfilled, i.e., the username and password provided by the user match the username and password stored at the user accounts and the user is not already online. These preconditions are represented on the arcs using: (i) the set of users U attempting to log in, where $\forall u \in U : u = (u.\text{username}, u.\text{password})$ represents the username u.username and password u.password provided by a user, (ii) the set C of credentials known to the server, where $\forall c \in C : c =$

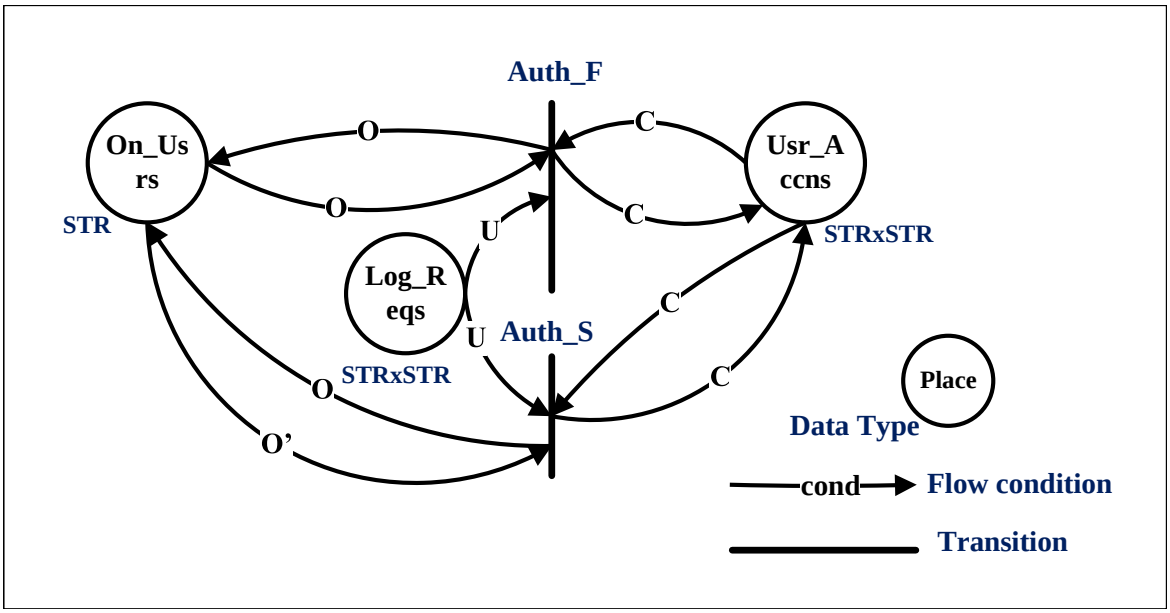

Fig. 4.3 Login system using HLPN

Table 4.1 Description and data type of places in Figure 4.3

Place	Description	Domain	Types
`Log_Reqs`	Login credentials.	$\mathbb{P}(Usernames \times Passwords)$	$Str \times Str$
`Usr_Accns`	Sever-side credentials.	$\mathbb{P}(Usernames \times Passwords)$	$Str \times Str$
`On_Usrs`	Online Users.	$\mathbb{P}(Usernames)$	Str

$(c.\text{username}, c.\text{password})$ represents the username $c.\text{username}$ and password $c.\text{password}$ known by the server, and (iii) set O represents the usernames that are already online. A successful authentication of the user transfers them to the list of online users by adding the new user to the set O which is denoted by O'. On the other hand, a violation in any of the conditions results in the firing of the transition AUTH_F instead. The predicate $R(\mathrm{T})$ denotes if a specific transition T is taken. The governing rules for both AUTH_S and AUTH_F are given in Equation (4.1) and Equation (4.2) respectively.

The implementation of these predicates is shown in Listing 4.1 which was performed using CPN tools [114]. Each of the following Petri net models was implemented using CPN tools to validate the functional behavior of the model. For instance, in the case of the login system, the validation of the model requires that a user with valid credentials should always

get authenticated and later move to `On_Usrs`.

$$R(\text{AUTH\_S}) = \exists u \in \mathsf{U} : u \in \mathsf{C} \wedge \\ u.\text{username} \notin \mathsf{O} \wedge \\ O' := \mathsf{O} \cup \{u.\text{username}\} \tag{4.1}$$

$$R(\text{AUTH\_F}) = \forall u \in \mathsf{U} : u \notin \mathsf{C} \vee \\ u.\text{username} \in \mathsf{O} \tag{4.2}$$

Listing 4.1 CPN ML implementation of Equation (4.1)

```
1  colset Usernames = string; (* Type of Usernames is string *)
2  colset Passwords = string; (* Type of Passwords is string *)
3  colset UNxPW = record un:Usernames * pw:Passwords;
4  (* Type for multiple fields *)
5  var un:Usernames; (* Variable of type Usernames *)
6  var pw:Passwords; (* Variable of type Passwords *)
7  var U,C:UNxPW; (* Variables of type UNxPW *)
8  Auth_S = [#un(U)<>O andalso #un(U)=#un(C) andalso #pw(U)=#pw(C)]
9  (* Trans. guard*)
10 O' = O^#un(U) (* Username is added to online users *)
11 Auth_F = [#un(U)=O orelse #un(U)=#un(C) orelse #pw(U)=#pw(C)]
12 (* Trans. guard *)
```

Figure 4.4 shows a snippet of the CPN tools interface after completing the declarations and adding the guards to the respective transitions. For instance, the place `ON_Usrs` holds the users that are online and currently it is empty. The `Log_Reqs` currently has a single token with the username *"sm"* and password *"t1"*. The same information is stored at the `Usr_Accns` place. In case a `place` holds more than a single token, the number in the green circle will depict the number of tokens the place holds. The AUTH_S is highlighted to indicate that the transition is enabled. In Petri nets the transitions are enabled after all the input places to the transition have at least one token (information) but the transition is only fired after both the transition guard and the output condition of the transition are satisfied. The firing of the transition takes the respective tokens from the input places and add them to the output places considering the output condition. In the case of AUTH_S, the transition guard is to match credentials, and the output condition is to add the user to the `On_Usrs` place.

It is evident that rules-based information flow is independent of the underlying technology since any appropriate technology could be used to determine the validity of the credentials. The subsequent section expands the authentication system by introducing additional Cloud

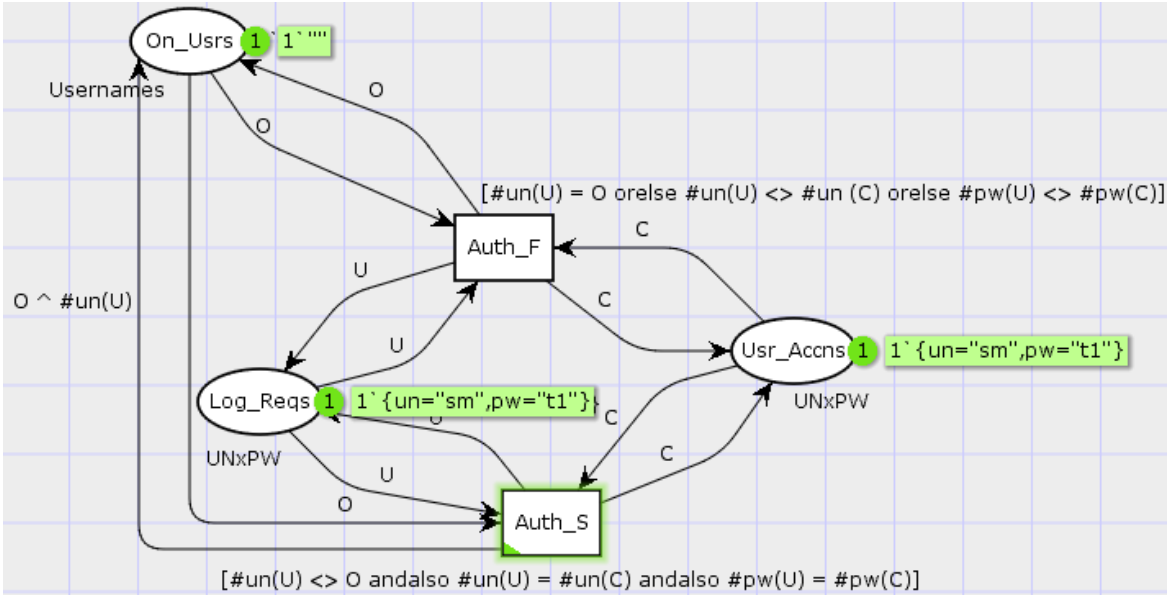

Fig. 4.4 Snippet of CPN tools of the Login system

functionality and eventually representing the Cloud behavior using HLPN. Consequently, the resulting information flow model is agnostic to specific underpinning technologies.

4.2.1 Instantiation of the Cloud Functional Behavior

The previous sections demonstrate the use of HLPN in modeling the login system to validate users. This section extends the authentication system by adding additional services from the Cloud model and eventually, translating the Cloud functional model (cf., Figure 3.1) to an HLPN model which is shown in Figure 4.5. The description of places and their data types are mentioned in Table 4.2. The function *domain*(V) takes an HLPN place V and returns the set of all possible values that V could have.

In Figure 3.2, I presented the sequence diagram of the operations in the Cloud in launching a VM. These operations are revisited from the perspective of creating rules to govern the flow of information among the services and replicating the functional behavior of the Cloud.

1. Transitions T1.1a/T1.1b/T1.2 determines the credentials validity and a successful authentication lead to a dashboard enabling the user to access his/her existing VMs.

2. Transitions T1.3a/T1.3b are triggered after a user initiates the process of the VM creation and provides properties for the VM (e.g., CPU, RAM, disk space). These properties are checked for compliance with the associated quota of the user.

Table 4.2 Description and data type of places in the Cloud Model

Place	Description	Domain	Types
UI	User's interface to enter credentials.	$\mathbb{P}(Usernames \times Passwords)$	$Str \times Str$
AS	Authentication server at the server storing credentials	$\mathbb{P}(Usernames \times Passwords)$	$Str \times Str$
CA	Access restrictions	$\mathbb{P}(Usernames)$	Str
DB	Stored list of VMs	$\mathbb{P}(Usernames \times VMs)$	$Str \times Arr$
INT	Interface to run/initiate VMs	$\mathbb{P}(Username \times CPU \times RAM \times Disk \times Arr)$	$Str \times Str \times Int \times Int \times Arr$
UQ	Users quota and configurations	$\mathbb{P}(Username \times CPU \times RAM \times Disk)$	$Str \times Str \times Int \times Int$
SL	Potential server for the VM request	$\mathbb{P}(Username \times CPU \times RAM \times Disk)$	$Str \times Str \times Int \times Int$
AR	Available resources that can launch the requested VM	$\mathbb{P}(Loc \times DC)$	$Str \times Str$
HS	Receives selected hosting server and VM configurations	$\mathbb{P}(Loc \times DC \times Username \times CPU \times RAM \times Disk)$	$Str \times Str \times Str \times Str \times Int \times Int$
NIC	MAC address and virtual and physical network interface mapping	MAC	Str
NET	Assigns dynamic IP to the instance	$\mathbb{P}(IP \times MAC)$	$Str \times Str$
DI	Holds Disk Image of the VM	$\mathbb{P}(DI)$	Str
HYP	Receives all the configurations and launches the VM	$\mathbb{P}(CPU \times RAM \times Disk \times IP \times MAC \times DI)$	$Str \times Int \times Int \times Str \times Str \times Str$
VM	VM is started on the server	$\mathbb{P}(Loc \times DC \times Username \times CPU \times RAM \times Disk \times DI \times IP \times MAC)$	$Str \times Str \times Str \times Str \times Int \times Int \times Str \times Str \times Str$

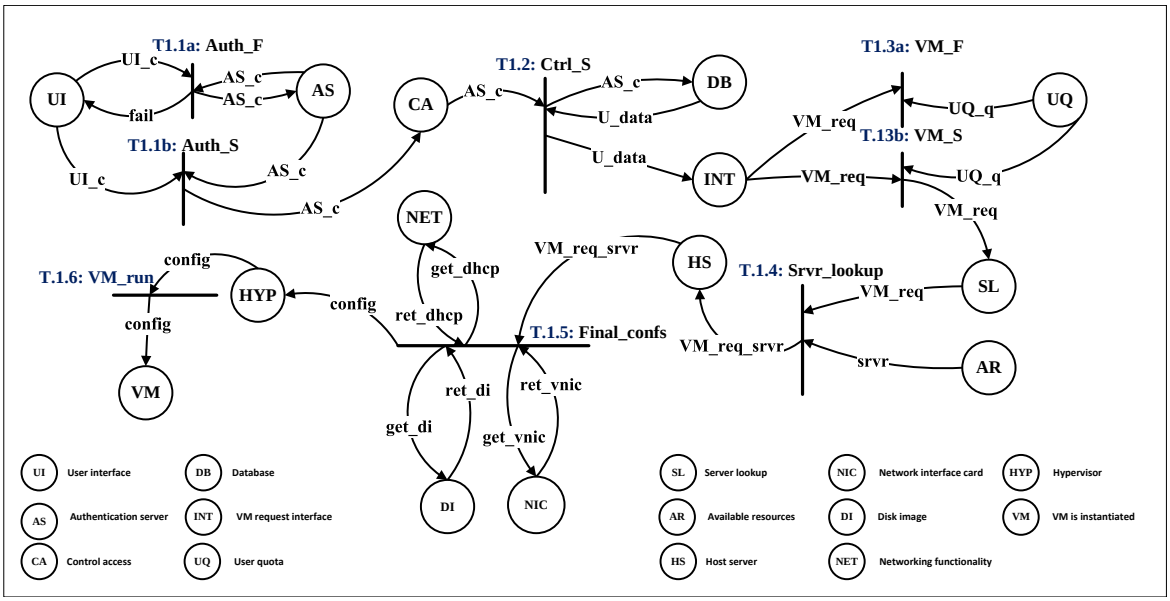

Fig. 4.5 Transforming Cloud Model to HLPN

3. Transition T1.4 is fired after the scheduler service determines a potential data center and a host to run the requested VM.

4. Transition T1.5 is triggered after multiple services provide the respective tokens (information). For instance, a disk image is provided from the repository and the network service initializes a virtual network interface card and assigns MAC/IP addresses. These configurations are pushed onto the hypervisor which configures the VM instance accordingly.

5. Transition T1.6 is fired after it receives the final configuration and the VM has started executing successfully. The `VM` place in Figure 4.5 shows the terminating state of the Cloud model.

The rules that govern the flow of tokens (information) from input places to output places are defined. A new token is generated each time a user tries to login triggering transitions AUTH_F and AUTH_S to determine the validity of the user's credentials. A user can provide multiple credentials and UI_c is the set of provided inputs and AS_c is the set of credentials stored at the server. These credentials are used in Equations (4.3) and (4.4) to check the validity of the user's credentials.

$$R(\text{AUTH\_F}) = \forall u \in \mathsf{UI\_c} : u \notin \mathsf{AS\_c} \tag{4.3}$$

$$R(\text{AUTH\_S}) = \exists u \in \mathsf{UI\_c} : u \in \mathsf{AS\_c} \tag{4.4}$$

Equation (4.3) represents that the credentials provided by the user are invalid, and therefore the user is requested to reenter the valid credentials. On the other hand, the valid credentials trigger AUTH_S transition, and correspondingly, access privileges are granted to the user. The user is transferred to an interface to access the assigned VMs or request new VM instances. Equations (4.5) and (4.6) determine the success or failure of the VM request considering several factors, including the quota associated with the user. The $\mathsf{vm\_req}$ stores the configurations of the requested VM (CPU, RAM, and Disk storage) which are checked for compliance against the allocated quota of the user. The users' quota is stored in UQ and $\mathsf{UQ\_q}$ is the quota of the specified user.

$$\begin{aligned} R(\text{VM\_F}) = \forall d \in \mathsf{VM\_req} : (&d.\text{username} \neq \mathsf{UQ\_q}.\text{username} \vee \\ &d.\text{cpu} \neq \mathsf{UQ\_q}.\text{cpu} \vee \\ &d.\text{ram} \neq \mathsf{UQ\_q}.\text{ram} \vee \\ &d.\text{disk} \neq \mathsf{UQ\_q}.\text{disk}) \end{aligned} \tag{4.5}$$

$$\begin{aligned} R(\text{VM\_S}) = \exists d \in \mathsf{VM\_req} : (&d.\text{username} = \mathsf{UQ\_q}.\text{username} \wedge \\ &d.\text{cpu} = \mathsf{UQ\_q}.\text{cpu} \wedge \\ &d.\text{ram} = \mathsf{UQ\_q}.\text{ram} \wedge \\ &d.\text{disk} = \mathsf{UQ\_q}.\text{disk}) \end{aligned} \tag{4.6}$$

Equation (4.5) determines the invalidity of the VM request due to a lack of access privileges for additional VM or if the configurations of the requested VM do not comply with the associated quota. The compliance of the requested VM invokes the scheduler service that selects an appropriate server to instantiate the requested VM. Furthermore, the selection of the server triggers multiple services to configure the VM. For instance, the disk image service provides a guest operating system for the VM. The network service provides networking capabilities to the VM, i.e., initiating a virtual network interface card, assigning a MAC address, and determining the mapping between the virtual and the physical interfaces of the machine. NET is responsible for leasing IP addresses and the corresponding IP address mapping to the MAC address. These configurations are pushed onto the hypervisor, which executes the VM on the physical hardware. These configurations follow Equation (4.7) for triggering the

respective transition. In Equation (4.7), $++$ is used to denote tuple concatenation and $:=$ to denote assignment resulting in a variable being updated.

$$
\begin{aligned}
R(\text{FINAL\_CONFS}) = \exists \mathsf{im} \in \mathit{domain}(\mathtt{DI}) : \mathsf{im} = \mathsf{ret\_di} \wedge \\
\exists \mathsf{vn} \in \mathit{domain}(\mathtt{NIC}) : \mathsf{vn} = \mathsf{ret\_vnic} \wedge \\
\exists \mathsf{dh} \in \mathit{domain}(\mathtt{NET}) : \mathsf{dh} = \mathsf{ret\_dhcp} \wedge \mathsf{dh.mac} = \mathsf{vn.mac} \wedge \\
\mathsf{config} := \mathsf{VM\_req\_srvr} ++ (\mathsf{im}) ++ \mathsf{dh}
\end{aligned}
\tag{4.7}
$$

The implementation of Equation (4.7) in CPN tools is shown in Listing 4.2 and the respective snippet of the transitions and places in CPN tools is shown in Figure 4.6. As Figure 4.6 shows, the place SL receives VM configurations and the server look-up is initiated. After the selection of the server for the VM, both the VM and server information are passed onto HS which temporarily holds this information. The variable VM_req_srvr is of type *VMCONF* (line 8 and 9 in Listing 4.2), i.e., the variable is able to hold both the VM configurations and the server information. This is passed to FINAL_CONFS transition. Moreover, at this transition, DI provides an operating system for the VM, NET provides IP and MAC addresses and NIC provides a MAC address and a mapping between the physical virtual interfaces. This is achieved by the transition that is fired after comparing vnic with ret_dhcp. Following this, the final configurations (VM_req_srvr, ret_di, ret_dhcp) are passed onto the hypervisor which runs the VM as per the received configurations.

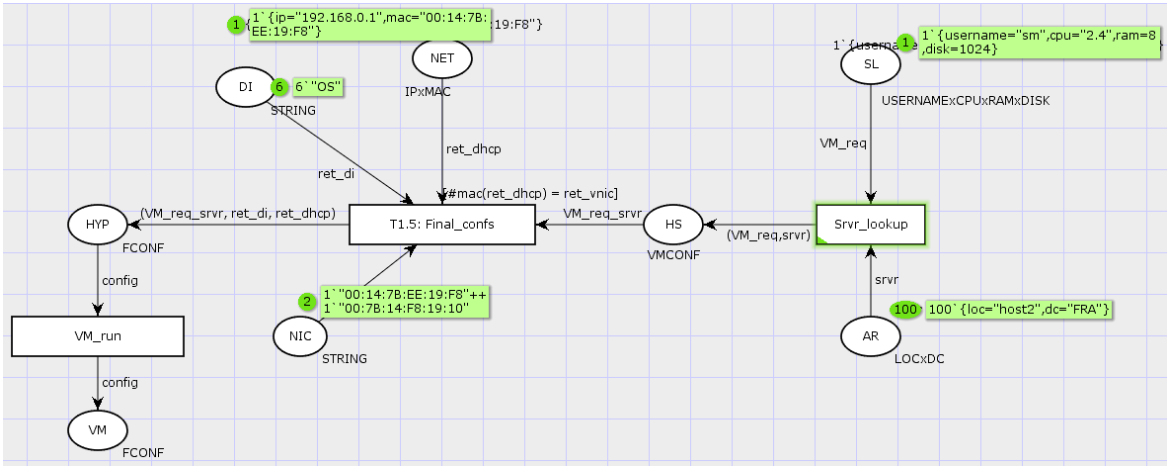

Fig. 4.6 Snippet of CPN tools of the Final Configurations

This section explained the functional behavior of the Cloud as a rule-based transition system irrespective of the underlying technologies. The rules determine the information flow among the services for the proper functioning of the Cloud. On the other hand, a threat's

Listing 4.2 CPN ML implementation of Equation (4.7)

```
colset CPU = string; (* Type of CPU is string *)
colset RAM = int; (* Type of RAM is int *)
colset DISK = int; (* Type of RAM is int *)
colset USERNAMExCPUxRAMxDISK =
record un:USERNAME * cpu:CPU * ram:RAM * disk:DISK
var VM_req:USERNAMExCPUxRAMxDISK;
(* Variable of type USERNAMExCPUxRAMxDISK *)
colset LOCxDC= record loc:LOC * dc:DC; (* Type of multiple fields *)
var srvr:LOCxDC; (* Type of LOCxDC *)
colset VMCONF = product USERNAMExCPUxRAMxDISK * LOCxDC
(* Immutable fields *)
var VM_req_srvr:VMCONF; (* Variable of type VMCONF *)
colset IP = string; (* Type of IP is string *)
colset MAC= string; (* Type of MAC is string *)
colset IPxMAC= record ip:IP * mac:MAC; (* Type of multiple fields *)
var ret_dhcp:IPxMAC; (* Variable of type IPxMAC *)
colset DI = string; (* Type of DI is strin *)
var get_di:DI; (* Variable of type DI *)
colset FCONF = product VMCONF * DI * IPxMAC;
var config:FCONF;
Final_confs = [#mac(ret_dhcp) = ret_vnic] (* Trans. guard*)
```

input can alter the behavior of the Cloud leading to malfunctioning. Thus, the following section defines the behavior and characteristics (e.g., preconditions, consequence, etc.) of a threat that are given as the spurious input to the Cloud to analyze its impact on the functional behavior of the Cloud.

4.3 Instantiation of a Threat's Behavior

The previous sections described the normal functional behavior of the Cloud similar to the basic transition system (cf., Figure 4.1 in Section 4.1) in a technology-agnostic manner. As previously described, in Figure 4.1, the paths to the terminal states are modified by additional inputs. Thus, this section presents threats as additional inputs to the Cloud. Similar to the Cloud services, a threat's behavior is defined by representing the necessary conditions required for a threat to exploit a service. Moreover, modeling the behavior facilitates in assessing the impact of a threat on a particular service and consequently tracking its progression across the system. The threats are given as input to the Cloud model, and the consequence of the threat dictates the next place/state in the Cloud model. Furthermore, in combination with the CPN tools [115], the HLPN can be simulated to enumerate benign behavior to validate the functionality of the Cloud and conversely investigate the attack paths

generated due to the threat. The instantiation of a threat using HLPN is shown in Figure 4.7 and Table 4.3 describes the places used in the HLPN model along with their description and data types. The significance and utilization of these places in defining the threat behavior are explained in the following.

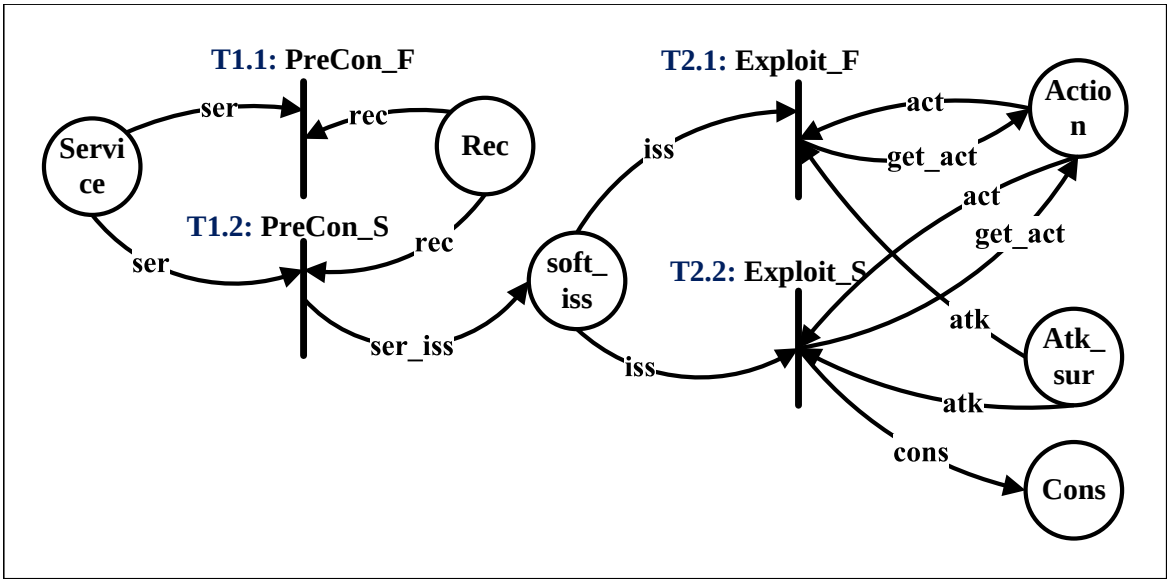

Fig. 4.7 Modeling a threat's behavior using HLPN

Table 4.3 Description and data type of places in Figure 4.7

Place	Description	Mapping	Types
`Service`	Targeted services.	$\mathbb{P}(Services \times Issues)$	$Str \times Str$
`Rec`	Reconnaissance step input.	$\mathbb{P}(Services \times Issues)$	$Str \times Str$
`soft_iss`	Potential issues in the target.	$\mathbb{P}(Services \times Issues)$	$Str \times Str$
`Action`	Action to exploit the issues.	$\mathbb{P}(Action)$	Str
`Atk_sur`	Attack surface.	$\mathbb{P}(Atk\_sur)$	Str
`Cons`	The consequence of the threat.	$\mathbb{P}(Cons)$	Str

4.3.1 Reconnaissance Step

This step uncovers potential weaknesses in a system that could be exploited by an attacker. For example, the installation of a vulnerable version of the software or a misconfigured service could be a potential weakness. Additionally, this step explores the necessary preconditions to

exploit the weakness. The reconnaissance step can be done using different tools but for the purpose of profiling threats and investigating their progression, data published in the national vulnerability database [30] suffices since the purpose is to collect weaknesses in the services and use them as a triggering condition of a transition. Consequently, this enables tracking the progression of the threat in the system. Equations (4.8) and (4.9) determines if the necessary preconditions of the potential weakness are fulfilled.

$$R(\text{PRECON\_S}) = \exists r \in \mathit{domain}(\mathsf{Rec}) : r \in \mathsf{ser} \tag{4.8}$$

$$R(\text{PRECON\_F}) = \forall r \in \mathit{domain}(\mathsf{Rec}) : r \notin \mathsf{ser} \tag{4.9}$$

Equation (4.8) demonstrates the fulfillment of preconditions, i.e., there exists a service with a potential issue discovered during the reconnaissance step. The absence of such an exploitable weakness instead fires PRECON_F as determined by Equation (4.9).

4.3.2 Exploit Step

This step is triggered if a service has an existing issue that could be exploited. This requires an attacker to utilize an action designed to exploit the specific weakness. An absence of such an action indicates an open window of compromise. The rules governing the exploit step are described in Equations (4.10) and (4.11).

$$\begin{aligned} R(\text{EXPLOIT\_S}) = {} & \exists i \in \mathit{domain}(\texttt{soft\_iss}) : i = \mathsf{iss} \wedge \\ & \exists a \in \mathit{domain}(\texttt{Action}) : (a = \mathsf{act} \wedge a = \mathsf{iss.issue}) \wedge \\ & \exists as \in \mathit{domain}(\texttt{Atk\_sur}) : as = a \end{aligned} \tag{4.10}$$

$$\begin{aligned} R(\text{EXPLOIT\_F}) = {} & \forall i \in \mathit{domain}(\texttt{soft\_iss}) : i \neq \mathsf{iss} \vee \\ & \nexists a \in \mathit{domain}(\mathsf{Action}) : (a = \mathsf{act} \vee a = \mathsf{iss.issue}) \vee \\ & \nexists as \in \mathit{domain}(\texttt{Atk\_sur}) : as = a \end{aligned} \tag{4.11}$$

A successful exploit might affect the normal operations of a system. For instance, a Denial of Service (DoS) would limit the availability of the service. These consequences are represented as the `Cons` in Figure 4.7. On the other hand, if the consequence of the threat is to bypass authentication then the consequence of the threat is the next available place for the attacker after circumventing the authentication service.

The implementation of the threat's instantiation in the CPN tools is given in Listing 4.3 and the respective snippet from the CPN tools is shown in Figure 4.8. For simplicity, the suc-

Listing 4.3 CPN ML implementation of Equation (4.8) and Equation (4.10)

```
1  colset SERVICE = string; (* Type of service is string *)
2  colset ISSUE = string; (* Type of ISSUE is string *)
3  colset SERxISS = record s:SERVICE * i:ISSUE;
4  var ser, rc, iss:SERxISS; (* Variable of type SERxISS *)
5  var act, atk:STRING;
6  PreCon_S = [#s(ser) = #s(rc) andalso #i(ser) = #i(rc)]
7  (* Trans. guard*)
8  Exploit_S = if #i(iss) = act
9                then 1`"bypass"
10             else emtpy (* Trans. guard and output condition merged *)
```

cess cases of the rules are shown, i.e., implementation of Equation (4.8) and Equation (4.10). As can be seen from Figure 4.8, the service *Auth* is vulnerable to token mismanagement and is discovered during the reconnaissance phase. Following this discovery, the respective action is taken from the `Action` place. The `Action` holds actions that attackers can take. For instance, an issue can be exploited by different actions, a single action can impact multiple issues, or a single issue belonging to various services exists that a single action can target. Therefore, actions are not tied to specific services or issues. A successful exploit leads to the `Cons` place that holds the impact of the exploit.

4.4 Connecting the Cloud Model and Threats

The previous sections have described the information flow model and the instantiation of the threat behavior using Petri nets and their implementation in CPN tools. However, the connection between the Cloud model and the threats still remains. This is shown in Figure 4.9, where after successfully bypassing the authentication server (`AS`), the next place available to the attacker is the `VMReq`. `VMReq` is the same as `INT` in Figure 4.5.[2] As can be seen from Figure 4.9, the data type of the `AS` is `UNxPW`, indicating that this place can hold username and password. Moreover, at the top of the `AS`, actual values of usernames and passwords are highlighted. These values are used against the stored values at the server side to grant/reject authentication. Currently, there are two users with usernames "sm" and username "ai" and are differentiated with ++ between their values. The *1* at the beginning highlights that the user is still waiting for authentication, once the values are passed onto the transition this counter is reduced to *0* to indicate that the user has attempted to log in. This limits the attempts from the user to get authenticated.

[2] `INT` is a reserved keyword in CPN tools and hence cannot be used as name for a place.

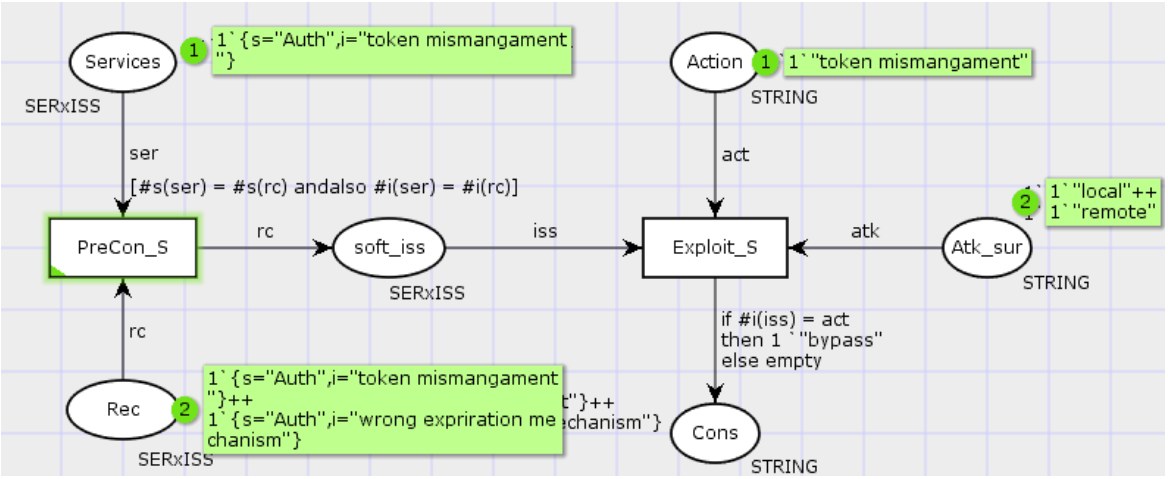

Fig. 4.8 Snippet of CPN tools depicting threats behavior

On the other hand, both the *Threats* and the *Cloud Model* in Figure 4.9 indicate that the functionality is hidden. These are termed as hierarchical Petri nets and the aim of the hierarchy Petri nets is to hide the places and transitions of the individual blocks while only showing the connection among the blocks. For instance, the *Threats* block encompasses the places and transitions represented in Figure 4.7. The `VM` (denoted with a red circle) is the terminating state of the model.

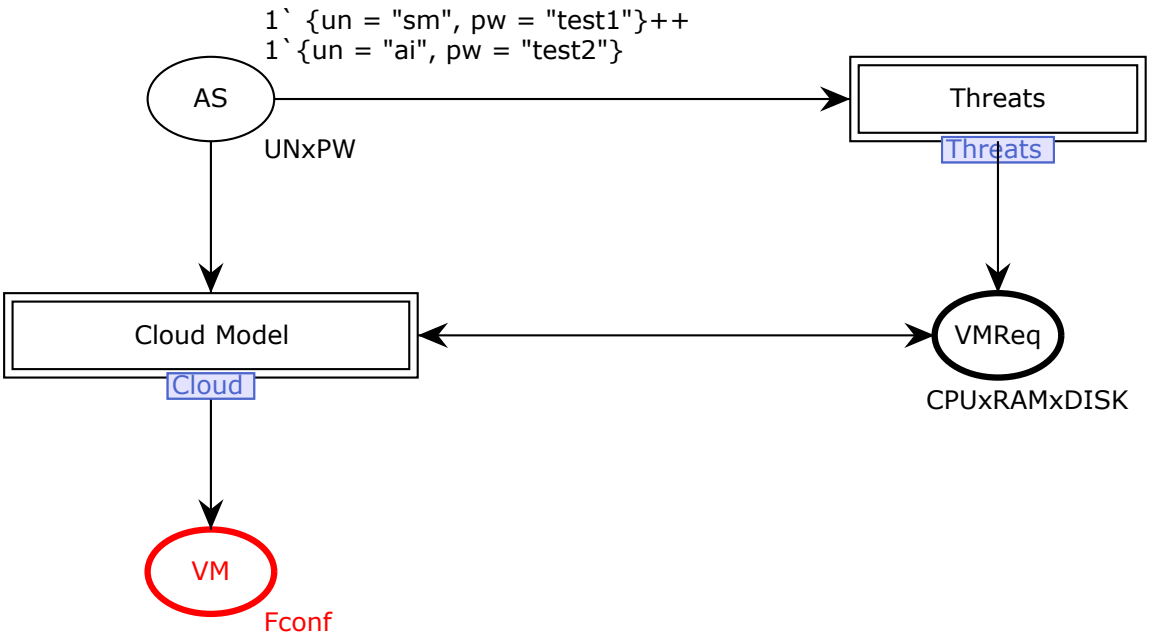

Fig. 4.9 Link between threats and the Cloud Model

4.5 Conclusion

This Chapter has investigated the first research question, how Cloud Service Providers (CSPs) can examine the Cloud's security at different abstractions of the operations. To this end, this Chapter presented the first contribution of the thesis, i.e., modeling the Cloud operations to an information flow model capturing the services interactions to represent the behavior without being specific to technologies used in the Cloud. Moreover, this Chapter illustrates the use of threats as spurious input to varied services across the operational stack to determine their impact on the Cloud. The information flow model and the threat behavior are defined using HLPN, enabling me to assign multiple specifications to each service. For instance, the service's regular interaction with other services is one specification, while the second specification could be a threat targeting the service. These specifications are achieved by defining rules and constraints that determine the triggering of transitions after their respective preconditions have been fulfilled. Moreover, the states in Petri nets are distributed, i.e., the threat's impact could be determined locally, which enables tracking the threat from the targeted service instead of the global starting point. This essentially allows CSPs to examine the security of the Cloud at different abstractions of the operations.

Chapter 5

Threat Analysis

This Chapter addresses the second research question: How can the effects of a threat in a service on other interconnected services be identified? To this end, this Chapter contributes by proposing path-illustrative threat analysis techniques, ThreatPro and AttackDive, that can reveal the progression of a threat in the system. These approaches support profiling threats, analyzing their impact on targeted services, and the propagation of threats across the multiple layers of the Cloud. ThreaPro assesses the impact of threats to Cloud services at different levels of abstraction, e.g., considering threats at multiple services/layers and the possibility of a threat's combination to violate a security requirement of the user. Furthermore, the progression of a threat in the Cloud's dynamic environment, where resources migrate from one physical host to another that may have different security properties, is investigated in the Chapter. Additionally, ThreatPro provides the capability to perform a speculative analysis to examine the potential of a threat to compromise a security requirement. On the other hand, AttackDive investigates the operational visibility of the Cloud and explores its exploitability corresponding to different attacker profiles, e.g., insider vs. outsider attackers.

Both ThreatPro and AttackDive approaches utilize the functional Cloud and the information flow model which are described in Chapter 3 and Chapter 4 respectively. Briefly, the Cloud model is an abstraction of services from real-world deployments, while the information flow model governs the flow of information among the services using transitions that are triggered after their respective conditions are satisfied. Moreover, Chapter 4 comprehensively detailed the threats and their required preconditions in the form of constraints to transitions, so this Chapter builds on the previous Chapters to perform threat analysis in the Cloud.

However, before proceeding to threat analysis, The correct behavior of the Cloud is validated to establish a baseline for the operations in launching a VM. Specifically, I examine if the Cloud always terminates to the `VM` state each time a user requests a new VM or starts an existing VM. Consequently, allowing me to enumerate all the execution paths that lead to the

correct terminal state. The terminal state is VM for both starting an existing VM or launching a new instance of the VM. Thereafter, additional constraints are inserted that act as threats to different services in order to investigate paths leading to violations of security requirements.

By using an HLPN to build the information flow model it means CPN tools [115] can be used to simulate the model and enumerate the Cloud behavior. The simulation allows for the analysis of Cloud's behavior when no adversary is present, i.e., given a valid VM request the terminating state should always be the VM state. CPN tools also support triggering transitions at certain time intervals which facilitates modeling dynamic Cloud behavior. This is accomplished by triggering new events (e.g., launching a new VM, migrating a VM, or fulfillment of a threat's preconditions) after a certain time period has elapsed in the simulation. This establishes the handling of the dynamic behavior of the Cloud by discerning the impact of the new events in the model.

In the following sections, CPN tools is utilized to generate states enumerating the Cloud's benign behavior and also the possible deviation in its behavior after inserting threats to different services in order to perform threat analysis. Following this overview, the subsequent sections detail each enumerating the Cloud behavior to establish a baseline behavior. Thereafter, threats are added and their propagation is ascertained in the Cloud.

5.1 Enumerating the Cloud behavior

Initially, the behavior of the Cloud without the threats must be validated to understand the operations of the Cloud in their absence. This is achieved through simulating the Cloud model based on the HLPN (cf., Figure 4.5) using CPN tools. The functional property is validated initially, i.e., given a request by a user, the Cloud should always launch the VM, and hence the terminating state should always be the VM state. This also enables validating the correct functional behavior before the threats are added to the Cloud. The sequence of states is generated using CPN tools for the scenario where a valid user requests a VM. In this valid request, the execution will always terminate at the VM state. An illustration of a subset of valid paths is shown in Figure 5.1 where those paths all terminate at the VM state. Some paths show VM+Data instead of VM to represent the scenario in which a user had requested storage capacity along with a VM. This is simply used to differentiate between VMs with and without storage. These paths correspond to the instantiation of the Cloud behavior presented in Section 4.2.1. Enumerating the Cloud behavior establishes a baseline of operations without the presence of a threat and demonstrates the proper functioning of the Cloud.

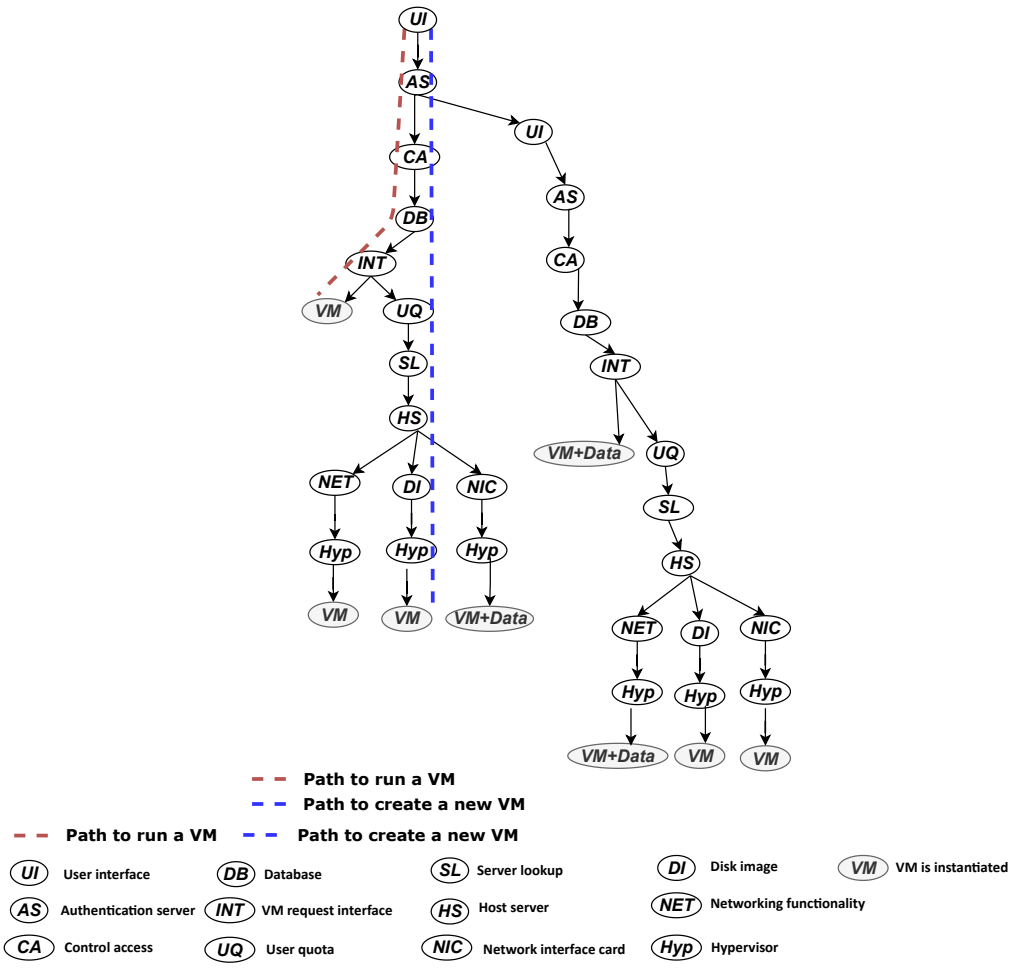

Fig. 5.1 Example of valid execution paths in the Cloud environment

5.2 ThreatPro: A Multi-layer Dynamic Threat Analysis

This section performs the threat analysis by adding constraints (e.g., threat conditions at different services) to the HLPN and simulating the Cloud behavior in the presence of these threats. The threats are added at different layers/services to investigate both the cause-effect relationship and to analyze their impact on the Cloud's functional behavior.

To demonstrate the generalization of the presented approach, I first perform speculative analysis using vulnerabilities reported in the national vulnerability database [30] to identify corresponding attack scenarios. The objective of this analysis is to identify potential paths that could be used by an attacker to undermine a security requirement.

The set of the vulnerabilities presented in Table 5.1 to demonstrate the effectiveness of ThreatPro in analyzing the potential impact of threats at different layers of the Cloud and the potential of a threat to progress in the Cloud. The first column in the table is the CVE

Table 5.1 List of vulnerabilities from NVD with CIA consequences indicated

CVE#	Service	HLPN Place	C	I	A
CVE-2012-4457	Authentication	AS	✓		
CVE-2013-2006	Authentication	AS	✓		
CVE-2013-4222	Authentication	AS	✓		
CVE-2013-7130	Compute	HYP	✓		
CVE-2014-0134	Compute	HYP		✓	
CVE-2014-2573	Neutron	NET	P		
CVE-2014-9623	Glance	DI		P	
CVE-2015-2687	Compute	HYP	✓		
CVE-2016-5362	Neutron	NET	✓		
CVE-2016-0757	Cinder	SL	✓		
CVE-2018-14432	Cinder	CA	✓		
CVE-2018-14635	Neutron	NET	P		

entry, while the second and third columns show the targeted service and its corresponding HLPN place. The last three columns show the vulnerability's consequence on Confidentiality, Integrity, and Availability (CIA). A partial impact is indicated with P and a full impact with ✓. Where a partial impact means that a subset of data was revealed to an adversary (confidentially) or a subset of data was corrupted (integrity). The attack graph generated from these vulnerabilities is shown in Figure 5.2. The multiple paths violating security requirements are explained below, where each path enumerates a single attack.

Path 1: A successful exploitation of vulnerabilities in path 1 of Figure 5.2 leads to attaining additional resources in the Cloud from a disabled user. It is accomplished by exploiting vulnerability CVE-2013-4222/CVE-2012-4457 to request a new authorization token of the disabled user and utilize this token in accessing the victim's resources. A precondition of the attack requires authentication of the user which is achieved by exploiting vulnerabilities CVE-2013-2006, CVE-2015-3646

Path 2: Exploiting CVE-2014-5251 at the control service allows attackers to bypass access restrictions and potentially discover restricted projects. However, in combination with CVE-2018-14432, an attacker can escalate the impact to retain access to these restricted projects with an expired authorization token. Alternatively, an attacker in combination with vulnerability CVE-2016-0757 at SL service might be able to change the VM's configuration.

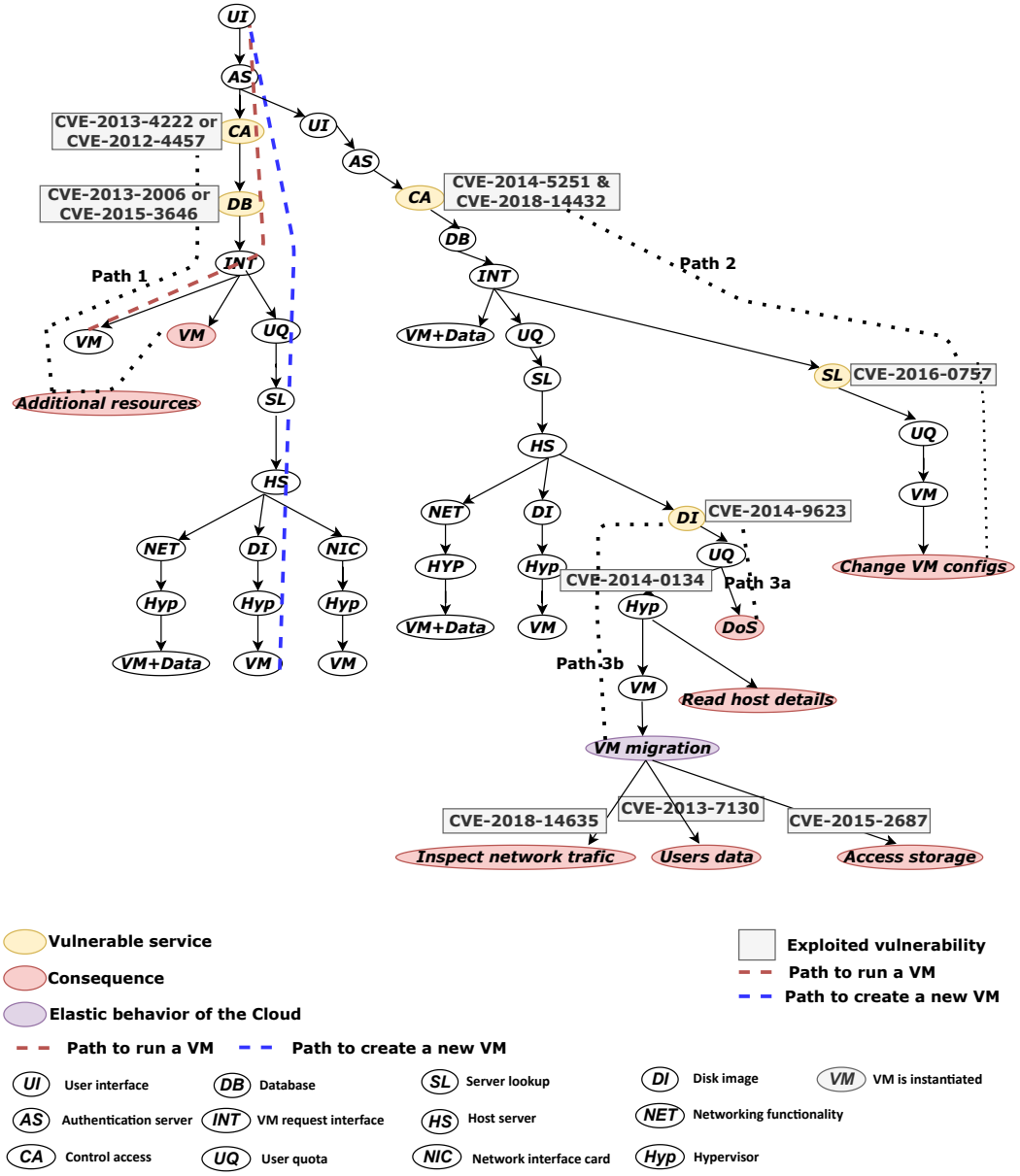

Fig. 5.2 Attack paths based on the selected vulnerabilities

This path specifically shows that combining vulnerabilities from different services can increase the overall impact and therefore, the potential of a threat's progression should be considered in the threat analysis process.

Path 3: Similar to Path 2, this path has multiple potential consequences depending on the combination of the exploited vulnerabilities. In path 3a, the vulnerability CVE-2014-9623 at the disk image service is exploited to bypass the storage quota and thus enable the attackers to upload a large image file causing a denial of service. However, path 3b illustrates alternative paths in which the vulnerability is combined with a hypervisor vulnerability (CVE-2014-0134), resulting in either reading the configuration file of the physical server, breaching the confidentiality, or potentially causing the VM to migrate. The latter case opens up new attack surfaces such as when exploiting CVE-2018-04635 during VM migration which could allow attackers to intercept network traffic. Alternatively, the vulnerability CVE-2013-7130 facilitates attackers to access other users' data.

These attack surfaces are introduced due to the elastic behavior of the Cloud. Since this analysis happens at run-time the ThreatPro methodology is able to identify these attack paths. Other threat analysis tools that only consider a static view of the system would only be able to incorporate the changes in the system after they are executed again. These tools might require a large number of re-executions in order to process all the changes that elastic Cloud behavior may introduce.

Speculative Analysis

The speculative analysis allows the exploration of the potential paths an attacker could use to accomplish their objectives. Moreover, the speculative analysis facilitates a proactive approach to threat mitigation and prioritization of threats according to their impact or the threat's degree of centrality in the path. In the following section, a post-mortem analysis of two cases is performed to illustrate the attack paths that could violate different security requirements. This demonstrates the effectiveness of ThreatPro in identifying threat progression in the system as well as disclosing alternative attack paths through speculative analysis.

5.2.1 Validation: Real-world Case Studies

The previous sections outlined the processes of ThreatPro in conducting actual and speculative threat analysis to identify attack paths. To validate ThreatPro, in this section, multiple CVEs are used that are related to real-world attacks to enumerate the attack paths attackers used to compromise the system. In addition, ThreatPro is able to conduct a post-mortem analysis of

these attacks by introducing speculative conditions and exhibiting alternative potential cases of violation of the security requirements. In essence, these potential attack paths determined through speculative analysis highlight ThreatPro's predictive capabilities for identifying alternate possible attacks.

I now present two case studies of actual Cloud attacks to illustrate the process of ThreatPro's methodology. The first attack is the Equifax attack on breach of confidentiality [116] where attackers exfiltrated confidential data of Equifax's customers. The second attack is a resource consumption attack that exhausts the system's resources hindering the availability of the application [20].

Case I: Confidentiality as a Requirement

The first attack scenario covers the violation of a confidentiality requirement. I review the Equifax data breach where attackers successfully ex-filtrated the financial and private records of approximately 148 million users, making it one of the largest data breaches and an attack with one of the largest financial settlements [117]. Furthermore, this case specifically highlights the significance of multi-layer attacks where supposedly negligible issues at different layers were combined to create an aggregated impact. Although threat analysis techniques are useful to determine these issues individually at each service, ThreatPro provides the capability of assessing the impact of the threats and their possible combination in the system. This is achieved through modeling the functional behavior to determine a threat's possible progression in the system. A brief analysis of the attack is presented in the following illustrating the path taken by attackers to access the confidential data of the users. The readers are referred to [116] for a complete analysis of the data breach.

1. Attackers exploited a vulnerability in the web portal granting them access to the web server.

2. User names and passwords were saved in plain text facilitating attackers to penetrate further into the system using these credentials.

3. Networks and systems were not segmented properly allowing attackers to move laterally across the network and systems without any restriction.

This attack is an example of attackers moving across the services/layers and eventually reaching restricted states of the system due to the presence of negligible issues at each service/layer. For instance, the proper partitioning of the network/systems would have limited the impact of the attack as well as encrypting the credentials at rest. However, the combination of these negligible issues across different services/layers amplified the impact

of the attack. Using ThreatPro, the sequence of steps is generated that enable attackers to access the data which are shown in Figure 5.3.

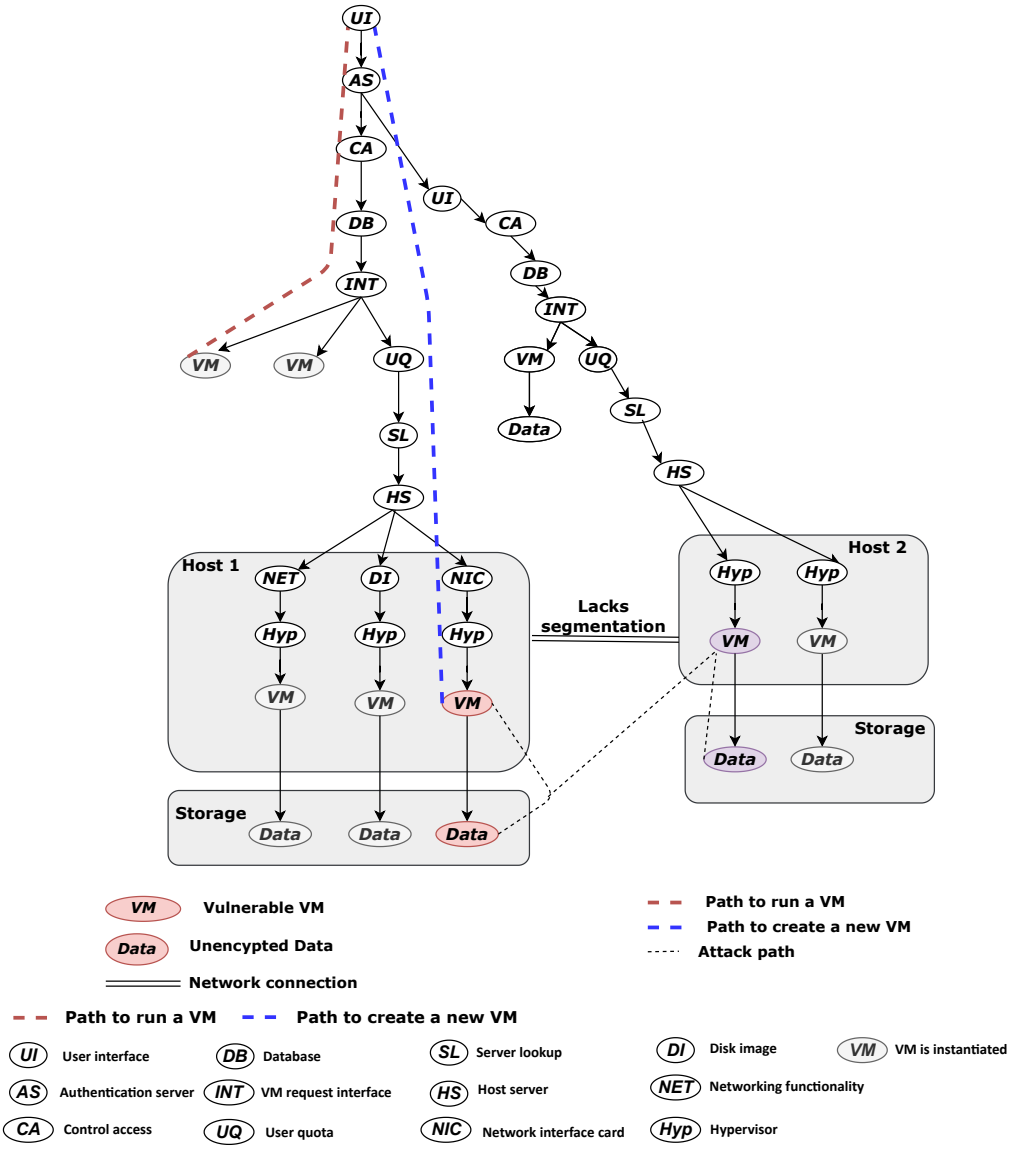

Fig. 5.3 Attack path in the Equifax data breach

Figure 5.3 shows the attacker compromised the web server running on the VM at host 1 by exploiting the publicly known vulnerability CVE-2017-5638. This allowed attackers to gain access to the VM resources and the storage of the unencrypted credentials which facilitated attackers to penetrate further into the system by using these credentials. On the other hand, systems/networks were not properly segmented allowing attackers to use the credentials on VMs running at different hosts, e.g., host 2 in Figure 5.3. In the following

section, the capability of ThreatPro in revealing alternative attack paths at the attacker's disposal is demonstrated by performing speculative analysis.

Speculative Analysis

Figure 5.3 shows the potential issues that were exploited by the attacker, however, the speculative analysis of the Equifax data breach reveals that the attackers have alternative attack paths at their disposal to accomplish their goals. For instance, if the network is partitioned properly, an alternative route for the attacker could be to intercept network traffic by exploiting vulnerability CVE-2016-5363/CVE-2016-5362 at the network service. Thus, speculative analysis is useful to determine the alternative paths exploitable by an attacker in case a mitigation strategy is deployed.

5.2.2 Case II: Availability as a requirement

The second attack illustrates the use of ThreatPro in determining the paths violating the availability requirements of an application. Specifically, this attack entails exhausting the resources to limit the availability of an application and eventually causing a denial of service. These attacks typically target content delivery applications where timely delivery of content is the primary objective [118, 119]. Recently, Amazon reported that it has thwarted the biggest attack on its services [20]. The documented information is limited in these cases to avoid leakage of propriety information that could potentially be used in future attacks. However, using the threats published in the NVD, ThreatPro is able to depict scenarios where an attacker can target individual services or discover a combination of vulnerabilities to cause exhaustion of the resources. These attack paths are shown in Figure 5.4 and are explained below.

Paths 1 and 2

Using CVE-2016-5362 or CVE-2016-5363 at the network service, an attacker can intercept the traffic and cause a resource consumption attack. This vulnerability allows the interception of traffic destined for other hosts and thus, could potentially be used to intercept snapshots of the VM during the migration process and consequently enable attackers to exhaust resources. On the other hand, in path 2, a vulnerability (CVE-2014-9623) exploited at the disk image service combined with a vulnerability at the hypervisor (CVE-2014-2573) leads to a resource consumption attack instead. Furthermore, exploiting either CVE-2017-17051 or CVE-2015-3241 at the hypervisor also leads to exhausting resources by repeatedly rebuilding instances with new disk images.

Using CVE-2016-5362 or CVE-2016-5363 at the network service, an attacker can intercept the traffic and cause a resource consumption attack. This vulnerability allows the interception of traffic destined for other hosts and thus, could potentially be used to intercept snapshots of the VM during the migration process and consequently enable attackers to exhaust resources. On the other hand, in path 2, a vulnerability (CVE-2014-9623) exploited at the disk image service combined with a vulnerability at the hypervisor (CVE-2014-2573) leads to a resource consumption attack instead. Furthermore, exploiting either CVE-2017-17051 or CVE-2015-3241 at the hypervisor also leads to exhausting resources by repeatedly rebuilding instances with new disk images.

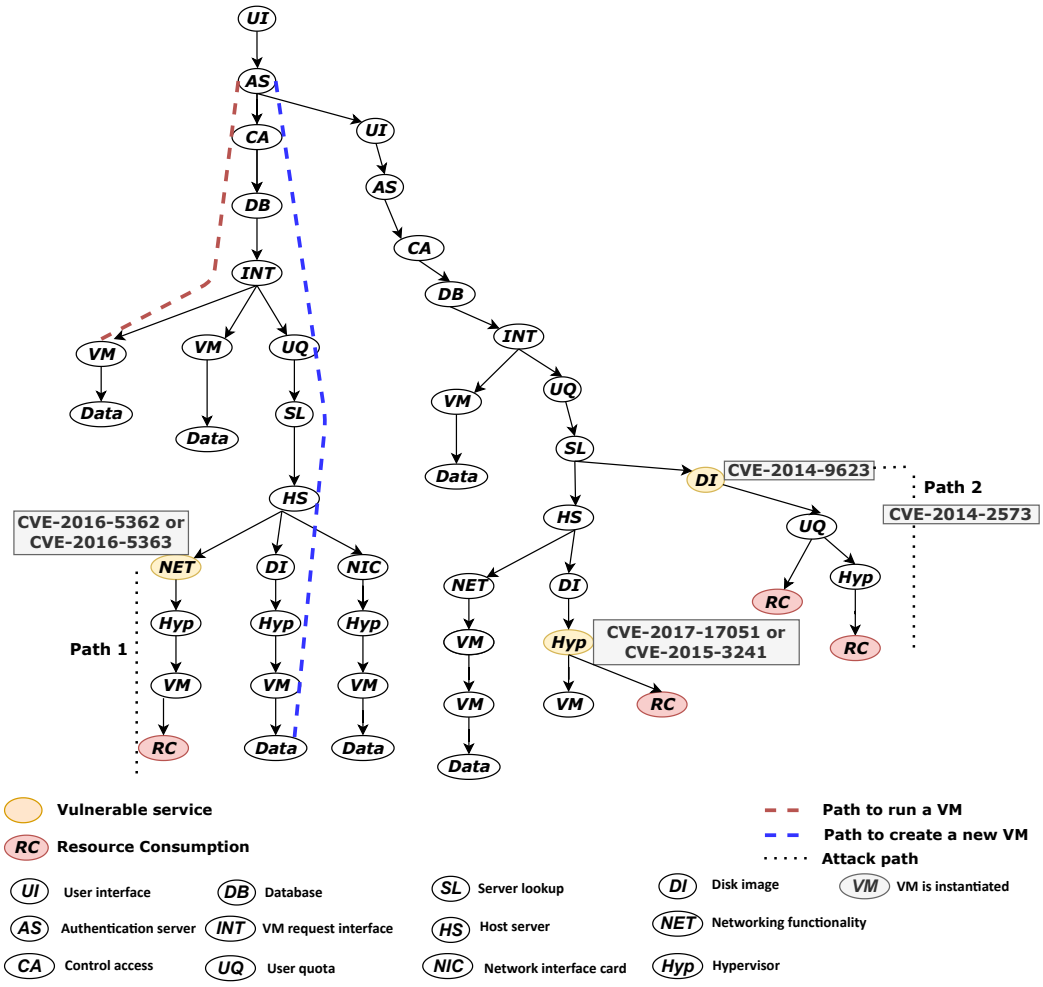

Fig. 5.4 Attack path in a resource consumption attack

Speculative analysis

Performing speculative analysis reveals alternative paths that might result in exhausting a resource. For example, the vulnerabilities CVE-2017-17051 and CVE-2015-3241 can be used to exploit the functionality of a hypervisor to exhaust resources by repeatedly building the same instance. This causes double allocations and repeating the process causes the denial of service as the resources get exhausted.

These attack scenarios illustrate that a proactive approach is required to analyze the progression of a threat in the Cloud to explore possible attack paths that can be exploited by the attackers. ThreatPro can be used to perform speculative cause-effect analysis to determine the impact of a threat at a single service as well as analyze the impact of combined threats on the violation of a security requirement.

5.3 AttackDive: Exploring Attack Surfaces

The technique AttackDive investigates the visibility of services corresponding to varied attacker profiles to explore attack surfaces within their view. This delineation is desired as different attackers can have disparate states visible to them and can have a different impact on the Cloud's operational behavior. Therefore, the criticality of services for different attacker profiles is segregated, as the set of operations insider attackers can perform on services might be different due to their access privileges. Thus, the following sections present attack surfaces visible to different attacker profiles.

5.3.1 Insider Attacker vs. Outsider Attackers

An insider attack is orchestrated by entities responsible for managing the Cloud operations. They usually have elevated access to the services and also possess intimate knowledge of the infrastructure. This makes an insider attack have a high potential for damage. Thus, an analysis of possible attack paths from an insider perspective is required to identify malicious sequences of operations corresponding to them.

An analysis of the behavioral interactions of multiple services that are only accessible to an insider is presented. Consider UQ (cf., Figure 5.1) service, which is responsible for holding quota configurations and access policy of the customer. If the admin changes the quota associated with a user, it might lead to the over-provisioning of resources to the user without the user's knowledge. Further, since the Cloud uses a pay-as-you-go model, these additional resources will incur an additional cost that the user would have to pay. The alternate consequence of changing the UQ configuration is changing the host server selection.

A malicious admin can add/delete resources to a user such that eventually, the resource migrates to a server the admin controls. After that, the admin can launch an attack on the resource, e.g., using networking or cache as a side-channel to breach the confidentiality of the user [36, 77] or denial of service by migrating the resource to a server that is over-provisioned and thus, limiting the availability of the resource [78]. Additionally, a malicious admin can do SQL injection attack in which an insider writes malignant entries directly to the database. This would result in skipping a number of transitions and directly pushing the maligned server and VM configurations onto the hypervisor which will then instantiate the VM with the wrong configurations on an attacker-controlled server. The visibility of services according to different attackers is shown in Figure 5.5.

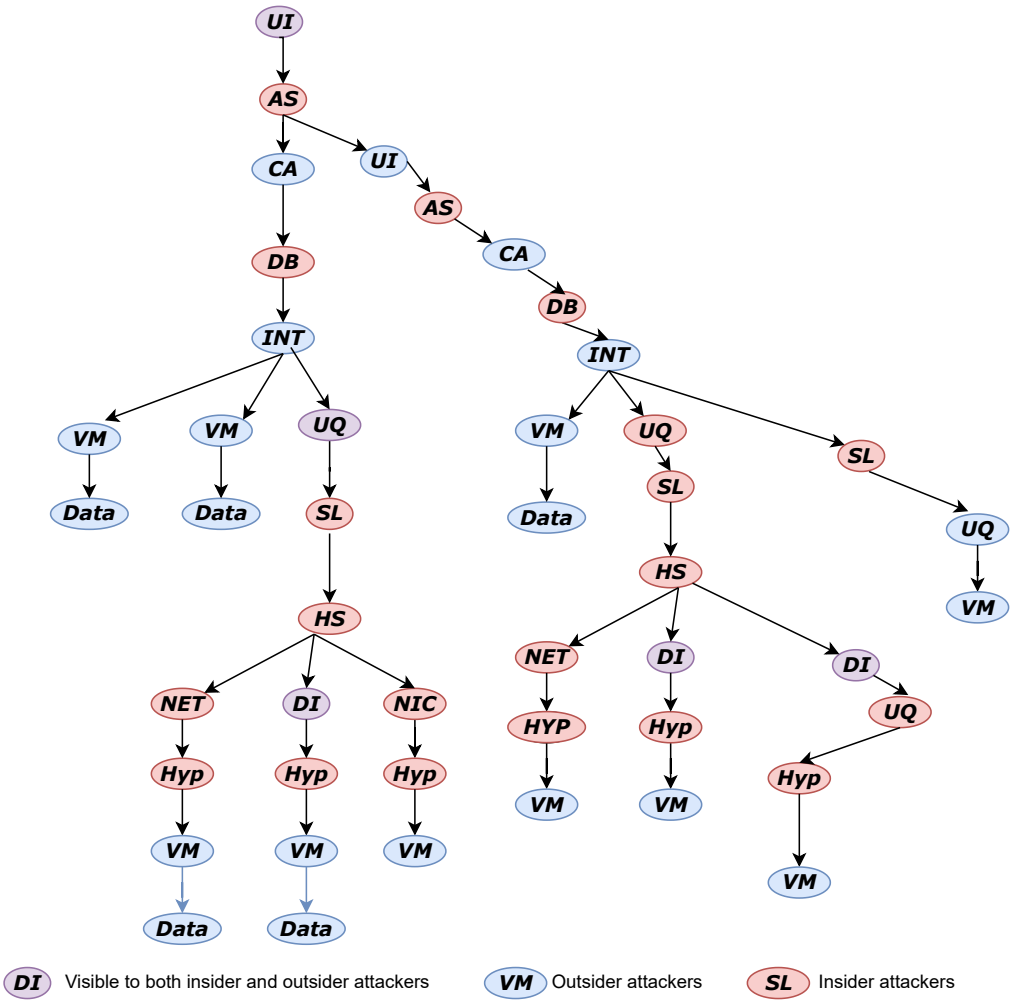

Fig. 5.5 Visibility of states to attackers

On the other hand, attack surfaces that can be exploited by an outsider attacker is limited in comparison to an insider attacker. An outsider attacker has a different (and limited)

externally set of services visible to them versus an insider attacker. The most common outsider interface is the authentication interface that can be compromised to gain full access privileges [120]. However, as mentioned in Chapter 2, an outsider can use the *DI* service to run malicious software in the Cloud.

As Figure 5.5 shows, there are services that are accessible to both insider and outsider attackers, e.g., *UQ* service. However, the access to this service for an outsider is limited compared to an outsider attacker. Therefore, the analysis of both the insider and outsider attacker gives insights into the services that are critical according to the class of the attacker. Further analysis can be done by creating rules for a different attacker profile, e.g., a combination of both the insider and outsider rules to explore attack surfaces for a collaboration attack.

5.4 Conclusion

This Chapter has explored threat analysis for the Cloud, which is technology-agnostic as the underlying information flow model is independent of the underpinning Cloud technologies. Furthermore, the Chapter elaborates on comprehensively analyzing the threats in the Cloud by considering the entire operational stack involved in launching a VM and the inherently elastic nature of the Cloud. This is achieved by modeling varied levels of functional stack and interfaces using Petri nets. The obtained model is the basis for extensively identifying base operational states that enumerate the “normal" sequence of Cloud operations. After verifying the functional property of the Cloud, threats are inserted into the model across the entire functional stack to determine the anomalous sequence of operations caused by the threat. Thus, correspondingly, enumerating the multi-level attack surface exploitability by attackers based on their accessibility of the service. Unlike traditional approaches that focus on a single layer of Cloud operations or consider a single technology, the threat analysis approaches, detailed in the Chapter, perform a comprehensive analysis to explore attack surfaces across varied levels of the operational stack and according to different attacker profiles.

Chapter 6

Requirements-based Threat Analysis

The previous chapters of the thesis addressed research questions 1 and 2, and the respective contributions led to the development of threat analysis approaches to evaluate the security of the Cloud at different abstractions of the functional stack. However, thus far, the threats in the approaches were considered independent of each other, i.e., the relationship among the threats was largely unexplored. Furthermore, specific user or service requirements were also missing from the threat analysis process. Therefore, this Chapter addresses research question 3, which investigates the interplay between user requirements, threats (and their variants[1])), and the Cloud services. Consequently, this Chapter contributes to the development of requirement-based threat analysis to include threats and their variants in the threat analysis process and to prioritize the threat analysis according to the user's requirements.

Consequently, the research in this Chapter addresses two main facets. First, an exploration of the relationship between the threats is investigated. The objective of the study is to identify threats that are closely related to each other in terms of, e.g., attack patterns, preconditions, etc., such that in the presence of the primary threat, its variants should also be considered in the security assessment. This helps system administrators to patch the primary vulnerability and assess the potential of its variants to undermine the security requirements. Second, a Design Structure Matrix (DSM) based approach is detailed to visualize the relationship between user's requirements, services, and vulnerabilities to investigate the security threats pertinent to specific requirements of the users. Among the advantages of the DSM are scalability and applicability of different algorithms (e.g., clustering, sequencing, and tearing) to perform comprehensive threat analysis.

In lieu of the above-mentioned aspects, the contributions of the Chapter can be summarized as follows.

[1] If two distinct vulnerabilities are characterized by a semantically equivalent exploit and consequence, then they are referred to as a variant of each other.

1. Investigating the relationship between vulnerabilities to explore potential variants of vulnerabilities that should also be considered in the threat analysis process.

2. Development of a DSM-based approach for investigating security threats by detailing attack surfaces stemming from vulnerabilities and variants of vulnerabilities within and across the layers of the Cloud model.

The remainder of the Chapter is organized as follows. In Section 6.1, exploration of the variants of vulnerabilities is studied. Section 6.2 details the insights and specifics of requirement-based threat analysis covering the relationship between user requirements, vulnerabilities, and the Cloud services.

6.1 Investigating Variants of Threats

This Section covers the first facet of the contribution which is to explore potential variants of threats to be included in the analysis. The number of disclosed vulnerabilities in databases, e.g., NVD [30], is constantly increasing, and analyzing each vulnerability is cumbersome and error-prone. These databases serve as a knowledge base for system administrators to look up for vulnerabilities that have the potential to compromise their systems. However, some vulnerabilities are overlooked which leads to outstanding vulnerabilities in the wild [121]. Even if a vulnerability has been patched, a variant of the vulnerability could exist in the database that could also undermine the security of the system. Thus, key questions system administrators face are (a) which vulnerabilities to prioritize and patch, and (b) are there variants of the prioritized vulnerabilities that should also be considered?

To assist administrators, various vulnerability assessment frameworks have been proposed that either measure the severity impact of the vulnerability qualitatively or quantify the risk associated with the vulnerability. Most notably, FIRST's Common Vulnerability Scoring System (CVSS) [122] is the defacto standard in the community. Each vulnerability in CVSS is characterized by several components, e.g., attack surface, attack complexity, and textual summary. Moreover, each vulnerability is assigned a "Base Score", which is given on a scale from 0 to 10, representing the severity of the respective vulnerability. The calculation of the base score depends on a variety of factors that constitute and reflect characteristics of the vulnerability. Another widely used public vulnerability database is the National Vulnerability Database (NVD) maintained by the NIST corporation [30]. The entries of the vulnerability database are Common Vulnerabilities and Exposures (CVEs) which summarize, in natural language, each vulnerability's technical description, its consequence as well as a list of software and respective versions affected by the vulnerability. A significant part of each

CVE is the score that is given following the CVSS. Besides NVD there are also databases from product vendors such as Microsoft, Google, Oracle, etc., that disclose vulnerabilities affecting their respective products.

The applicability of these databases is restricted due to their inherent limitations. More specifically, these databases are based on heuristics and the vulnerability reports tend to be subjective. Another aspect of the databases limiting their effectiveness is the classification of vulnerabilities which uses vulnerability consequence as the sole criterion. For example, if a vulnerability has a Denial of Service (DoS) as a consequence, then it is classified into the DoS class. This way limits the fine-grained and meaningful vulnerability classification that can be used to evaluate the extent to which the vulnerabilities are related to each other. Thus, in its current form, the state-of-the-art databases fail to give a holistic view of the system's security and facilitate systematic reasoning regarding vulnerability's potential variants on the system. On the other hand, CVSS base metric alone is not enough to assess the impact of the vulnerability, as shown in [123].

Thus, this contribution explores the variant(s) of a vulnerability and clustering vulnerabilities based on their "contextual similarity" rather than their consequences. A methodology is proposed to reveal the extent to which different vulnerabilities are related to each other. This methodology is explained in the following sections.

Figure 6.1 shows the stages involved in achieving the aforementioned goal. The proposed methodology is explained in the following that assesses and clusters vulnerabilities based on the structural and behavioral similarities among them.

I start by capturing the vulnerability data from multiple vulnerability databases to form a coherent view on the vulnerability reports in stage A, as depicted in Figure 6.1. Then in stage B, I extract and create a suitable feature space from these databases. The feature space represents a set of structural as well as behavioral attributes characterizing the vulnerability. The structural attributes identify the structural semantics of the vulnerability while behavioral attributes reflect upon the exploitation mechanism of the vulnerability. This information is encompassed in the textual description of the vulnerability. In Stage C, I use the feature space to build a language model and create contexts of the vulnerabilities persistent with their characteristics. In stage D, I employ these contexts to evaluate similarity among the vulnerabilities based on a similarity index. All four stages of the presented approach are explained in detail in the following sections.

6.1.1 Stage A. Vulnerability Data

There are multiple vulnerability databases that report on the disclosure of the vulnerabilities to the public. Thus, a coherent view of these vulnerability reports is created by combining

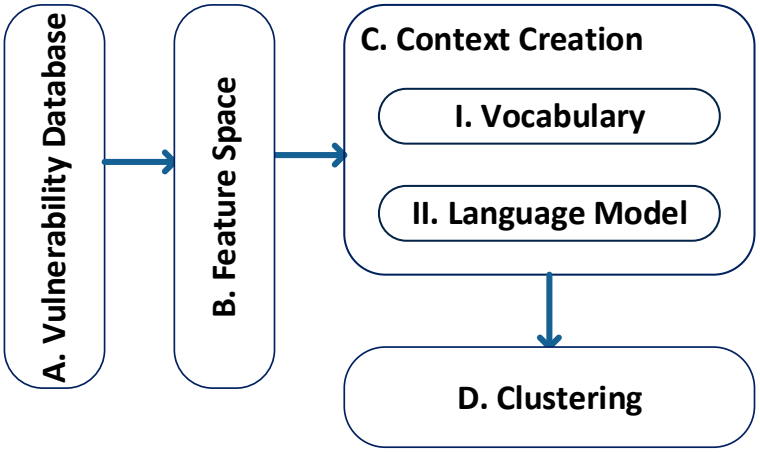

Fig. 6.1 Stages in the proposed methodology

suitable features from these databases to create a feature set that is representative of the vulnerability characteristics to be used in creating contexts. The extraction of the feature set is explained in stage B.

6.1.2 Stage B. Feature Extraction

In this stage, different databases are combined to create a feature space. The set of attributes is shown in Table 6.1 along with their types and the database source of the attribute.

The feature field in Table 6.1 identifies possible vulnerability attributes while the characteristics field in the table details the potential aspects of the respective feature. The type indicates the class of the feature and possible values for the type field are category, text, etc. For example, the feature has a category type if there are multiple possible aspects of the respective feature, or text if the feature contains a natural language description. The count field in the table indicates potential subcategories of the feature while the database source identifies the database which the feature stems from. As an example, the feature *vulnerability type* indicates the class to which the vulnerability belongs. This is normally registered as a consequence of the vulnerability, and since a vulnerability can have multiple consequences, therefore, it is of a category type. The number of potential subcategories for a consequence is 8 in the case of OpenStack vulnerabilities. A key feature of the vulnerability report is the textual summary which describes, in natural language, the technical details of the vulnerability and its exploit mechanism. I, therefore, utilize this textual description to

Table 6.1 Selected features from the vulnerability databases.

Feature	Characteristics	Type	Count	Database Source
Vulnerability Type	DoS	Category	8	CVE
	Bypass			
	Code Execution			
Access Vector (AV)	Network (N)	Category	3	NVD
	Local (L)			
	Physical (P)			
Access Complexity (AC)	Low (L)	Category	3	NVD
	Medium (M)			
	High (H)			
Authentication (Au)	Multiple (M)	Category	3	NVD
	Single (S)			
	None (N)			
Confidentiality Impact	Complete (C)	Category	3	NVD
	Partial (P)			
	None (N)			
Integrity Impact	Complete (C)	Category	3	NVD
	Partial (P)			
	None (N)			
Availability Impact	Complete (C)	Category	3	NVD
	Partial (P)			
	None (N)			
Product Type	OS	Category	2	CVE
	Application			
	None (N)			
Product Vendor	-	String		CVE
Product version	-	String		CVE
Summary	Keywords	Text	0-2000	NVD
	Contexts	Text	2	NVD

evaluate syntactical similarity among the vulnerabilities by creating vulnerability-specific contexts and assessing further vulnerabilities that follow similar contexts.

I further abstract the features of Table 6.1 into structural and behavioral attributes. This helps in creating specific contexts with respect to the vulnerability. The structural attributes define the structure of the vulnerability while the behavioral attributes reflect the exploiting mechanism of the vulnerability. All the features except the summary feature belong to the structural feature space as these attributes characterize the structural semantics of the vulnerability. The behavioral feature details how the vulnerability is exploited and the steps necessary to exploit the vulnerability. This information is extracted from the textual summary of the vulnerability. Both the structural and behavioral features are utilized in creating the context specific to the vulnerability which is explained in Stage C of the methodology.

6.1.3 Stage C. Creating Context

This section details creating contexts for the vulnerability considering the extracted feature space. Before proceeding to create the context, the vulnerability context is defined in terms of its structural and behavioral semantics and consequences, as follows:

- *Structure* represents the attributes of the feature set that characterize the structural semantics of the vulnerability.
- *Behavior* embodies the exploit mechanism from the summary that represents how the vulnerability can be exploited.
- *Consequence* defines the impact of the vulnerability such as a denial of service, information leakage, etc.

Vocabulary Creation:

The behavioral characteristics of the vulnerability are embodied in the description of the vulnerability. Therefore, a vocabulary is built from the textual summary of the vulnerability to be used in defining the vulnerability-specific contexts. To make the vocabulary meaningful with respect to the vulnerability, stop words are removed and only concentrate on the words that contain technical information about the vulnerability. Therefore, I also remove the words that can be extracted from other fields such as affected products and applications. This limits the vocabulary to the keywords that describe the behavioral characteristics of the vulnerability. I proceed further to create a language model based on this vocabulary.

Language Model:

The created vocabulary is used in the language model. This step is needed following the primary objective to create the context for each vulnerability that reflects vulnerability characteristics. Thus, each context is composed of structural and behavioral characteristics and consequences. The context provides the basic unit of analysis for comparing the strength of the connections between vulnerabilities. In order to achieve this, An application of the cosine similarity [124] is proposed to create a similarity matrix showing the distance among the vulnerabilities' analysis tuple. The critical aspect of the context is the behavioral keywords that are extracted from the vulnerability description. The current state-of-the-art schemes use Bag of Words (BoG) representation to convert the textual description of the vulnerability to the vector space model. However, this technique is not employed, as BoG has several limitations. First, the order and sequence are not maintained in the BoG representation and for an exploit, it is important to conduct the steps in the given order. Second, the BoG representation cannot be used to extract the syntactical semantics of textual description. However, both of these requirements are critical in creating context representing the characteristics of the vulnerability. Therefore, the bi-gram model is utilized to retain the order of words by considering two consecutive words from the vocabulary. To comprehensively cover different writing aspects of the summary, I create an alternative context by varying the order in a bi-gram. Both of these contexts are shown as:

$$Context\ 1 = [SC][(w_1, w_2)(w_2, w_3)(w_3, w_4)...(w_{n-1}, w_n)][CN(s)] \tag{6.1}$$

$$Context\ 2 = [SC][(w_1, w_3)(w_3, w_5)(w_5, w_7))...(w_{n-2}, w_n)][CN(s)] \tag{6.2}$$

The structural keywords are not subjective and do not depend on the security expert. Opposite to that, the textual description is highly subjective. Thus, in order to comprehend multiple writing styles, multiple contexts of the same vulnerability are derived based on the behavioral semantics. Another reason for creating multiple contexts is the lack of ground truth of the vulnerability for validating the context.

6.1.4 Stage D. Clustering

The created contexts form the basic unit to compare vulnerabilities. Thus, the next step is to perform hierarchical clustering to cluster the vulnerabilities that have a high similarity index. The reason for selecting hierarchical clustering is twofold. Firstly, it is critical to assess each vulnerability's contextual similarity with the rest of the data samples and this is

exactly what the hierarchical clustering algorithm is suitable for. Secondly, the hierarchical algorithm can have multiple distance metrics that can be used to assess the distance between the data samples. For this case, cosine similarity index [124] is used to compute the distances between the vulnerabilities. Cosine similarity is widely used to estimate text similarity as it works by evaluating the angular difference between the texts rather than their magnitudes. Thus, texts with the same orientation will have a cosine similarity of 1, and the texts with opposite orientation have a cosine similarity value of 0. This makes the cosine similarity index a suitable choice for the problem. By utilizing it, I create a cosine matrix between the vulnerabilities which is shown in Equation (6.3)

$$SM = \begin{array}{c} \\ Vul_1 \\ Vul_2 \\ \\ Vul_n \end{array} \begin{array}{c} \begin{array}{cccc} Vul_1 & Vul_2 & \dots & Vul_n \end{array} \\ \begin{pmatrix} V_{1,1} & V_{1,2} & \dots & V_{1,n} \\ V_{2,1} & V_{2,2} & \dots & V_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ V_{n,1} & V_{n,2} & \dots & V_{n,n} \end{pmatrix} \end{array} \tag{6.3}$$

The matrix *SM* is then given as input to the clustering algorithm to cluster vulnerabilities with high similarity (lowest distance). An example of a clustering vector is shown in Equation (6.4). The cluster consists of vulnerabilities (Vul_1 *and* Vul_2) with *distance* representing the respective distance between the pair. To further fine-tune the clustering distance among the two vulnerabilities can be set to a lower value resulting in clustering vulnerabilities with maximum similarity index or lowest distance. The count identifies the number of data samples of the respective iteration of the clustering algorithm.

$$Cluster = (Vul_1 \qquad Vul_2 \qquad Distance \qquad Count) \tag{6.4}$$

The result of applying cosine similarity is a matrix with values ranging between 0 and 1. A Similarity Matrix (SM) between the *N* vulnerabilities is shown in the matrix 6.3. The cosine similarity ranks the potential variants of a vulnerability based on the similarity among their contexts. An evaluation for OpenStack [125] is performed to explore vulnerabilities that are contextually equivalent and a potential result is shown in matrix 6.5.

$$SM = \begin{array}{c} \\ CV...8914 \\ CV...5362 \\ CV...5363 \end{array} \begin{array}{c} \begin{array}{ccc} CV...8914 & CV...5362 & CV...5363 \end{array} \\ \begin{pmatrix} 1 & 0.73 & 0.66 \\ 0.73 & 1 & 0.77 \\ 0.66 & 0.77 & 1 \end{pmatrix} \end{array} \tag{6.5}$$

The matrix 6.5 shows that the vulnerabilities are closely related and in the presence of a vulnerability the existence of remaining vulnerabilities should be checked. The advantage

of such classification results in revealing further vulnerabilities that have the potential to undermine the security of the system. This leads to patching a class of vulnerabilities instead of individual vulnerability patches.

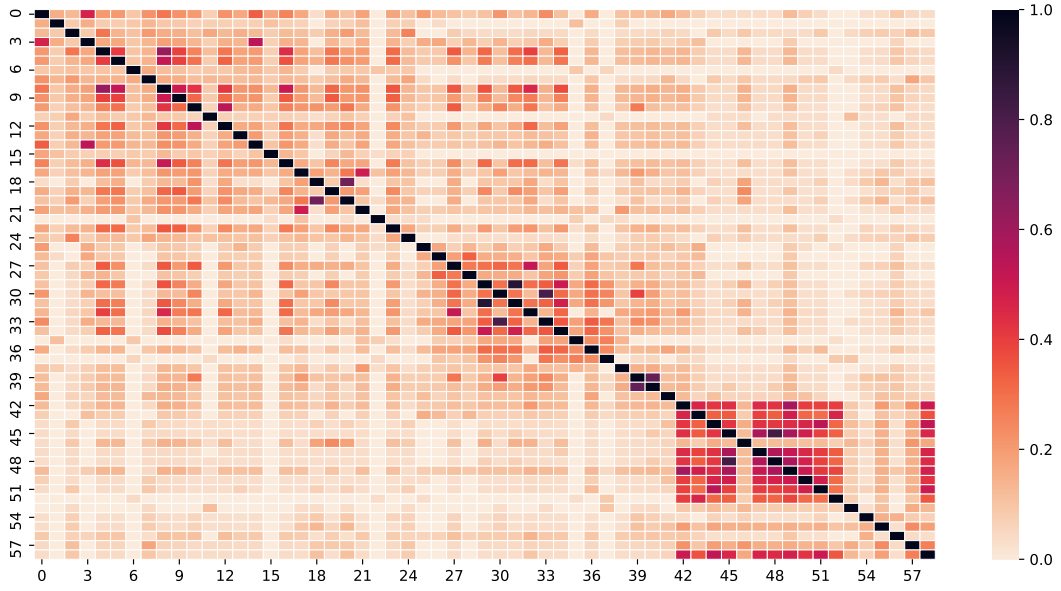

Fig. 6.2 Context-based similarity among the Cloud vulnerabilities

The similarity among OpenStack vulnerabilities is shown in Figure 6.2. As mentioned before, I use OpenStack as a use case but the methodology is a technology and product independent. The X and Y-axis are data points representing vulnerabilities and the colors represent the extent to which the vulnerabilities are related to each other. As the cosine similarity of 1 represents the maximum similarity, therefore, the diagonal of the heatmap shows this evidence since it represents self-comparison. From Figure 6.2, it is evident that the vulnerabilities between 42 to 52 have higher similarities. A further fine-tune analysis can be achieved to investigate the similarity among the vulnerabilities in terms of attack mechanisms, preconditions, etc. For instance, Figure 6.3 illustrates the similarity among the vulnerabilities, in terms of the attack mechanism used in the vulnerabilities. This essentially enables the security analysts to better understand the attack mechanism used in multiple vulnerabilities. Consequently, an effective countermeasure against the attack mechanism patches a class of vulnerabilities and thus, limits the reuse of the attack mechanism.

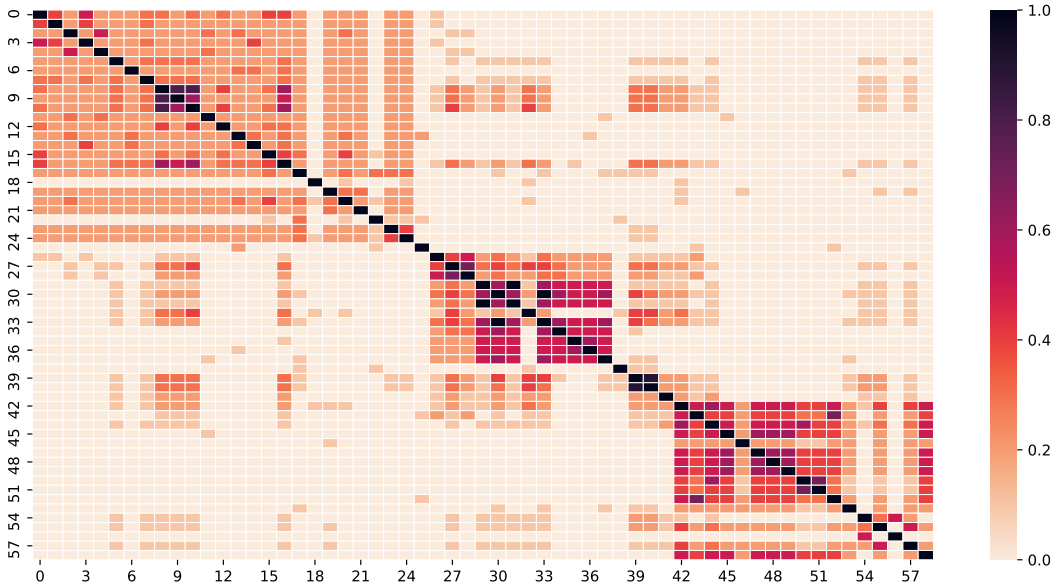

Fig. 6.3 Attack mechanism similarity among the Cloud vulnerabilities

6.1.5 Results and Discussion

The application of the proposed approach to the OpenStack vulnerabilities reveals interesting results which can be summarized as follows:

- The initial observation the analysis shows is that there exists a similarity among the vulnerabilities that go beyond the common consequence of the vulnerabilities. I investigated OpenStack vulnerabilities and observe that around 10 percent (cf., Figure 6.2) of the reported vulnerabilities have a varying degree of correlation between them. Moreover, the degree of correlation is higher among the vulnerabilities exploiting the same service. This is partially due to the deployment of a "similar" attack to exploit the functionality of the service. On the other hand, the similarity between the vulnerabilities exploiting different services shows a lower correlation.

- Similarly, the degree of attacker's reliance on utilizing the same mechanism in different vulnerabilities is also explored. Out of the vulnerabilities investigated, around 15 percent of the vulnerabilities have strong similarity (cf., Figure 6.3), i.e., the cosine similarity is above 0.5. Furthermore, I investigated the applicability of an attack mechanism across different services. The insight into this investigation is similar to the results of contextual similarity. I see a high degree of re-usability of the attacks

for the same service. For instance, there is a high chance of using cross-site scripting attacks with different payloads for authentication services. However, the same attack is less effective on other services in the Cloud platform.

- I also investigate the possible attack surface for both the authenticated and non-authenticated attackers. In the former case, I observe that most of the vulnerabilities or the attacker's actions were related to the manipulations of the VM that could lead to either a denial of service or cause the VM to behave incorrectly. In the latter case, since the attacker is not authenticated, therefore, the primary target of the vulnerabilities is the authentication service.

Thus, this contribution asserts that by revealing more elaborate patterns among the vulnerabilities, system defenders can patch subsequent vulnerabilities in the presence of a primary vulnerability. This ensures that the same vulnerability cannot be exploited on a different system or that a variant of the vulnerability cannot be used to compromise the system. Furthermore, a system defender can proactively hypothesize different threats that could potentially damage the critical services of the system.

6.2 Requirement based Threat Modeling

The previous section investigated the extent to which different vulnerabilities are related to each other. This section focuses on exploring the interplay between security requirements, services, and the corresponding threats to develop a requirements-based threat modeling and analysis of the Cloud. An ontology is developed depicting the relationships among different actors involved in the ontology. The primary actors are user, Cloud, and threats as shown in Figure 6.4. I explain ontology from the perspective of users and vulnerabilities in the following sections.

6.2.1 Users and Requirements Capturing

The users specify their requirements for the Cloud. For prioritizing the requirements, the user can specify the criticality of the requirement by assigning weight to it. These weights are assigned by using linguistic terms such as Highly-Critical (HC), Critical (C), Less-Critical (LC), and Not-Critical (NC)), i.e., the HC requirements are more critical and hence a violation of these requirements or threats arising from the violation also have high severity than the rest. As can be seen in Figure 6.4, requirements satisfy security goals which could be maintaining confidentiality, integrity, or availability.

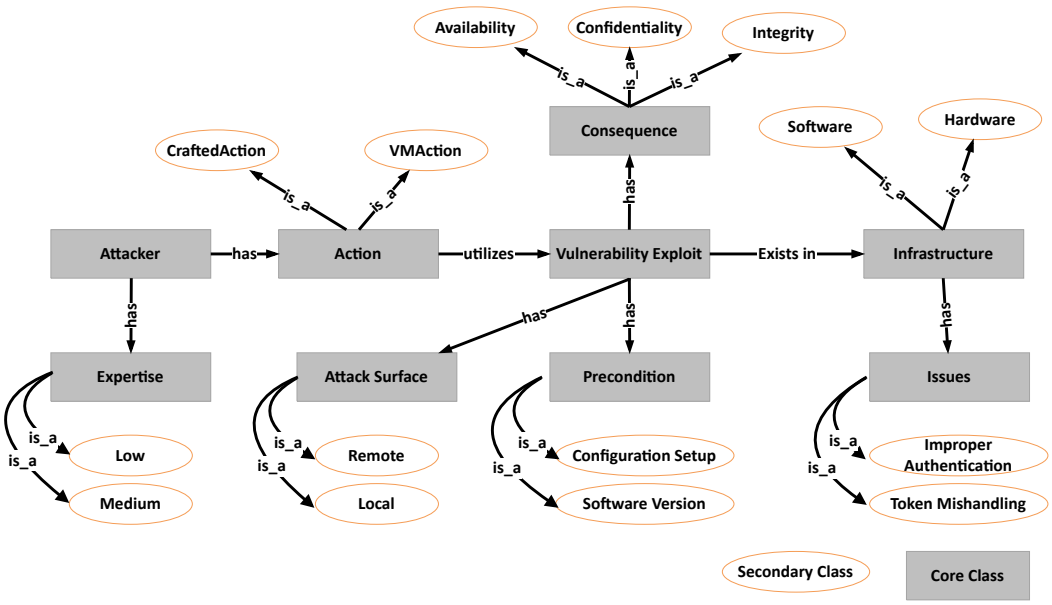

Fig. 6.4 Correlation among Services, requirements and threats

6.2.2 Vulnerability Perspective of the Ontology

This section describes the utility of an ontology from a vulnerability perspective that exists in a Cloud component or a service. The consequence of the vulnerability exploit is to violate the security goals and cause damage to the user and the Cloud. In order to undermine the system's security goals, an attacker tries to exploit the vulnerability by satisfying its pre-conditions. For this work, I restrict myself to these characteristics of the vulnerability without exploring the actual exploitation of the vulnerability by the attacker.

6.2.3 Using Design Structure Matrix for Threat Analysis

In order to perform threat analysis from the perspectives of varied actors, a mapping from the ontology to a Design Structure Matrix (DSM) data structure is achieved to show the cross-relations across the row/column entities. The advantages of a DSM include multi-facet representation and reordering of a DSM to a particular perspective. Furthermore, the DSM provides a coherent visualization of the ontology and the relationships across the actors and allows restructuring for varied actors. The mapping of the ontology to a DSM is shown in Figure 6.5.

Row#1 in Figure 6.5 exhibits the relation (marked as X) between requirement R1, service S1, and vulnerability V1. The requirement applies to the service S1 and the potential vulnerability V1 can be used to violate the requirement by exploiting it on S1 and thus, undermining the security goal of R1. The DSM can also maintain a transitive relation, for

		Requirements			Services			Vulnerabilities		
#		R 1	R 2	R 3	S 1	S 2	S3	V 1	V2	V3
1	R 1				X			X		
2	R 2					X			X	
3	R 3		X				X		X	X
4	S 1	X					X			
5	S 2		X							
6	S 3	X		X						
7	V 1	X			X					
8	V 2		X	X		X				
9	V 3			X			X			

Fig. 6.5 Interactions and variants of a vulnerability

example, row #4 in Figure 6.5 identifies a transitive relation between R1 and S3 through service S1. Furthermore, the interaction between S1 and S3 could be utilized in launching a multi-stage attack. Similarly, a requirement can depend on another requirement for its proper functionality. This is shown in row #3 where requirement R3 depends on R2 and in case of R3 violation, the functionality of R2 could also suffer. The DSM also offers varied options for partitioning and restructuring its data elements as a means of exploring inter-relations. For example, the DSM can be re-structured to identify the highest influential actor, i.e., the actor with the highest dependencies or interactions. This can be achieved by reordering the DSM rows and columns to transform the DSM into a matrix that has the highest dependency/interactions at the first row and the least dependent/interactions placed at the last row. This is achieved using the following two steps:

- Step 1: Actors with the highest number of dependencies/interactions have a maximum number of values (marked as X) in their respective columns and thus, placed at the top of the DSM.
- Step 2: Actors that are ad-hoc and do not provide information on other actors are placed at the bottom of the DSM. This can be identified by observing the empty columns in the DSM.

Applying these two steps to Figure 6.5 recursively reorders the DSM with actors having the highest dependency/interactions placed at row number 1. This reordered DSM is shown in Figure 6.6 which shows that requirement R3 has the highest number of dependency/interactions among the actors. Thus, the requirement R3 is the most influential while the service S2 has the least influence on other actors involved in the ontology.

The use of DSM is useful to visualize the interplay between requirements, services, and threats. Further, the DSM can be re-structured to visualize different perspectives represented

	R 3	V 2	R 1	R 2	S 1	S 3	V 1	V 3	S 2
R 3		X		X		X		X	
V 2	X			X					X
R 1					X		X		
R 2		X							X
S 1			X			X			
S 3	X		X						
V 1			X		X				
V 3	X					X			
S 2				X					

Fig. 6.6 Interactions and variants of a vulnerability

in the matrix. In the following section, an application of the DSM is demonstrated in profiling the security of the Cloud.

6.2.4 Profiling Security of the Cloud

In this section, I elaborate, using a case study, on the effectiveness of the proposed ontology and DSM-based approach for profiling Cloud threats from varied perspectives of the involved actors. Table 6.2 presents an excerpt of the data from the actors that are used to assess the threats holistically considering the relationship among the actors. The requirements field in the table describes the user requirement, its goal, and the respective priority assigned by the user. The goal indicates the security purpose of the requirement while vulnerabilities are exploited by the attacker to undermine this security goal. The vulnerabilities presented in the table are extracted from publicly available databases, e.g., NIST's national vulnerability database [30]. The database discloses every vulnerability, its impact, and affected products to the public. The Cloud services presented in the table are extracted from the model. However, a mapping between the respective services of the model to the actual OpenStack service name is achieved. Thus, the name field in Table 6.2 represents the corresponding OpenStack service name performing the designated functionality. For example, the Keystone service in OpenStack is responsible for identity and access management. The relations among requirements, threats, and services are also indicated in the table. For example, the requirement R1 serves multiple security purposes (CIA) for the user while the associated threat delineates CI of the security purposes by exploiting the vulnerability of the responsible service S1.

To comprehensively cover different aspects of the threat assessment, a DSM is created, shown in Figure 6.7, using the data of Table 6.2. The relationship among the Cloud actors is represented in the DSM by marking X in the respective row and column. For completeness, I

Table 6.2 Excerpt of the actors data for profiling threats in the Cloud

Requirements				Threats			Cloud Services		
ID	Description	Priority	Goal	ID	Impact	Description	Name (ID)	Functionality	Interconnection
R1	Each user should have a unique user name and password to utilize Cloud services	HC	CIA	V1	CI	Incorrect timestamps comparison for tokens leads to retaining access via an expired token	Keystone (S1)	Identity and Access Management	Database Service (S2)
R2	The data at rest should be encrypted and only the authorized user should be able to decrypt	C	A	V2	I	improper client connections handling leads to denial of service	Keystone (S1)	Identity and Access Management	Storage (S3)
R3	The data in transfer should be encrypted	C	C	V3	A	Changing the device owner of the port leads to bypassing IP anti-spoofing controls.	Neutron (S4)	Network related operations	Hypervisor (S5)
R4	The Cloud service providers should not be able to delete, modify or access user's data.	HC	CIA	V4	CIA	When using Xen as a hypervisor, attackers can obtain sensitive password information by reading log files	Hypervisor (S5)	Virtualization Management	Keystone, Storage

included goal and priority for identifying threats violating a specific goal or assessing the influence of the requirement in the Cloud. In the following section, a DSM utilization is shown by reordering and restructuring the DSM to assess the influence of different actors in the Cloud.

6.2.5 Extracting Influential Actors using DSM

This section illustrates how reordering the DSM (steps from Sec 3.4) can help identify the most influential actor. To rearrange the DSM, the steps presented in Section 6.2.3 are followed, i.e., by placing the most interconnected element, marked as X, in the first row and recursively performing this operation. The rearranged DSM is shown in Figure 6.8 having the most influential actor in the first row. The most influential component is requirement R1 a highly critical requirement for the user and also the most influential due to its interactions with most of the actors involved. Thus, violating this requirement or vulnerability affecting this requirement has the potential to propagate across the system due to its high degree of connections. Alternatively, from the threat analysis perspective, the DSM identifies critical aspects of the threat propagation and impact on the system. For example, vulnerability V1 can be used to undermine R1 by compromising service S1. However, S1 can also be compromised by vulnerability V2 and due to S1 interactions with services S2 and S3, the likelihood of propagation of threats should also be assessed. The DSM can also be used to lower the number of dependencies to restrict the impact of the respective actor.

	Requirements				Priority			Goal			Cloud Services					Vulnerabilities			
	R 1	R 2	R 3	R 4	HC	C	LC	C	I	A	S 1	S 2	S 3	S 4	S 5	V 1	V 2	V 3	V 4
R 1					X			X	X	X	X					X			
R 2						X				X	X		X				X		
R 3						X		X						X				X	
R 4					X			X	X	X					X				X
HC	X			X															
C		X	X																
LC																			
C	X															X			
I	X																X		
A	X	X																	
S 1	X	X										X	X			X	X		
S 2											X								
S 3											X								
S 4			X												X				
S 5											X		X						X
V 1	X							X	X		X								
V 2		X							X		X								
V 3			X							X				X					
V 4				X				X	X	X					X				

Fig. 6.7 Design structure matrix of the case study data

Besides reordering, many other algorithms are available for DSM to highlight the respective perspective. These algorithms are briefly explained below.

Sequencing: It enables understanding of the interactions among the vulnerabilities and the propagation of the vulnerabilities in the system. This is equivalent to traversing a path in the attack tree to determine possible exploits.

Tearing: It can be applied to limit a DSM structure to a point of interest. For example, a point of interest could be to reveal only vulnerabilities belonging to a particular service/technology.

Pattern Matching: This is useful in determining whether a particular pattern/set of vulnerabilities exists in the system. This can be used to reveal composite vulnerabilities, i.e., vulnerabilities composed of multiple other vulnerabilities.

Applying these algorithms not only reveals direct threats pertinent to the Cloud but also the indirect threats that violate specific requirements of a user. Also, the sequencing algorithm can be utilized to traverse the DSM structure to assess multi-stage attacks. Alternatively, the tear algorithm can be applied to the DSM to analyze it from a specific perspective. For example, Figure 6.9 shows the perspective of tearing the DSM for analyzing threats that impact the confidentiality of the system. As the Figure depicts, the critical vulnerability to undermine confidentiality is $V1$ which is exploited on service $S1$. Similarly, $V1$ can be used to undermine the requirements $R1$ and $R2$. Therefore, $V1$ patching should be prioritized in order to maintain the confidentiality of the system.

	R 1	R 4	S 1	R 2	V 4	R 3	V 1	V 2	V 3	S 5	HC	C	C	I	A	S 4	S2	S 3	LC
R 1			X				X				X		X	X	X				
R 4					X					X	X		X	X	X				
S 1	X			X			X	X									X	X	
R 2			X					X				X			X			X	
V 4		X										X	X	X	X	X			
R 3									X			X	X			X			
V 1	X		X										X	X					
V 2			X	X										X					
V 3					X	X										X			
S 5			X		X													X	
HC	X	X																	
C				X		X													
C	X						X												
I	X							X											
A	X			X															
S 4						X	X												
S 2			X																
S 3										X									
LC																			

Fig. 6.8 Reordering to extract most influential actor.

The previous sections covered the use of DSM to investigate the interplay between security requirements, services, and the threats in the Cloud. However, the threat's variant also poses a security risk to the system. Therefore, in the following sections, I explore the threats and their potential variants that should also be included in the threat analysis process.

6.3 Conclusion

This Chapter explored threat variants from the publicly available information and investigated the interplay between threats, security requirements, and the services in the Cloud. The potential variants of the vulnerabilities are explored from the public reports published in the databases. The advantage of including variants in the assessment is twofold. First, it allows the system administrators to analyze the security of their systems holistically. Second, it enables system administrators to prioritize patching based on the combined impact of vulnerability and its variants. A feature space is created from the vulnerability databases to define a context that becomes a unit of comparison among the vulnerabilities. This enables fine-grained classification of the vulnerabilities and reveals vulnerabilities that have similar exploit semantics in contrast to the current classification which is based solely on using the consequence of the vulnerability for classification. The second facet of the Chapter focused on the development of the requirements-based threat analysis. It is achieved by DSM which can be utilized to identify the most critical/influential as well as least critical/influential actors in the Cloud.

	C	V1	S1	R1	R2	R3	R4	S4	S5
C		X				X	X		
V1	X		X	X	X				
S1									
R1									
R2									
R3	X							X	
R4	X								X
S4						X			
S5							X		

Fig. 6.9 Viewpoint of the confidentiality requirement

Chapter 7

A Customer-Centric Approach to Validate the Cloud

The previous chapters have focused on the security assessment of the Cloud from the provider's perspective to identify threats facing the Cloud. The process of security assessment considered the varied attacker profiles and user requirements. On the other hand, this Chapter addresses the security concerns from the users' perspective, i.e., how can the Cloud users assess that their requirements (qualitative and quantitative) are satisfied by the Cloud Service Providers (CSPs)? To this end, this Chapter contributes by proposing an approach that users can utilize to validate the fulfillment of their requirements by CSPs. In case of a violation, the trust state of the CSP is downgraded, reflecting the violation of the requirement.

The services offered by the Cloud are attracting businesses and individuals to use the Cloud. However, the non-transparent architecture of the Cloud, the paucity of mechanisms to provide definitive assurance about the fulfillment of user requirements by Cloud Service Providers (CSPs), and the unclear assurance on security facets of service delivery impede many businesses from adopting the Cloud services. These often result in Cloud Service Customers (CSCs) being unable to trust the CSPs. Broadly, trust implies reliance on something that is expected to behave or deliver as promised [126]. In the context of the Cloud, trust is referred to as the degree of reliance on the services offered by the CSP with respect to the customer's requirements.

Members of the Cloud community[1] advocate specifying service provisions in Service Level Agreements (SLAs) to establish common semantics to provide and manage assurance. Fundamentally, an SLA represents a formal contract between a customer and a CSP that specifies service provisions with respect to the customer's requirements. The SLA includes a

[1]For example, standard bodies such as NIST, the European Network and Information Security Agency (ENISA), Cloud Security Alliance (CSA), ISO/IEC, and the European Commission.

list of attributes which are the measurable elements that specify service levels provided by the CSP in comparison to the customer's requirements along with the agreed-upon quality level for each attribute (e.g., latency, throughput, etc) along with penalties for non-delivery of services. Thus, a logical way to assess the trust of the CSP is to validate the SLA.

Although the state of the art predominantly focuses on the methodologies to negotiate and design Cloud SLAs [127–130], most of these methodologies assume that the CSPs are actually providing the services as agreed in the SLAs. The techniques to detect SLA violations are conspicuous by their paucity. A violation happens if an agreed SLA is not fulfilled by the CSP. In other words, the SLA is violated if the CSP is not provisioning the services according to the customer's requirements. Therefore, it is important to provide customers with comprehensive support in order to (i) validate the SLA through the course of time and operations at the CSP and (ii) enable automatic detection of SLA violations.

Thus, this Chapter aims to solve the aforementioned issues by proposing a novel reasoning approach to:

- Assess the qualitative and quantitative attributes 'to-be-provided' by the CSP and 'required' by the customer. Set theory is used for the qualitative attributes, and propose pairwise comparators for quantitative attributes. This comparison forms the basis to validate the SLA and detect violations on the customer's requirements.
- Classify violations into "trust states" according to the preferences specified by the customer.
- Validate the trust of Cloud IaaS by applying the proposed methodology to detect SLA violations during the launch and migration of a virtual machine.

The rest of the Chapter is organized as follows. Section 7.1 introduces the basic concepts on SLAs. Section 7.2 reviews contemporary SLA validation approaches. Subsequently, Section 7.3 describes the proposed methodology, and in Section 7.4, case studies are presented to evaluate the services involved in a Cloud IaaS. Section 7.5 concludes the Chapter.

7.1 Basic Concepts

A Cloud SLA describes the provided services and represents a binding commitment between a CSP and a customer. The SLAs outline the desired services, each of which contains a list of attributes. Each attribute is composed of one or more metrics, as relevant, that help in the measurement of the Cloud services by defining parameters and measurement rules. Hence, the SLA contains a list of attributes, with corresponding desired values, that the CSP

is committed to fulfilling. If any of these committed values is not fulfilled by the CSP, then the SLA is violated. In practice, one way to assess the trust of the CSP is by periodically validating the SLA. Thus SLA monitoring schemes are used to quantitatively validate what a CSP is providing and which assurances are actually met.

Based on the analysis of the state of practice presented in [131], Cloud SLAs are typically modeled using a hierarchical structure as shown in Figure 7.1. The root of the structure defines the main container for the SLA. The intermediate levels (second and third levels in Figure 7.1) are the services and the lowest level represents the actual attributes committed by the CSP and consequently offered to the customer. These attributes form the threshold values which are specified in terms of metrics in relation to the customer's requirements.

The concept of an SLA is formalized using the following definition.

Definition 2 *An SLA consists of a set of services $S = s_1, \ldots, s_n$. Each service s consists of a finite positive number n of attributes k_i; where $i = 1 \ldots n$. Each attribute k_i consists of m different values v_i; such that $k_i = v_{i,1}, v_{i,2}, \ldots, v_{i,m}$. Each value implies a different service level offered by the CSP and required by the customer. Each k_i value is mapped to a numerical value according to its type/range.*

Note that, the attributes can have varied types/ranges of qualitative and quantitative values. Hence, a process for comparing different attributes across the customer's requirements is needed in order to detect SLA violations by the CSP.

7.2 Related Work

With the rapid growth of Cloud services, multiple approaches have emerged to assess the functionality and security of CSPs. In [132], the authors proposed a framework to compare different Cloud providers across performance indicators. In [129], an Analytic Hierarchy Process (AHP) based ranking technique was proposed that utilizes performance data to measure various Quality of Service (QoS) attributes and comparatively ranks the CSPs. In [133], a framework that enables a comparison of Cloud services based on critical characteristics is presented. However, these studies (i) focused on assessing the performance of Cloud services but not their security properties and (ii) did not consider SLA validation for identifying violation-prone service providers.

Security requirements for non-Cloud scenarios have been addressed by Chaves et al. [134] who explored security in SLAs by proposing a monitoring and controlling architecture for web services. In [135] and [136], the authors proposed a technique to aggregate security metrics from web services. However, the authors did not propose any techniques to assess

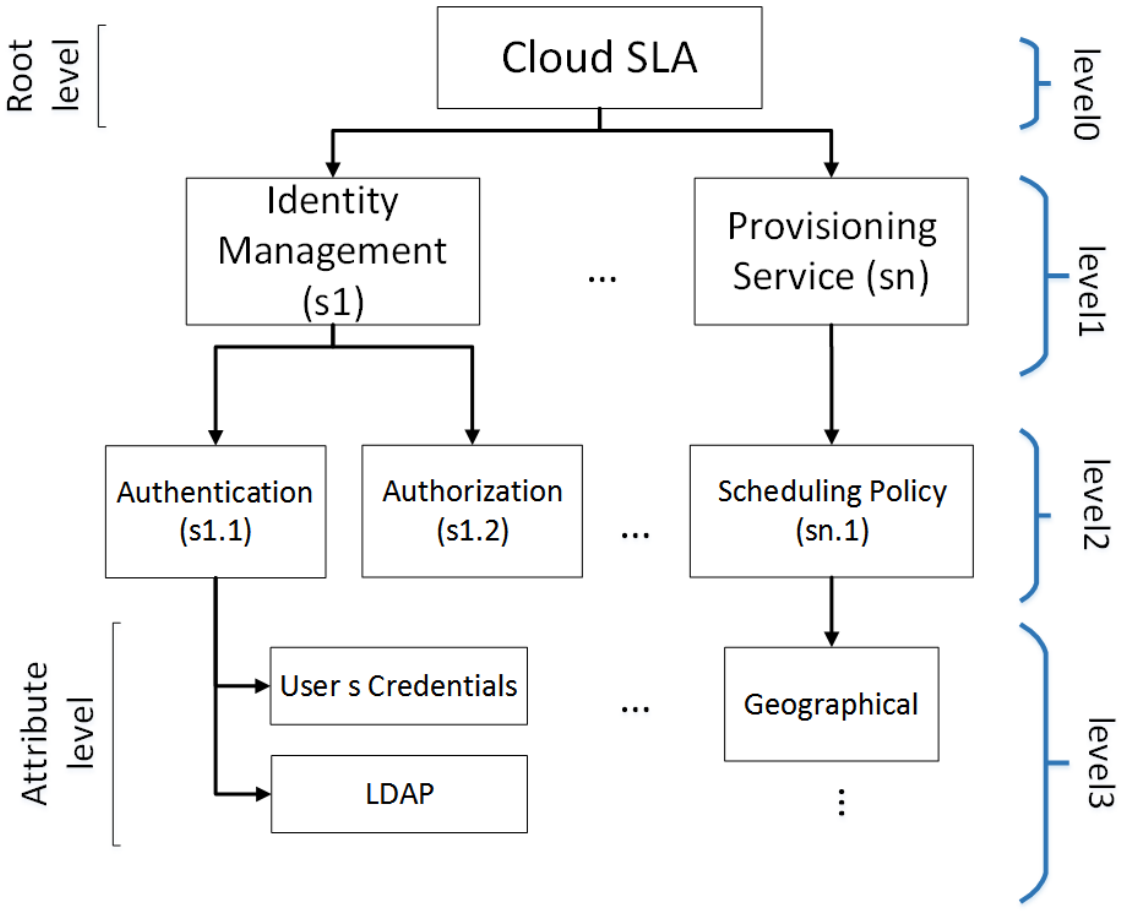

Fig. 7.1 Cloud SLA hierarchy

Cloud SLAs or empirically validate the proposed metrics. Luna et al. [127] presented a methodology to quantitatively benchmark Cloud security with respect to the customer's defined requirements (based on control frameworks). In [128], the authors presented a framework to compare, benchmark, and rank security levels provided by two or more CSPs. However, in both of them, the SLA validation was not covered.

An SLA validation framework can help in identifying violation-prone service providers. However, in order to serve the customer best, a trust model should take into account all the available sources of information including the customer's requirements and feedback. In [137], the authors proposed a multifaceted Trust Management (TM) to identify trustworthy Cloud providers in terms of different attributes. However, their assessment considered trust as a security service level and furthermore, they did not manage and maintain dynamically changing trust values. In both [138, 139], the authors considered the SLA validation as the main factor for establishing trust in grid and web service providers. However, they only considered QoS attributes.

The methodology presented in this Chapter differs from the above-mentioned works in respect to the attributes under consideration and maintaining the dynamic state of trust. Trust models for Cloud computing need to take Cloud specific attributes into account and these go beyond the usual QoS parameters. The Cloud-specific attributes are considered by evaluating services in launching and migrating a VM. A validation of these attributes to detect violations periodically and the effect of these violations on the state of trust. A state transition-based approach is used to model different trust states and demonstrate how violations dictate the transition across them.

7.3 Proposed Methodology

This section describes the research methodology to validate an SLA by comparing the service provisions of a CSP with the customer requirements over the lifetime of the service. The stages involved in the methodology are shown in Figure 7.2. In Stage A, the customers specify their requirements, and the CSPs specify their service provisions. The customers then select a CSP that "best" matches their requirements. In Stage B, the selected CSP is monitored to acquire the attribute values of the desired services. Stage C validates these attributes with the customer's requirements and assesses the current state of trust. As mentioned earlier, trust is referred to as the degree of reliance on the offered services with respect to the customer's requirements. Therefore, if a CSP is provisioning the services according to the requirements, the CSP is consequently fulfilling the SLA to be deemed to be in a trusted state. However, if the requirements are violated, consequently, the state of trust is changed to reflect the degree of violation. The methodology is periodically[2] applied to assess the current state of trust which is modeled in Stage D using a state diagram.

7.3.1 Stage A. Requirements Specification

In this stage, the customers specify their requirements, and the CSPs specify their provisions. The customer evaluates and ranks each CSP according to the requirements and selects a CSP that best matches these requirements. Any ranking algorithm can be used to select the CSP, e.g., the ranking algorithm proposed in [128]. A periodical validation of the selected CSP provisions is targeted in this Chapter. For clarity, the customer's requirements are specified that are used to validate the CSP provisions. The same SLA hierarchical structure (cf., Figure 7.1) is used to model requirements. The customer can specify his/her requirements at different

[2]A periodic interval can be chosen or an event-based schema using a violation threshold as a trigger can be used.

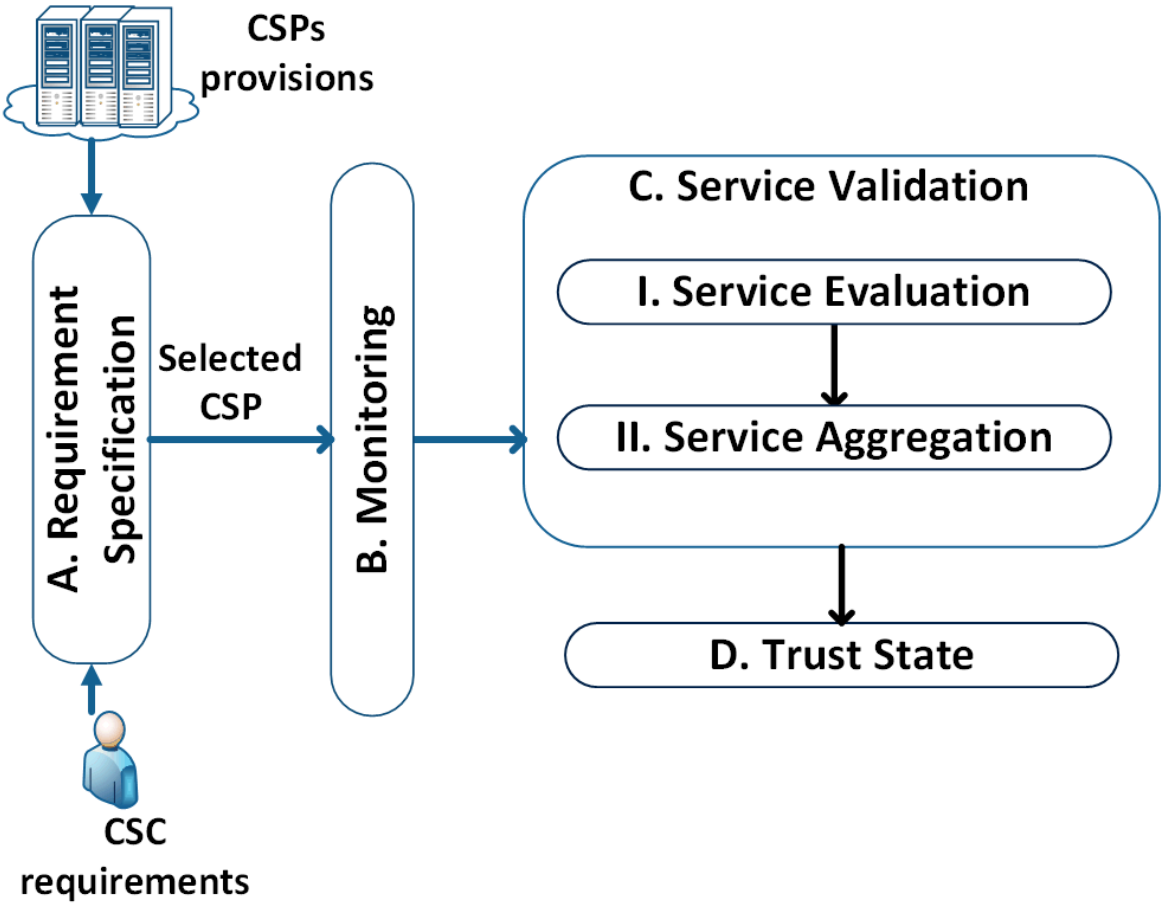

Fig. 7.2 Stages of the proposed methodology

levels of granularity and can specify the priorities of the requirements by assigning weights to them. Furthermore, the customer can specify weights by using linguistic terms (Highly-Critical (HC), Critical (C), Less-Critical (LC), and Not-Critical (NC)) for the attributes and/or services. The highly-critical attributes have high importance for the customer, while not-critical attributes have the least importance. Thus, violating highly-critical attributes have severe implications on trust than the rest of the attributes. Critical and less-critical specify the customer's different degrees of importance regarding these requirements, where they can accept varied values depending on the considered scale.

7.3.2 Stage B. Monitoring the Selected CSP

This stage involves monitoring the selected CSP to capture the values of the attributes. Monitoring plays a significant role in identifying violations as it provides the attribute values from the CSP for comparison. There are various monitoring schemes proposed by industry

and academia, e.g., Cloud Watch [140] which is used to monitor Amazon EC2, Cloud Stack [141] as an open source framework for monitoring the Cloud, and Ayad and Dipel [142] proposed agent-based monitoring for virtual machines in an IaaS environment. The primary interest of the proposed methodology is in the values of the attributes from the CSP to compare with the requirements of the customer. Thus, any monitoring technique can work with the approach, and thus, the details of these monitoring schemes are beyond the scope of this thesis. Interested readers are referred to [143], a survey on monitoring schemes for the Cloud.

7.3.3 Stage C. Service Validation

The goal of this stage is to validate the service provisions of the CSP in comparison to the requirements of the customer. As mentioned in Section 7.1, each service consists of a set of attributes that are necessary to provide the desired functionality. Therefore, services are validated using the attribute values provided by the CSP and required by the customer.

Service validation forms the basis to detect a requirement violation. If a CSP is fulfilling all the customer's requirements, consequently, the CSP is in a trusted state. However, in presence of a requirement violation, the Impact Factor (IF) of the violation is assessed, which determines the severity of the violation by measuring the distance between the provided and the required values. In Table 7.1, different levels of IF are defined that relate to the severity of the violation. The levels are normalized between 0 and 1 based on the degree of deviation from the required value, i.e., the greater the deviation, the more severe the violation. Ideally, IF should be 0, which indicates no violation from the CSP. The second level ($0.1 < IF \leq 0.25$) indicates violations that have minimum severity as IF deviated minimally. The third level ($0.25 < IF \leq 0.5$) indicates the medium severity of the violations, while the last level ($0.5 < IF \leq 1$) specifies that the distance between the service provision of the CSP and the customer's requirement is the farthest and hence, indicates the maximum severity of the violation.

Table 7.1 The relation of Impact Factor to the severity of the violation(s).

Impact Factor	Severity of violation(s)
$IF = 0$	No Violation
$0.1 < IF \leq 0.25$	Minimum Severity
$0.25 < IF \leq 0.5$	Medium Severity
$0.5 < IF \leq 1$	Maximum Severity

The subsequent Phases I and II evaluate each discrete service to detect requirement violations and to assess the impact of these violations according to the levels described in Table 7.1.

Phase I. Service Evaluation

The services can have multiple attributes that can be either quantitative or qualitative in nature. The attributes such as *CPU*, *RAM* and *disk space* are quantitative attributes while *scheduling policy* and *authentication methods* are examples of qualitative attributes. This complicates the process of modeling and comparing values to evaluate a quantitative metric. To address this complexity, I first classify different types of attributes and provide a validation method for each type. The attributes can be classified as either numerical or unordered sets. This classification sets the basis for validating attributes, as the validation method for numerical values differs from the validation of the unordered set.

Numerical: The attributes such as are *CPU*, *network latency* and *bandwidth* are classified as numerical since their values can monotonically increase or decrease. A validation for numerical attributes is achieved by comparing values provided by the CSP with the values required by the Cloud Service Customer (CSC). The relationship between the *CSP* and the *CSC* with respect to attribute k and value V is represented using a pairwise comparison such that:

$$CSP_k / CSC_k = \frac{V_1}{V_2} \tag{7.1}$$

i.e., assume a *CSP* and a *CSC*, with values V_1 and V_2 for *network latency* attribute respectively, such that: the CSC's required value for network latency is $100ms$ (i.e., $V_2 = 100ms$) and assume $100ms$ is provided by the CSP (i.e., $V_1 = 100ms$). The pairwise comparison relation between V_1, V_2 is defined as: $\frac{V_1}{V_2} = 1$. Therefore, *CSP* is fulfilling the requirement.

If the result of the pairwise comparison is not equal to 1, this indicates a violation. This violation could be due to over-provisioning when the result is greater than 1, or under-provisioning when the result is less than 1. The over-provisioning of resources is also considered a violation since a malicious administrator or an inside attacker could over-provision the attribute and the customer would have to pay for this additional provisioning.

In case of a violation, the impact factor of the violation is calculated as:

$$IF_k = |1 - \frac{CSP_k}{CSC_k}| \tag{7.2}$$

Equation 7.2 calculates the impact factor of the violation as an absolute value. The same levels as mentioned in Table 7.1 are used to indicate the severity level of the violation. An

impact factor IF_k of 1 indicates that the CSP has violated the attribute with maximum severity while IF_k of 0 indicates no violation.

Unordered Set: An unordered set is defined for the attributes that are qualitative in nature. These attributes include *access policy* and *authentication methods* and validating these attributes comparatively is not possible. For qualitative attributes set theory is utilized to detect violations and estimate the impact factor of violations. The advantages of set theory are twofold. Firstly, its ability to generalize logic behavior, i.e., the same operations work for *access policy* and *scheduling techniques* although they belong to different services and have a different set of values. Secondly, using sets for qualitative attributes provide support to evaluate the impact factor of the violations by calculating dissimilarity between sets. This dissimilarity could be a result of adding or removing a value in the set. Assuming that for a *CSC* the required set for *access policy* is $\{read, write\}$. A violation is detected whenever a *CSP* adds/deletes any value to/from the access policy and as a result its impact factor should be assessed.

The violations are identified by calculating the symmetric set difference between the *CSP* provided set and the *CSC* requested set. If the symmetric difference results in a *null* set, this implies that the *CSP* provisions and the *CSC* requests are the same and hence no violation. However, if the result is not a *null* set, then a violation has occurred. Lets suppose the following *sets* list an attribute k values of a *CSP* and a *CSC*.

$$CSP_k = \{v1, v3, v5\}$$
$$CSC_k = \{v1, v3\}$$

To find out the violation(s), I calculate the symmetric difference between the values provided by the *CSP* and those requested by the *CSC*, such that:

$$CSP_k - CSC_k \neq CSC_k - CSP_k \neq \{\emptyset\} \tag{7.3}$$

If the result of the symmetric difference is not a *null* set, then the *CSP* is violating the customer's requirement(s). To detect a violation due to the addition or the removal of a value, $CSP_k - CSC_k = \{v5\} \neq \{\emptyset\}$ and $CSC_k - CSP_k \neq \{\emptyset\}$ is respectively used.

After identifying the violations, the impact factor of these violations is calculated similarly to the numerical type. For sets, the Jaccard Index [144] is a natural choice as it calculates dissimilarity between the sets by estimating the distance between the provided and the required values. This distance measurement is equivalent to the definition of the impact factor and calculated as:

$$IF_k = 1 - \frac{|CSP_k \cap CSC_k|}{|CSP_k \cup CSC_k|} \tag{7.4}$$

Equation (7.4) evaluates the impact factor of the violation. The same levels as mentioned in Table 7.1 are used for qualitative attributes, i.e., $IF = 0$ indicates no violation while $IF = 1$ indicates the maximum severity of the violation.

Using the above comparison metrics for each attribute, the impact factor(s) of the violating attribute(s) is obtained. This results in a matrix of size N if there are N attributes in a service. In order to evaluate the service assurance, I aggregate the impact factors of all attributes belonging to a service.

Phase II. Service Aggregation

After validating the lowest (attribute) level of the SLA, I move up in the hierarchy (cf., Figure 7.1) and assess the aggregated assurance of the service provided by the CSP. In Phase I, the impact factors of each attribute is evaluated and these are further used as input in this phase for an aggregation method. Equation 7.5 is used to aggregate the impact factors of attributes IF_i along with their weights to evaluate the service impact factor.

$$IF_{service} = \sum_{i=1}^{n} \left(\frac{IF_i * weight_i}{n} \right) \tag{7.5}$$

By using this equation, $IF_{service}$ results in a value between 0 and 1 and uses the same levels as described in Table 7.1. Thus, $IF_{service}$ of value 0 indicates that the service is provisioned as required by the customer while value 1 indicates that every requirement was violated with the maximum severity level.

7.3.4 Stage D. Trust State

After assessing services individually, the next step is to aggregate the impact factors of the services $(IF_{service})$ to calculate the impact factor at root level IF_{root}. Services are aggregated according to Equation 7.5 and IF_{root} uses the same levels as described in Table 7.1. I use IF_{root} to assess the state of trust and implications of the violations using a state diagram which is shown in Figure 7.3. Each state represents the severity level of the violations, i.e., state 1 represents no violation while states 2, 3 and 4 represents minimum, medium and maximum severity of the violations respectively.

- State 1: I evaluate IF at the root level and if $IF = 0$, this implies that the CSP has not violated any requirement and consequently, it is in the trusted state. This is shown as

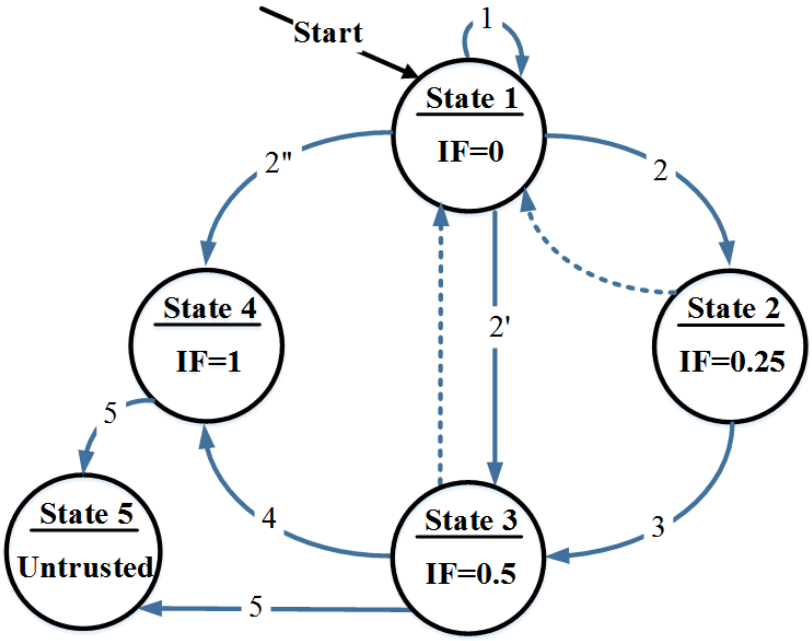

Fig. 7.3 Effect of impact factor on states of trust

transition 1 in the state diagram. However, if violations are detected, then the state of the CSP is changed to either state 2, state 3 or state 4 (as indicated by the respective transitions 2, 2' and 2") corresponding to the severity of the violations.

- State 2: The transition 2 in the state diagram illustrates that the state of the CSP is changed to state 2 as the violations result in an IF value of between 0.1 and 0.25. I evaluate IF again in state 2 to deduce the next state of the CSP. If the CSP has not violated any requirement ($IF = 0$) then the state is changed back to state 1. However, if there are again violations with minimum severity ($0.1 < IF \leq 0.25$) then the state is changed to state 3 to indicate the aggregated impact of violations.

- State 3: The CSP state is changed to this state from state 1 if the detected violations in state 1, resulted in medium severity. Thus, the CSP is moved to state 3 to indicate this behavior. A counter is used in states 3 and 4 to ascertain how many times these states have been transited. The threshold is a value specified by the customer to signify how many times the customer can endure violations. This counter plays an integral role in deciding the next state of the CSP. From state 3, the CSP can recover to state 1 if no further violations are detected, and the count is below a specified threshold.

- State 4: This state indicates the maximum impact of violations and the state of the CSP is changed to this state if the (aggregated) impact factor of violations results in a value of between 0.5 and 1. From this state, CSP cannot recover to state 1.
- State 5: This is the untrusted state and the state of the CSP is permanently changed to this state if the CSP violates requirements more than the threshold specified by the customer.

The state diagram is useful in determining the current state of trust of the CSP based on the customer's requirement violations. In the next section, the methodology is applied to detect violations and assess the trust of the CSP during Cloud operations.

7.4 Case Study: Trust Assessment of the Cloud

This initial validation scenario demonstrates how a Cloud customer can apply the methodology presented in this Chapter to assess the state of trust of the CSP during the course of operations. I evaluate the trust of the CSP by considering two scenarios: (1) launching a VM and (2) migrating a VM. To start a VM, the customer requests the CSP to boot an instance of a VM according to his/her requirements. IF_{root} is evaluated to determine the state of trust during the course of launching a VM. The migration scenario considers moving a VM from one physical host to another in compliance with the customer's requirements. IF_{root} is evaluated again to ascertain violation(s) during the migration phase and the impact of these violations on the state of trust.

The services involved in the operation of the Cloud are comprehensively covered in Chapter 3. Table 7.2 presents a sample dataset used for the services involved and the values associated with these services. In order to perform a comprehensive validation, the selected attributes include both qualitative and quantitative attributes. Furthermore, weights assigned by the customer to indicate his/her priorities are specified as a numerical value such that Highly-Critical *(HC)* indicates a relative value of *1*. Critical *(C)* and Not-Critical *(NC)* can be considered any intermediate values between *1* and *0*. In this analysis they indicate a relative values of *0.7* and *0.3* respectively.

In the rest of the section, I outline the computation process to assess the state of trust of the CSP with respect to the requirements defined in Table 7.2.

7.4.1 Case I: Launching a VM

Chapter 3 comprehensively covers the services interaction in launching and migrating a VM. Therefore, with respect to these services, I use the data shown in Table 7.2 for assessing

Table 7.2 Excerpt of SLA's from CSPs and customer's requirements.

Cloud secSLA					Customer (CSC)		CSP	
Services			Attributes	Values	*req*	*weight*	Case I	Case II
Root	ID Management *IM*	Authncation *IM*1	Auth *IM*1.1	Unordered set	Credentials	HC	Credentials	Credentials
		Authorization *IM*2	Policy *IM*2.1	Unordered set	launch, restart	C	launch, restart, delete	launch, restart
			Roles *IM*2.2		user		user	user
	IaaS *IS*	Provisioning *IS*1	CPU *IS*1.1	Numeric	6.4 GHZ	LC	2.4 GHZ	6.4 GHZ
			RAM *IS*1.2		8 GB		8 GB	8 GB
			DISK *IS*1.3		1 TB		1 TB	1 TB
			Scheduling technique *IS*1.4	Unordered set	Location based	HC	Location based	Random
	Storage *SO*	Storage *SO*1	Type *SO*1.1	Numeric	Persistent	C	Persistent	Persistent
			Location *SO*1.2		Local		Local	Local
	Network *NW*	Network *NW*1	BW *NW*1.1	Numeric	100 Mbps	NC	10 Mbps	100 Mbps
			Latency *NW*1.2		100 ms		10 ms	100 ms

the state of the trust of the CSP. The evaluation starts from the attribute level, i.e., $IM1.1$ is assessed to check if there is any violation in *authentication* during the course of launching a VM at the CSP. Equation 7.3 is used to detect violation by calculating the symmetric difference between the sets provided by the CSP and requested by the CSC, such that:

$$CSP_{IM1.1} - CSC_{IM1.1} = CSC_{IM1.1} - CSP_{IM1.1} = \{\emptyset\}$$

The result of the symmetric difference is the empty set which indicates that the CSP is fulfilling the customer's requirement and hence the impact factor is 0. As $IM1$ service consists of only one attribute and, therefore, the service impact factor is the same as the attribute, i.e., $IF_{IM1} = IF_{IM1.1} = 0$.
Next *authorization* is evaluated by validating $IM2.1$ and $IM2.2$ using symmetric difference calculation.

$$CSP_{IM2.1} - CSC_{IM2.1} \neq CSC_{IM2.1} - CSP_{IM2.1} \neq \{\emptyset\}$$
$$CSP_{IM2.1} - CSC_{IM2.1} = \{delete\}$$

Since the symmetric difference is not the empty set, this indicates a violation and, therefore, I assess its impact factor using the Jaccard Index (cf., Equation 7.4) such that:

$$IF_{IM2.1} = 1 - J(CSP_{IM2.1}, CSC_{IM2.1})$$

Where,

$$J(CSP_{IM2.1}, CSC_{IM2.1}) = \frac{|\, CSP_{IM2.1} \cap CSC_{IM2.1} \,|}{|\, CSP_{IM2.1} \cup CSC_{IM2.1} \,|} = \frac{2}{3}$$

Thus $IF_{IM2.1} = 1 - \frac{2}{3} = \frac{1}{3}$

In a similar way, $IM2.2$ impact factor is calculated. Using Equation 7.3, I premeditate the symmetric difference for $IM2.2$ which detects no violation.

$$CSP_{IM2.2} - CSC_{IM2.2} = CSC_{IM2.2} - CSP_{IM2.2} = \{\emptyset\}$$

The impact factor of $IM2.1$ and $IM2.2$ is 0.66 and 0 respectively. These impact factors are aggregated to assess service provision using Equation 7.5.

$$IF_{IM2} = \frac{IF_{IM2.1} * weight_{IM2.1} + IF_{IM2.2} * weight_{IM2.1}}{2}$$
$$IF_{IM2} = \frac{0.66 * 0.7 + 0 * 0.7}{2} = 0.23$$

After each service calculation, the transition to an upper level in the hierarchy happens to calculate the impact factor of the domain that contains these services. IF_{IM1} and IF_{IM2} are aggregated to get the IF_{IM} such that:

$$IF_{IM} = \frac{IF_{IM1} + IF_{IM2}}{2} = 0.11$$

As expected IF_{IM} is above 0, which implies that the CSP violated requirements for the identity management domain IM.

Similarly, the remaining services are evaluated using their attributes. Since $IS1.1$ metric value is represented by numeric as shown in Table 7.2, Equation 7.2 is used to calculate $IS1.1$

impact factor such that:

$$IF_{IS1.1} = \sqrt{\left(1 - \frac{CSP_{IS1.1}}{CSC_{IS1.1}}\right)^2} = \sqrt{\left(1 - \frac{2.4}{6.4}\right)^2} = 0.625$$

This means that the CSP is under-provisioning the customer's requirement for $IS1.1$ and thus violating the requirement. I calculate the impact factors for $IS1.2$, $IS1.3$, and $IS1.4$ in a similar way and aggregate them to calculate the impact factor of the service IF_{IS}.

$$IF_{IS} = IF_{IS1} = \frac{IF_{IS1.1} + IF_{IS1.2} + IF_{IS1.3} + IF_{IS1.4}}{4} = 0.07$$

Similarly, the impact factors of storage SO and network NO are calculated and their impact factors are aggregated to calculate IF_{root} as:

$$\begin{aligned} IF_{Root} &= \frac{I_{IM} + I_{IS} + I_{SO} + I_{NW}}{4} \\ &= \frac{0.11 + 0.07 + 0 + 0.45}{4} = 0.15 \end{aligned}$$

The IF_{root} indicates that CSP has violated requirements and the overall impact of these violations is minimum. Consequently, the state of trust is changed from trusted $(IF_{root} = 0)$ to $(IF_{root} = 0.25)$. Figure 7.5 shows the aggregated impact factors of attributes belonging to a single service. In the figure, the root impact factor depicts the current "Trust State" for the CSP during launching and migrating a VM.

7.4.2 Case II: VM Migration

In Case I, the state of trust of the CSP by evaluating the IF_{root} is assessed. The IF value of 0.15 indicated that the CSP violated requirement(s) and the aggregated impact of these violations was minimal. Hence, the state is changed to $IF_{root} = 0.25$. From this state, I again evaluate the IF_{root} to assess the CSP trust level by considering the migration process. For completeness, the services involved in the VM migration are shown in Figure 7.4. After the user has been authenticated, the provisioning service starts a new instance of the VM and provides details of the instance to the migration service. The role of the migration service is to migrate data from the old VM to the new VM. Therefore, the migration service accesses the old VM and transfers data over the network to the new instance of the VM as shown in transitions 4-6 of the figure.

I assess services $IM1.1$ to check if requirements are violated by the CSP during the VM migration process. Equation 7.3 is used to calculate the symmetric difference between $IM1.1$

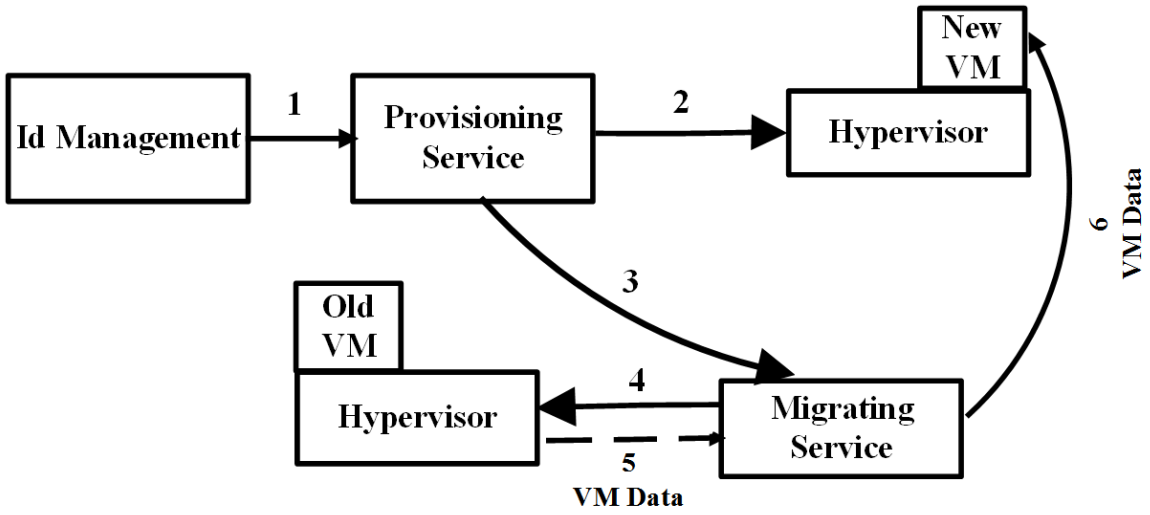

Fig. 7.4 Services and their communication in migrating a VM

values provided by the CSP and requested by the CSC so that:

$$CSP_{IM1.1} - CSC_{IM1.1} = CSC_{IM1.1} - CSP_{IM1.1} = \{\emptyset\}$$

The *null* set indicates that there are no violations. Using Equation 7.5, the impact factor of the service can be calculated as:

$$\begin{aligned} IF_{IM1.1} &= 1 - J(CSP_{IM1.1}, CSC_{IM1.1}) \\ &= 1 - \frac{|\, CSP_{IM1.1} \cap CSC_{IM1.1} \,|}{|\, CSP_{IM1.1} \cup CSC_{IM1.1} \,|} = 1 - 1 = 0 \end{aligned}$$

Similarly, both $IM2.1$ and $IM2.2$ impact factors are calculated. Using Equation 7.3, I premeditate the symmetric difference for both $IM2.1$ and $IM2.2$ such that:

$$IF_{IM2.1} = 1 - 1 = 0$$
$$IF_{IM2.2} = 1 - 1 = 0$$

This means that the CSP is offering $IM2.2$ and $IM2.1$ as agreed with the customer. Thus the $IM2$ impact factor is then premeditated by aggregating $IF_{IM2.1}$ and $IF_{IM2.2}$ using Equation 7.5

such that:

$$IF_{IM2} = \frac{IF_{IM2.1} + IF_{IM2.2}}{2} = 0$$

Subsequently, both impact factors IF_{IM1} and IF_{IM2} are aggregated to get IF_{IM} such that:

$$IF_{IM} = \frac{IF_{IM1} + IF_{IM2}}{2} = 0$$

Similar to case I, the values of IS, SO, and NW are calculated. Finally, the IF_{root} impact factor is calculated such that:

$$IF_{Root} = \frac{IF_{IM} + IF_{IS} + IF_{SO} + IF_{NW}}{4} = \frac{0 + 0.25 + 0 + 0}{4} = 0.06$$

As the current trust value is evaluated to be 0.06, this indicates that the state of the trust should be changed to the trusted state ($IF = 0$) state (cf., Figure 7.3). Thus the proposed methodology enables the customers to (a) assess the state of trust of the CSP during the course of operations and (b) also provide traceable justification for trusting the CSP.

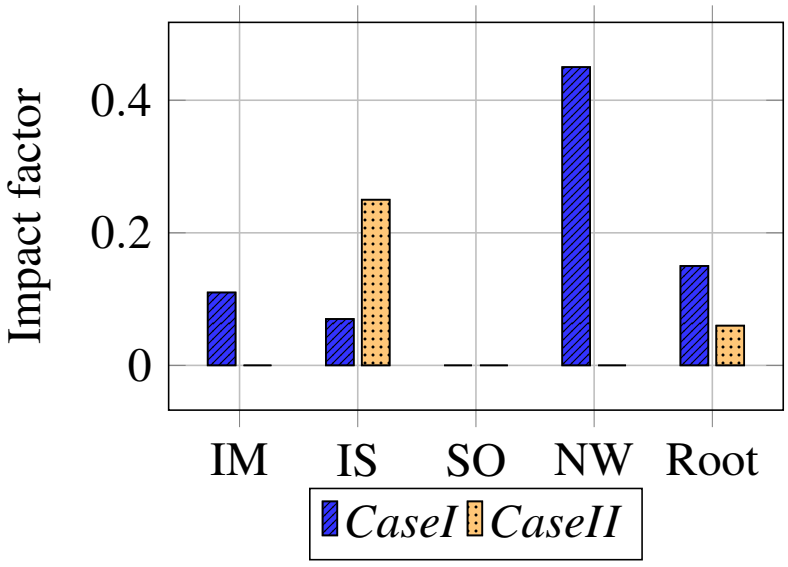

Fig. 7.5 Service and root Impact factors of Cloud IaaS

7.5 Conclusion

Cloud service providers expect their customers to "trust" the CSP services offered to them but not every customer is willing to grant this trust without justification. It should be possible for a customer to establish that their SLA requirements are fulfilled by the CSP's provisioning of the specified services. With this aim, a methodology is proposed that enables the customer to

identify the CSP violation of requirements over the life of the service. For violation detection, I compare and validate each SLA attribute with respect to the customer requirements. In case of a requirement violation, I calculate the severity of the violation by calculating an impact factor. The impact factor determines the degree of deviation of the provided value from the required value and consequently dictates the change in the level/state of trust in the CSP. The customer can periodically apply the proposed methodology to assess the behavior of the CSP over the life of the service. The application of the proposed trust assessment methodology was established via two actual use cases of CSPs offering IaaS.

Chapter 8

Conclusions and Future Work

Cloud computing is a technology paradigm to facilitate the delivery of services and computing resources over the Internet that can be rapidly and flexibly provisioned to users. Due to its inherent elasticity, the Cloud can give the illusion of containing an infinite amount of resources shared among the users. Moreover, the *pay-as-you-go* model offered by the Cloud providers offers economic benefits to the organizations. Therefore, many organizations have migrated their businesses to the Cloud or adopted a “Cloud first" strategy. The proliferation of the Cloud is expected to increase over the next years. However, security remains a significant concern for both individual Cloud users and organizations.

To address these security concerns, threat analysis is advocated to examine a system’s exposure to threats. Microsoft pioneered STRIDE, a model to classify threats to a system into different categories. The model is useful in investigating architectural flaws and is an attacker-centric model. On the other hand, many effective threat analysis approaches exist that explore the potential threats at the system level or at the application level. Furthermore, graphical security models such as attack graphs/trees have also been applied to the Cloud to understand the attack paths in the Cloud. These techniques are either technology-focused or assume that the interconnection among the assets (e.g., service, data, resources, etc.) is static. Thus, these schemes reveal threats pertinent to the specific technology. However, the Cloud is a complex and dynamic environment entailing both the physical and virtual resources that can migrate from one physical host to another. Therefore, assessing the security of the Cloud requires analyzing threats across the operational stack to evaluate the security of the Cloud holistically. Moreover, incorporating users’ requirements and potential variants of the threats enables the threat analysis process to be comprehensive.

8.1 Conclusions

This thesis investigated assessing the security of the Cloud holistically: First, a multi-layer functional model of the Cloud is developed. Second, the model is translated to an information flow model that is agnostic to underpinning technologies. From the perspective of threat analysis across the abstraction, the information flow model captures the services interaction in fundamental operations of the Cloud, e.g., the instantiation of a virtual machine. Third, the interplay between the users' requirements, Cloud services, and threats is investigated to identify the relationship between the requirements and the threats in the Cloud. Additionally, variants of the threats are also included to assess their potential in compromising the Cloud. Finally, from the perspective of Cloud users, the goal was to enable users to validate if their requirements are provisioned according to the signed contract, i.e., the service level agreement. In summary, this thesis investigated the following research questions and proposed the contributions detailed under the respective research question.

Research Question 1 (RQ1): How can Cloud Service Providers (CSPs) examine the security of the Cloud at different abstractions of the operations?

A broad spectrum of issues impeded the adoption of Cloud computing, with security arguably among the most significant. To address the security issues, threat analysis is often used to determine the threats targeting the Cloud. However, the lack of transparency in the architecture and the complexity of the Cloud make threat analysis for the Cloud a challenging task. Further, the Cloud comprises various technologies/services necessary for the proper functioning of the Cloud, and numerous attack surfaces exist across these technologies. Thus, threat analysis approaches for Cloud typically target a specific technology to reveal threats pertinent to the technology. The effectiveness of these techniques beyond the targeted technology is limited and thus, making these techniques technology-dependent. Furthermore, the operational behavior of the technology is often overlooked in these techniques. For instance, the interaction of the technology with other services/technologies in the Cloud. Thus, this question investigates assessing the security across the entire operational stack of the Cloud to enable providers to assess the threats at different abstractions of the functionality stack.

Contribution 1 (C1): A Cloud model capable of representing the fundamental operations of a Cloud in a technology agnostic manner.

The security assessment of the Cloud at different abstractions strongly depends on the model of the Cloud used. Typically, modeling the Cloud implies modeling the running application for performance optimization, and this abstraction of the Cloud operation does not contemplate the Cloud's internal functionality. However, to assess the security of the Cloud at different abstractions, the model needs to be comprehensive to cover the entire spectrum of the functionality. This reveals the potential of a threat's propagation from the internal Cloud functionality to the application layer. For example, a denial of service attack targeting the Cloud might consequently limit an application's availability, although the application is not directly attacked. Thus, to comprehend the threats facing the Cloud, representing the operations of the Cloud is essential to determine the presence of attack surfaces at an attacker's disposal.

Therefore, the development of the Cloud model is the thesis's first contribution, detailed in Chapters 3 and 4. Chapter 3 detailed the multi-layer functional model of the Cloud and the interaction of the services abstractly. The sequence of operations in launching a virtual machine is also explained in the Chapter. Chapter 4 translated the functional Cloud model into a technology-agnostic information flow model applicable to a spectrum of Cloud offerings. The information flow model is based on Petri nets that supports dynamically capturing the services interaction and time-driven information flow across the services. The states in Petri nets are distributed, enabling Cloud providers to examine the security of the Cloud at different. The Cloud's functionality in Petri nets is represented through conditional transitions triggered after their respective preconditions are satisfied. Thus, both Chapters 3 and 4 contributed toward (a) examining the benign behavior of the Cloud in the absence of threats and (b) analyzing the interactions amongst the services belonging to different layers in the Cloud.

Research Question 2 (RQ2): How can the effects of a threat in a service on other interconnected services be identified?

The significant factor that differentiates the Cloud environment from the traditional IT environment is that the Cloud is a dynamic environment. It means that new interconnection occurs at run-time, e.g., when a user requests a new resource or when a resource migrates among the physical hosts. In addition, due to the coupling of various technologies/services in the Cloud operation, the number of attacks targeting multi-layer/multi-technologies has increased, highlighting the need for multi-layer and dynamic threat analysis approaches applicable to the Cloud environments. Specifically, this question investigated the attack paths at an attacker's disposal, considering the holistic view of the Cloud operations. Furthermore,

this question investigated the visibility of the Cloud operations with respect to varied attacker profiles.

Contribution 2 (C2): A path-illustrative approach to profile threats, analyze their impact on targeted services and the propagation of threats across the multiple layers of the Cloud.

In Chapter 5, I presented threat analysis approaches Threatpro, and AttackDive, which explore attack paths that lead to violating the security requirements. The Chapter provided details on the addition of threats to the Cloud model. i.e., the necessary preconditions of a threat. The threats were also defined using Petri nets, and including threats at different layers/services facilitates examining the cause-effect relationship between a threat and a service. Consequently, it enables analyzing the propagation of the threats in the Cloud.

Both ThreatPro and AttackDive simulated (a) the benign Cloud functioning to baseline the operational behavior of the Cloud, (b) the impact of a threat on a particular service/technology, and (c) the effect of multiple threats across the services belonging to different layers of the Cloud. Furthermore, these techniques are capable of performing what-if analysis using the vulnerabilities reported in the national vulnerability database to determine the corresponding attack scenarios. The what-if analysis can investigate paths that an attacker could use to undermine a security requirement. Additionally, the Chapter detailed different operational views of the Cloud with respect to different attacker profiles. For example, insider attackers have a different view of the operations than outsider attackers.

Research Question 3 (RQ3): How can the Cloud service providers gather and organize knowledge concerning the interplay between security threats and a user's requirements?

The disclosed vulnerabilities require significant effort from system administrators to assess each vulnerability's potential to compromise their system. Therefore, this research question investigated the inclusion of requirements into the threat analysis process to prioritize the vulnerabilities pertinent to the security requirements of a user. The requirement-based threat analysis facilitates Cloud providers to comprehend the relationship between security requirements, threats, and the targeted services. Moreover, this question also examined the limitation of the current vulnerabilities classification, which is based on using vulnerability consequence as a criterion to classify them into general categories such as denial of service, etc. Thus, this question investigated the extent to which different vulnerabilities are related to each other beyond their common consequence.

Contribution 3 (C3): Development of requirement-based threat analysis to prioritize threats according to the user's requirements.

Chapter 6 also presented the application of clustering to reveal common patterns between different vulnerabilities. For instance, the analysis showed that many vulnerabilities correlate with each other regarding attack mechanisms. Thus, adding these potential correlations (or variants) in the threat analysis process is helpful to Cloud providers in the organization of the knowledge associated with vulnerabilities.

Moreover, Chapter 6 presented the relationship among different actors involved in the Cloud ecosystem using the Design Structure Matrix (DSM). The advantage of using a DSM is that it helps visualize the relationship between requirement specifications, the Cloud model, and vulnerabilities targeting the services. The approach presented in the Chapter is adaptable to assess the security of the Cloud from varied actor perspectives. This is achieved by applying different algorithms to the DSM, e.g., DSM can be re-arranged to identify the most critical/influential.

Research Question 4 (RQ4): How can Cloud users assess that their requirements (qualitative and quantitative) are satisfied by the Cloud Service Provider (CSP)?

The perceived lack of trust and the security concerns affiliated with the Cloud hinder a wider adoption of the Cloud, specifically to critical applications. To remedy this, Cloud Service Providers (CSPs) typically set up Service Level Agreements (SLAs) that are legally binding between CSPs and Cloud Service Customers (CSCs). SLAs list the attributes the CSP must provide in compliance with the customer's requirements. While SLAs are promising as a concept, the inadequacy of validating SLAs during the operation of the Cloud limits the customer's capability to evaluate if the offered services meet their requirements. Thus, this question investigated the customer-centric mechanism of validating an SLA to evaluate any deviations from the SLA. In case of an SLA violation, a user is entitled to compensation.

Contribution 4 (C4): A customer-centric approach to assess the fulfillment of security requirements by the CSP.

Chapter 7 presented a customer-centric approach to validate the SLA signed with the CSP. The purpose of the approach is to verify that the attributes promised by the Cloud service providers are fulfilled during the Cloud operation. The approach proposed an evaluation mechanism for both qualitative and quantitative attributes. Moreover, the user can specify

weights to different attributes to rank the violation of these attributes accordingly. In case a violation occurs in the attribute provisioning, the severity of the violation is evaluated by calculating an impact factor of the violation. The impact factor establishes the degree of deviation from the required value of the provided attribute. Consequently, the higher the degree of deviation, the more severe the violation. Furthermore, the impact factor is used to determine the change in the level/state of trust in the CSP. The CSP can be in varied trust states depending on the severity level of the violations. A periodic application of the proposed methodology leads to assessing the behavior of the CSP over the life of the service. The trust state of a CSP during the launch of a VM and migration of a VM is used to determine any deviation between the provisioning of services from the CSP and the requirements of the user.

The Cloud proliferation is projected to grow over the following years, but the security challenges in the Cloud are still a significant hindrance to its adoption. This thesis has confirmed that there exists a plethora of attack surfaces across the operational stack of the Cloud that an attacker can use. These attack surfaces are specific to the Cloud; therefore, threat analysis approaches that are designed explicitly for the Cloud are required. Moreover, the thesis investigated single-stage and multi-stage attack scenarios to examine the security assessment of the Cloud holistically and independently of the underlying technologies. Addressing the security issues in the Cloud will expedite its adoption to industries still reluctant to migrate their operations to the Cloud, e.g., the health care industry. In the following, a potential future direction is presented that compliments the thesis by mitigating the threats proactively.

8.2 Future Work: Proactive Threats Mitigation Techniques

This thesis has investigated the security assessment of the Cloud by exploring threats across the operational stack of the Cloud and the interplay between threats, their variants, and the user's security requirements. On the other hand, mitigating threats in the theses was largely unexplored. Therefore, a natural extension of the thesis is to propose proactive methodologies for mitigating threats against the Cloud.

Moving Target Defense (MTD) [145] techniques are advocated as a promising proactive approach to improve the security of a system. The basic premise behind the MTD techniques is that it is impossible to secure the system completely, so introducing uncertainty and complexity for attackers might thwart the attacks. This is eventually achieved through changing the attack surfaces that reduce the window of opportunity and/or increase the cost of an attack. Multiple MTD techniques can be broadly organized into categories following a taxonomy proposed in [146]. These are dynamic run-time environments, dynamic platforms, dynamic software, dynamic networks, and dynamic data. The MTDs focusing on a dynamic run-time environment change the execution environment presented to the application [147, 148]. In contrast, MTDs for dynamic platforms disrupt the attack that typically relies on specific platform characteristics (e.g., a specific OS version) by replacing the platform [149]. The MTDs belonging to dynamic software operate similar to the dynamic platform MTDs, but their focus is on the software/application layer [150, 151] instead. At the network layer, MTDs techniques target modifying the network characteristics (e.g., IP/MAC addresses) [152–154] while changing the data format used in the application is termed as dynamic data MTDs [155].

The thesis has proposed a technology-agnostic Cloud model which can be used as a basis for applying the MTDs across the different layers of the Cloud as deploying an individual MTD, e.g., OS diversity at the VM level or changing hypervisor might not be sufficient to limit multi-stage attacks. Therefore, a combination of MTDs at different layers is required to maximize the security gains of the MTD techniques. Moreover, it is essential to figure out the placement and triggering of an MTD technique depending on the potential actions of an attacker. Thus, the aim of the future work is to explore the combination of different MTDs at different layers to maximize the security gains offered by the MTDs. Furthermore, I will propose a set of actions for MTD deployment (wait, deploy and reset) to determine the optimal MTD strategy coupled with selecting different MTDs across various layers. A quantification mechanism and metrics (attack path disruption and attack surface reduction) are proposed to evaluate the effectiveness of the MTDs against the attacks in the Cloud.

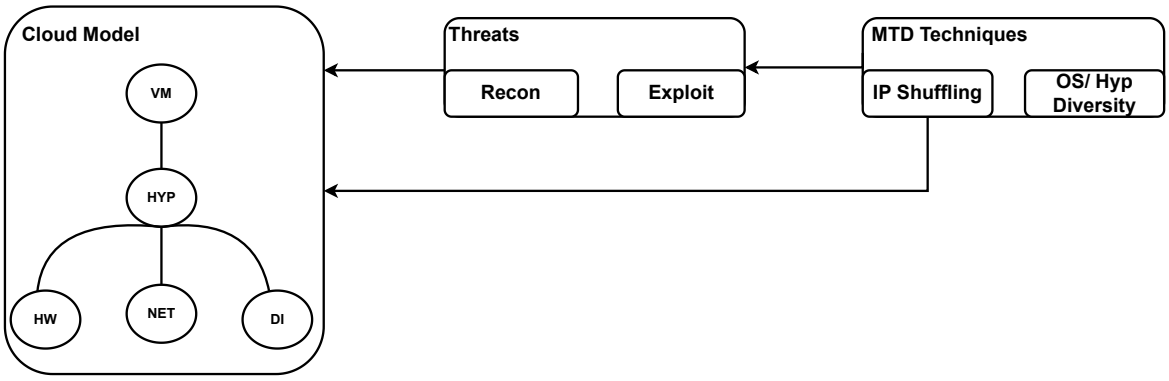

Fig. 8.1 Proposed moving target defense approach

Moving Target Defense Framework

The MTD framework is shown in Figure 8.1. There are three layers in our framework. The first is the set of MTD techniques such as IP shuffling, OS diversity, etc. The second layer is the threats that MTDs have an impact on. In order to attack a system, an attacker has to know the weaknesses of the system and how to exploit them. Therefore, threats have primarily two main blocks. Preconditions encompass the knowledge about the weaknesses of a system, and exploitation is the process of exploiting these weaknesses. It might be noted here that an MTD can affect any or both of the blocks. Finally. the third layer is the system under investigation. In my case, I show a subset of the Cloud operations since the goal is to determine the effectiveness of MTDs (or combination of MTDs) considering a holistic view of the Cloud.

Now I have defined the Cloud system and the threats, the last layer introduces the MTDs. Therefore I can now define an overall MTD framework as follows: An MTD quantification framework can be defined as a 5-tuple $(M, T, C, R_{mt}, R_{mc}, R_{tc})$ where

- M is a finite set of MTDs.
- T is a finite set of threats.
- S is a finite set of services in the Cloud.
- $R_{mt} \subseteq M \times T$ is the relationship between MTDs and the threats. It determines the corresponding impact of MTDs on threats.
- $R_{ms} \subseteq M \times S$ defines the relationship between MTDs and services in the Cloud. It signifies the applicability of an MTD on the service. For instance, dynamic application data is only applicable to the application running on the VM.

- $R_{ts} \subseteq T \times S$ defines the relationship between threats and the targeted services. It identifies the threats that could potentially target a specific service.

Utilizing these, I get a tripartite graph. The relationship among different layers is defined in the following:

Defining the threat to service relationship: A service can be compromised by exploiting vulnerabilities that target the service. A single or combination of vulnerabilities can compromise a service. Therefore, the probability of successfully compromising a service can be calculated using Equation (8.1). It assumes that vulnerabilities are independent of each other:

$$Pr(X_t = s) = \begin{cases} Pr(T_a \cup T_b) & \exists T_a \in T : (T_a, s) \vee \\ & \exists T_b \in T : (T_b, s) \\ 0 & \text{otherwise} \end{cases} \tag{8.1}$$

Satisfying a threat's precondition: In order to compromise a service, a vulnerability has to be exploited. However, there could exist preconditions that need to be satisfied to exploit the specific vulnerability. Therefore, the $P(T_a)$ can be further broken down into the satisfaction of each individual precondition. This can be done by simply calculating the likelihood of satisfying different preconditions of the vulnerability. This can be calculated as:

$$P(T_t) = \max(\prod_{i=0}^{n} P(Precondition_t)) \tag{8.2}$$

Effectiveness of MTDs on threats: Finally, the relationship between MTDs, threats, and the services is defined. It is assumed that in case an MTD is not deployed, an attacker can successfully exploit the system, i.e., the probability of the attack success is 1. Thus, the effectiveness of a single instance of MTD on a threat targeting a service can be calculated as:

$$E(M_m, T_t, S_s) = \begin{cases} 1 & \exists (T_t, S_s) \in R_{ts} \wedge \nexists (M_m, T_t) \in R_{mt} \vee \\ & \nexists (M_m, S_s) \in R_{ms} \\ 0 & \text{otherwise} \end{cases} \tag{8.3}$$

Equation (8.3) dictates that an MTD is not effective for a given threat on a service. In other words, this also means that an MTD has not been deployed, and the attacker's success is one as no mitigation strategy has been deployed for the threat.

The framework can be extended to include the effectiveness of each MTD and evaluate the aggregated effectiveness of the MTDs across the Cloud. This results in exploring the optimal place for MTD deployment for single- and multi-stage attack scenarios. However, it is evident that the relationship between the layers is many-to-many and thus, making the exploration of an optimal solution challenging. Furthermore, the cost function could be added as an alternative criterion to tweak the optimal solution accordingly. Therefore, the future objective of my work is to use the Cloud model as a basis to find the optimal solution for the deployment of the MTD that provides maximum security gains considering the holistic view of the system.

References

[1] Peter Mell, Tim Grance, et al. The nist definition of cloud computing. 2011.

[2] Amazon. Amazon web services, 2022.

[3] Google. Google cloud services, 2022.

[4] Microsoft. Microsoft azure, 2022.

[5] Simon Sharwood. Cloud a three-player market dominated by aws, google, microsoft, 2022.

[6] Dropbox. Keep life organized and work moving—all in one place, 2022.

[7] Github. Where the world builds software, 2022.

[8] Ron Miller. The cloud infrastructure market hit $129b in 2020, 2021.

[9] Louis Columbus. Public cloud soaring to $331b by 2022 according to gartner, 2019.

[10] Sean Marston, Zhi Li, Subhajyoti Bandyopadhyay, Juheng Zhang, and Anand Ghalsasi. Cloud computing—the business perspective. *Decision support systems*, 51(1):176–189, 2011.

[11] Abhijit Dubey and Dilip Wagle. Delivering software as a service. *The McKinsey Quarterly*, 6(2007):2007, 2007.

[12] Eric Keller and Jennifer Rexford. The" platform as a service" model for networking. *INM/WREN*, 10:95–108, 2010.

[13] Sushil Bhardwaj, Leena Jain, and Sandeep Jain. Cloud computing: A study of infrastructure as a service (iaas). *International Journal of engineering and information Technology*, 2(1):60–63, 2010.

[14] Kui Ren, Cong Wang, and Qian Wang. Security challenges for the public cloud. *IEEE Internet computing*, 16(1):69–73, 2012.

[15] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. Flush+ flush: a fast and stealthy cache attack. In *Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 279–299, 7-8 Jul, San Sebastián, Spain, 2016. Springer.

[16] Wayne A Jansen, Tim Grance, et al. Guidelines on security and privacy in public cloud computing. 2011.

[17] Flexera. State of the cloud, 2021.

[18] Check Point. Top trends in cloud security, 2020.

[19] Richard Speed. Github saved plaintext passwords of npm users in log files, post mortem reveals, 2022.

[20] Jon Porter. Amazon mitigated the largest ddos attack ever recorded, 2020.

[21] Bruce Lynch. 5 high-profile ddos attacks that should chill you to the bone, 2021.

[22] Samuel King and Peter M Chen. Subvirt: Implementing malware with virtual machines. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 14–327, Oakland, USA, 2006.

[23] Pratyusa Manadhata and Jeannette Wing. An attack surface metric. *IEEE Transactions on Software Engineering*, 37(3):371–386, 2011.

[24] Nils Gruschka and Meiko Jensen. Attack surfaces: A Taxonomy for Attacks on Cloud Services. In *Proceedings of the International Conference on Cloud Computing*, pages 276–279, 05-10 Jul, 5–10 July 2010. IEEE.

[25] Fang Liu, Jin Tong, Jian Mao, Robert Bohn, John Messina, Lee Badger, Dawn Leaf, et al. Nist cloud computing reference architecture. *NIST special publication*, 500(2011):1–28, 2011.

[26] M. Marlinspike. Null prefix attacks against ssl/tls certificates. *Thoughtcrime.org*, 2009.

[27] Jeanna Matthews, Eli Dow, Todd Deshane, Wenjin Hu, Jeremy Bongio, Patrick Wilbur, and Brendan Johnson. *Running Xen: a hands-on guide to the art of virtualization*. Prentice Hall, 2008.

[28] Jerry Honeycutt. Microsoft virtual personal computer. *Microsoft Technical Overview*, 2003.

[29] Forbes Guthrie, Scott Lowe, and Kendrick Coleman. *VMware vSphere design*. John Wiley & Sons, 2013.

[30] NIST. National vulnerability database, 2022.

[31] Bruce Potter. Microsoft sdl threat modelling tool. *Network Security*, 2009(1):15–18, 2009.

[32] Adam Shostack. Experiences threat modeling at microsoft. *MODSEC@ MoDELS*, 2008:35, 2008.

[33] Hesham Abusaimeh. Security attacks in cloud computing and corresponding defending mechanisms. In *International Journal of Advanced Trends in Computer Science and Engineering*, volume 9, pages 4141–4148, 2020.

[34] Hsin-Yi Tsai, Melanie Siebenhaar, Andre Miede, Yulun Huang, and Ralf Steinmetz. Threat as a service?: Virtualization's impact on cloud security. *IT professional*, 14(1):32–37, 2011.

[35] Daniele Sgandurra and Emil Lupu. Evolution of Attacks, Threat Models, and Solutions for Virtualized Systems. *In ACM Computing Surveys*, 48:1–38, 2016.

[36] Limin Wang, Ziyuan Zhu, Zhanpeng Wang, and Dan Meng. Colored Petri net Based Cache Side Channel Vulnerability Evaluation. *IEEE Access*, 7:169825–169843, 2019.

[37] Ronald Ritchey and Paul Ammann. Using model checking to analyze network vulnerabilities. In *Proceedings of the Symposium on Security and Privacy*, pages 156–165, Berkeley, CA, USA, 14–17 May 2000. IEEE.

[38] Devdatta Akhawe, Adam Barth, Peifung Lam, John Mitchell, and Dawn Song. Towards a formal foundation of web security. In *Proceedings of the IEEE Computer Security Foundations Symposium*, pages 290–304, 17–19 July, Edinburgh, UK, 17–19 July 2010.

[39] Kennedy Torkura, Muhammad Sukmana, Michael Meinig, Anne Kayem, Feng Cheng, Hendrik Graupner, and Christoph Meinel. Securing Cloud Storage Brokerage Systems Through Threat Models. In *Proceedings of the International Conference on Advanced Information Networking and Applications*, pages 759–768, Krakow, Poland, 16–18 May 2018. IEEE.

[40] Nawaf Alhebaishi, Lingyu Wang, Sushil Jajodia, and Anoop Singhal. Threat Modeling for Cloud Data Center Infrastructures. In *Proceedings of the International Symposium on Foundations and Practice of Security*, pages 302–319, 24–25 October, Québec, Canada, 2016. Springer.

[41] Armstrong Nhlabatsi, Jin Hong, DongSeong Kim, Rachael Fernandez, Alaa Hussein, Noora Fetais, and Khaled Khan. Threat-specific security risk evaluation in the cloud. *IEEE Transactions on Cloud Computing*, 9:1–13, 2018.

[42] Salman Manzoor, Jesus Luna, and Neeraj Suri. Attackdive: Diving deep into the cloud ecosystem to explore attack surfaces. In *Proceedings of the IEEE International Conference on Services Computing*, pages 499–502, 25-30 Jun, Honolulu, USA, 2017. IEEE.

[43] Salman Manzoor, Ahmed Taha, and Neeraj Suri. Trust validation of cloud iaas: A customer-centric approach. In *Proceedings of the IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pages 97–104, 23-26 Aug, Tianjin, China, 2016. IEEE.

[44] Salman Manzoor, Antonios Gouglidis, Bradbury Matthew, and Neeraj Suri. Threatpro: Multi-layer threat analysis in the cloud". In *ACM Transactions on Privacy and Security (Under Review)*, 2022.

[45] Salman Manzoor, Tsvetoslava Vateva-Gurova, Rubén Trapero, and Neeraj Suri. Threat modeling the cloud: An ontology based approach. In *Proceedings of the International Workshop on Information and Operational Technology Security Systems*, pages 61–72, 13 Sept, Crete, Greece, 2018. Springer.

[46] Salman Manzoor, Daniel Prince, and Neeraj Suri. Ontologies for vulnerability terrain mapping and attack reasoning. *International Conference on Network and System Security (Under Review)*, 2022.

[47] Sunilkumar Manvi and Gopal Shyam. Resource management for infrastructure as a service (iaas) in cloud computing: A survey. *International Journal of Network and Computer Applications*, 41:424–440, 2014.

[48] Andrew Younge, Gregor Laszewski, Lizhe Wang, Sonia Lopez-Alarcon, and Warren Carithers. Efficient Resource Management for Cloud Computing Environments. In *Proceedings of the International Conference on Green Computing*, pages 357–364, Chicago, IL, USA, 2010. IEEE.

[49] Umesh Deshpande, Danny Chan, Ten-Young Guh, James Edouard, Kartik Gopalan, and Nilton Bila. Agile live migration of virtual machines. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium*, pages 1061–1070, 23-27 May, Chicago, USA, 2016. IEEE.

[50] Waltenegus Dargie. Estimation of the cost of vm migration. In *Proceedings of the IEEE International Conference on Computer Communication and Networks*, pages 1–8, 04-07 Aug, Shanghai, China, 2014. IEEE.

[51] Tiago Gama Rodrigues, Katsuya Suto, Hiroki Nishiyama, and Nei Kato. Hybrid method for minimizing service delay in edge cloud computing through vm migration and transmission power control. *In IEEE Transactions on Computers*, 66(5):810–819, 2016.

[52] Shawn Hernan, Scott Lambert, Tomasz Ostwald, and Adam Shostack. Uncover security design flaws using the STRIDE approach. *MSDN Magazine*, Nov. 2006.

[53] Sven Bugiel, Stefan Nürnberger, Thomas Pöppelmann, Ahmad-Reza Sadeghi, and Thomas Schneider. Amazonia: When elasticity snaps back. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 389–400, 17-21 Oct, Chicago, USA, 2011. ACM.

[54] Akond Rahman, Chris Parnin, and Laurie Williams. The seven sins: Security smells in infrastructure as code scripts. In *Proceedings of the IEEE/ACM International Conference on Software Engineering*, pages 164–175, 25-31 May, Montreal, Canada, 2019. IEEE/ACM.

[55] Akond Rahman, Rayhanur Rahman, Chris Parnin, and Laurie Williams. Security smells in ansible and chef scripts: A replication study. *ACM Transactions on Software Engineering and Methodology*, 30(1):1–31, 2021.

[56] Chun-Ting Huang, Lei Huang, Zhongyuan Qin, Hang Yuan, Lan Zhou, Vijay Varadharajan, and Jay Kuo. Survey on securing data storage in the cloud. *APSIPA Transactions on Signal and Information Processing*, 3, 2014.

[57] Sugumaran, Bala Murugan, and Kamalraj. An architecture for data security in cloud computing. In *Proceedings of the World Congress on Computing and Communication Technologies*, pages 252–255, 27 February - 01 March, Trichirappalli, India, 2014. IEEE.

[58] Michele De Donno, Alberto Giaretta, Nicola Dragoni, Antonio Bucchiarone, and Manuel Mazzara. Cyber-storms come from clouds: Security of cloud computing in the iot era. *Future Internet*, 11(6):127, 2019.

[59] Haifa Mohamed Al Nasseri. *Detecting Cloud Virtual Network Isolation Security for Data Leakage*. PhD thesis, University of St Andrews, 2019.

[60] Fabiao Miao, Liming Wang, and Zailong Wu. A vm placement based approach to proactively mitigate co-resident attacks in cloud. In *Proceedings of the IEEE Symposium on Computers and Communications*, pages 00285–00291. IEEE, 2018.

[61] Mohamed Elshabka, Hanan Hassan, Walaa Sheta, and Hany Harb. Security-aware dynamic vm consolidation. *Egyptian Informatics Journal*, 22(3):277–284, 2021.

[62] Gordon Fyodor Lyon. *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure. Com LLC (US), 2008.

[63] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 199–212, 9-13 Nov, Chicago, USA, 2009.

[64] Diego Perez-Botero, Jakub Szefer, and Ruby Lee. Characterizing hypervisor vulnerabilities in cloud computing servers. In *Proceedings of the ACM International Workshop on Security in Cloud Computing*, pages 3–10, 8 May, Hangzhou, China, 2013. ACM.

[65] Ori Or-Meir, Nir Nissim, Yuval Elovici, and Lior Rokach. Dynamic malware analysis in the modern era—a state of the art survey. *In ACM Computing Surveys*, 52(5):1–48, 2019.

[66] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, et al. Spectre attacks: Exploiting speculative execution. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 1–19, 19-23 May, CA, USA, 2019. IEEE.

[67] Yuval Yarom and Katrina Falkner. {FLUSH+ RELOAD}: A high resolution, low noise, l3 cache {Side-Channel} attack. In *USENIX Security Symposium*, pages 719–732, San Diego, CA, 2014. USENIX Association.

[68] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby Lee. Last-level cache side-channel attacks are practical. In *Proceedings of the IEEE symposium on Security and Privacy*, pages 605–622, 17-21 May, CA, USA, 2015. IEEE.

[69] Hsin-Yi Tsai, Melanie Siebenhaar, Andre Miede, Yulun Huang, and Ralf Steinmetz. Threat as a service?: Virtualization's impact on cloud security. *IT professional*, 14(1):32–37, 2011.

[70] Thomas Ball. The concept of dynamic analysis. In *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 216–234, 6-10 Sept, Toulouse, France, 1999. ACM.

[71] Athanasios Naskos, Anastasios Gounaris, Haralambos Mouratidis, and Panagiotis Katsaros. Online analysis of security risks in elastic cloud applications. *IEEE Cloud Computing*, 3:26–33, 2016.

[72] Adrian Duncan, Sadie Creese, Michael Goldsmith, and Jamie Quinton. Cloud computing: Insider attacks on virtual machines during migration. In *Proceedings of the IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pages 493–500, 16-18 Jul, Melbourne, Australia, 2013. IEEE.

[73] Hua Deng, Qianhong Wu, Bo Qin, Jian Mao, Xiao Liu, Lei Zhang, and Wenchang Shi. Who is touching my cloud. In *Proceedings of the European Symposium on Research in Computer Security*, pages 362–379, 7-11 Sept, Wroclaw, Poland, 2014. Springer.

[74] Yujue Wang, Qianhong Wu, Duncan Wong, Bo Qin, Sherman Chow, Zhen Liu, and Xiao Tan. Securely outsourcing exponentiations with single untrusted program for cloud storage. In *Proceedings of the European Symposium on Research in Computer Security*, pages 326–343, 7-11 Sept, Wroclaw, Poland, 2014. Springer.

[75] Jon Oberheide, Evan Cooke, and Farnam Jahanian. Empirical exploitation of live virtual machine migration. In *Proc. of BlackHat DC convention*, pages 1–6. Citeseer, 2008.

[76] Jia-Rung Yeh, Hsu-Chun Hsiao, and Ai-Chun Pang. Migrant attack: A multi-resource dos attack on cloud virtual machine migration schemes. In *2016 11th Asia Joint Conference on Information Security (AsiaJCIS)*, pages 92–99, 2016.

[77] Atif Saeed, Peter Garraghan, Barnaby Craggs, Dirk van der Linden, Awais Rashid, and Syed Asad Hussain. A cross-virtual machine network channel attack via mirroring and tap impersonation. In *Proceedings of the IEEE International Conference on Cloud Computing*, pages 606–613, 02-07 Jul, CA, USA, 2018. IEEE.

[78] Atif Saeed, Peter Garraghan, and Asad Hussain. Cross-vm network channel attacks and countermeasures within cloud computing environments. *IEEE Transactions on Dependable and Secure Computing*, 19(3):1783–1794, 2022.

[79] Xen. Xen server architecture, 2022.

[80] L Kurth. Xenserver. org and the xen project. *Xen Project*, 2003.

[81] Ryan Roemer, Erik Buchanan, Hovav Shacham, and Stefan Savage. Return-oriented programming: Systems, languages, and applications. *ACM Transactions on Information and System Security*, 15(1):1–34, 2012.

[82] Marco Prandini and Marco Ramilli. Return-oriented programming. *IEEE Security & Privacy*, 10(6):84–87, 2012.

[83] Bruce Schneier. Attack trees. *Dr. Dobb's journal*, 24(12):21–29, 1999.

[84] Ping Wang, Wen-Hui Lin, Pu-Tsun Kuo, Hui-Tang Lin, and Tzu Chia Wang. Threat risk analysis for cloud security based on attack-defense trees. In *Proceedings of the International Conference on Computing Technology and Information Management*, pages 106–111, Seoul, South Korea, 2012. IEEE.

[85] Saif Malik, Samee Khan, and Sudarshan Srinivasan. Modeling and analysis of state-of-the-art vm-based cloud management platforms. *IEEE Transaction on Cloud Computing*, 1:50–63, 2013.

[86] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette Wing. Automated generation and analysis of attack graphs. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 273–284, 12-15 May, Berkeley, USA, 2002. IEEE.

[87] Lingyu Wang, Tania Islam, Tao Long, Anoop Singhal, and Sushil Jajodia. An attack graph-based probabilistic security metric. In *Data and Applications Security XXII*, pages 283–296, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[88] Marcel Frigault and Lingyu Wang. Measuring network security using bayesian network-based attack graphs. In *Proceedings of the IEEE International Computer Software and Applications Conference*, pages 698–703, 28 Jul-01 Aug, Turku, Finland, 2008.

[89] Armstrong Nhlabatsi, Khaled Khan, Jin Hong, Dong Kim, Rachael Fernandez, and Noora Fetais. Quantifying Satisfaction of Security Requirements of Cloud Software Systems. *IEEE Transactions on Cloud Computing*, pages 1–18, 2021.

[90] Amartya Sen and Sanjay Madria. Risk assessment in a sensor cloud framework using attack graphs. *IEEE Transactions on Services Computing*, 10:942–955, 2017.

[91] Shareeful Islam, Moussa Ouedraogo, Christos Kalloniatis, Haralambos Mouratidis, and Stefanos Gritzalis. Assurance of security and privacy requirements for cloud deployment models. *IEEE Transactions on Cloud Computing*, 6:387–400, 2018.

[92] Prasad Saripalli and Ben Walters. QUIRC: A Quantitative Impact and Risk Assessment Framework for Cloud Security. In *Proceedings of the International Conference on Cloud Computing*, pages 280–288, 05-10 July, Miami, USA, 2010. IEEE.

[93] Arpan Roy, Dong Seong Kim, and Kishor Trivedi. Attack countermeasure trees (act): Towards unifying the constructs of attack and defense trees. *Security and Communication Networks*, 5(8):929–943, 2012.

[94] Nayot Poolsappasit, Rinku Dewri, and Indrajit Ray. Dynamic security risk management using bayesian attack graphs. *IEEE Transactions on Dependable and Secure Computing*, 9:61–74, 2012.

[95] Dan Gonzales, Jeremy Kaplan, Evan Saltzman, Zev Winkelman, and Dulani Woods. Cloud-trust—a security assessment model for infrastructure as a service (iaas) clouds. *IEEE Transactions on Cloud Computing*, 5:523–536, 2017.

[96] Archana Ganapathi, Yanpei Chen, Armando Fox, Randy Katz, and David Patterson. Statistics-driven workload modeling for the cloud. In *Proceedings of the International Conference on Data Engineering Workshops*, pages 87–92, 1-6 Mar, Long Beach, USA, 1–6 March 2010. IEEE.

[97] Fumio Machida, Ermeson Andrade, Dong Kim, and Kishor Trivedi. Candy: Component-based Availability Modeling Framework for Cloud Service Management Using SysML. In *Proceedings of the International Symposium on Reliable Distributed Systems*, pages 209–218, Madrid, Spain, 4–7 October 2011. IEEE.

[98] Florian Metzger, Tobias Hoßfeld, André Bauer, Samuel Kounev, and Poul Heegaard. Modeling of aggregated iot traffic and its application to an iot cloud. *Proceedings of the IEEE*, 107:679–694, 2019.

[99] Roger Smith. Computing in the cloud. *International journal of research-technology management*, 52:65–68, 2009.

[100] Xin Jin, Qixu Wang, Xiang Li, Xingshu Chen, and Wei Wang. Cloud virtual machine llifecycle security framework based on trusted computing. *Journal of Tsinghua Science and Technology*, 24:520–534, 2019.

[101] Omar Sefraoui, Mohammed Aissaoui, and Mohsine Eleuldj. Openstack: Toward an open-source solution for cloud computing. *International Journal of Computer Applications*, 55(3):38–42, October 2012.

[102] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The Eucalyptus Open-Source Cloud-Computing System. In *Proceedings of the International Symposium on Cluster Computing and the Grid*, pages 124–131, Shanghai, China, 2009. IEEE/ACM.

[103] Ankita Desai, Rachana Oza, Pratik Sharma, and Bhautik Patel. Hypervisor: A survey on concepts and taxonomy. *International Journal of Innovative Technology and Exploring Engineering*, 2:222–225, 2013.

[104] Vijay Varadharajan. Petri net based modelling of information flow security requirements. In *Proceedings of the Computer Security Foundations Workshop*, pages 51–61, Franconia, NH, USA, 1990. IEEE.

[105] Zakaria Benzadri, Faiza Belala, and Chafia Bouanaka. Towards a Formal Model for Cloud Computing. In *Service-Oriented Computing*, pages 381–393. Springer, 2013.

[106] Károly Bósa, Roxana Holom, and Mircea Vleju. *A Formal Model of Client-Cloud Interaction*, pages 83–144. Springer, 2015.

[107] Hamza Sahli, Chafia Bouanaka, and Ahmed Dib. Towards a Formal Model for Cloud Computing Elasticity. In *Proceedings of the International WETICE Conference*, pages 359–364, 20 Oct, Parma, Italy, 23–25 June 2014. IEEE.

[108] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. *Progress on the State Explosion Problem in Model Checking*, pages 176—-194. Springer-Verlag, Berlin, Heidelberg, 2001.

[109] Khodakaram Salimifard and Mike Wright. Petri net-based modelling of workflow systems: An overview. *European Journal of Operational Research*, 134:664–676, 2001.

[110] Antonio Brogi, Andrea Canciani, Jacopo Soldani, and PengWei Wang. *A Petri Net-Based Approach to Model and Analyze the Management of Cloud Applications*, pages 28–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[111] Renée Boubour, Claude Jard, Armen Aghasaryan, Eric Fabre, and Albert Benveniste. A petri net approach to fault detection and diagnosis in distributed systems. In *Proceedings of the IEEE Conference on Decision and Control*, pages 720–725, 12 Dec, San Diego, USA, 12 December 1997. IEEE.

[112] ISO Central Secretary. High-level Petri nets — Part 1: Concepts, Definitions and Graphical notation. Standard ISO/IEC 15909-1:2019, International Organization for Standardization, Geneva, Switzerland, August 2019.

[113] Raymond M Smullyan. *First-order logic*. Courier Corporation, 1995.

[114] Kurt Jensen and Lars Kristensen. *CPN ML Programming*, chapter 3, pages 43–77. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[115] Kurt Jensen, Lars Kristensen, and Lisa Wells. Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. *International Journal on Software Tools for Technology Transfer*, 9:213–254, 2007.

[116] Ping Wang and Christopher Johnson. Cybersecurity incident handling: a case study of the equifax data breach. *Issues in Information Systems*, 19:150–159, 2018.

[117] Stacy Cowley. Equifax to Pay at Least 650 Million in Largest-Ever Data Breach Settlement, 2019.

[118] Cloudfare. Famous DDoS attacks: The largest DDoS attacks of all time, 2021.

[119] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. DDoS in the IoT: Mirai and other Botnets. *Computer*, 50(7):80–84, 2017.

[120] Juraj Somorovsky, Mario Heiderich, Meiko Jensen, Jörg Schwenk, Nils Gruschka, and Luigi Lo Iacono. All your clouds are belong to us: Security analysis of cloud management interfaces. In *Proceedings of the ACM workshop on Cloud Computing Security*, pages 3–14, 21 Oct, Chicago, USA, 2011. ACM.

[121] Su Zhang, Xinwen Zhang, and Xinming Ou. After we knew it: Empirical study and modeling of cost-effectiveness of exploiting prevalent known vulnerabilities across iaas cloud. In *Proceedings of the ACM Symposium on Information, Computer and Communications Security*, pages 317–328, 4-6 Jun, Kyoto, Japan, 2014.

[122] FIRST. Common vulnerability scoring system, 2022.

[123] Luca Allodi and Fabio Massacci. Comparing vulnerability severity and exploits using case-control studies. In *ACM Transactions on Information and System Security*, volume 17, pages 1–20, 2014.

[124] Michael Steinbach, George Karypis, Vipin Kumar, et al. A comparison of document clustering techniques. In *Proceedings of ACM workshop on Knowledge Discovery, Data and Text Mining*, pages 525–526. ACM, 2000.

[125] OpenStack. OpenSack cloud platform. *http://www.open- stack.org/*, 2010.

[126] Mass Lund, Bjornar Solhaug, and Ketil Stolen. Evolution in relation to risk and trust management. *In IEEE Computer*, 43(5):49–55, 2010.

[127] Jesus Luna, Robert Langenberg, and Neeraj Suri. Benchmarking Cloud Security Level Agreements Using Quantitative Policy Trees. In *Proceedings of the ACM Workshop on Cloud Computing Security*, pages 103–112, 19 Oct, North Carolina, USA, 2012.

[128] Ahmed Taha, Ruben Trapero, Jesus Luna, and Neeraj Suri. AHP-Based Quantitative Approach for Assessing and Comparing Cloud Security. *Proceedings of the IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pages 284–291, 2014.

[129] Kumar Garg, Steve Versteeg, and Rajkumar Buyya. A framework for ranking of cloud computing services. *In Future Generation Computer Systems*, 29(4):1012–1023, 2013.

[130] Zia Rehman, Farookh Hussain, and Omar Hussain. Towards multi-criteria cloud service selection. *Proceedings of the IEEE International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 44–48, 2011.

[131] Jesus Luna, Ahmed Taha, Ruben Trapero, and Neeraj Suri. Quantitative reasoning about cloud security using service level agreements. *IEEE Transaction on Cloud Computing*, (99):457–471, 2015.

[132] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. Cloudcmp: Comparing public cloud providers. pages 1–14, 1-3 Nov, Melbourne, Australia, 2010. ACM.

[133] Jane Siegel and Jeff Perdue. Cloud services measures for global use: the service measurement index (smi). In *Proceedings of the Global Conference*, pages 411–415, 24-27 Jul, San Jose, USA, 2012. IEEE.

[134] Shirlei Chaves, Carlos Westphall, and Flavio Lamin. SLA perspective in security management for cloud computing. In *Proceedings of the IEEE International Conference on Networking and Services*, pages 212–217, 07-13 Mar, Cancun, Mexico, 2010.

[135] Ganna Frankova and Artsiom Yautsiukhin. Service and protection level agreements for business processes. *Proceedings of the European Young Researchers Workshop on Service Oriented Computing*, pages 38–43, 2007.

[136] Leanid Krautsevich, Fabio Martinelli, and Artsiom Yautsiukhin. A general method for assessment of security in complex services. pages 153–164, 26-28 Oct, Poznan, Poland, 2011. Springer.

[137] Sheikh Habib, Sebastian Ries, and Max Mühlhäuser. Towards a trust management system for cloud computing. *Proceedings of the IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pages 933–939, 2011.

[138] Shou Wang, Li Zhang, Shuai Wang, and Xiang Qiu. A cloud-based trust model for evaluating quality of web services. *In Computer Science and Technology*, 25(6):1130–1142, 2010.

[139] Irfan Haq, Rehab Alnemr, Adrian Paschke, Erich Schikuta, Harold Boley, and Christoph Meinel. Distributed trust management for validating sla choreographies. In *Proceedings of the International Conference on Grids and Service-oriented Architectures for Service Level Agreements*, pages 45–55, 31 Jul, New York, USA, 2010. Springer.

[140] Developer Guide. Amazon cloudwatch. 2009.

[141] CloudStack. Open source cloud computing, 2013.

[142] Anas Ayad and Uwe Dippel. Agent-based monitoring of virtual machines. In *International Symposium on Information Technology*, volume 1, pages 1–6, 2010.

[143] Giuseppe Aceto, Alessio Botta, Walter de Donato, and Antonio Pescapè. Cloud monitoring: A survey. In *International Journal of Computer Networks*, volume 57, pages 2093–2115, 2013.

[144] Lieve Hamers, Yves Hemeryck, Guido Herweyers, Marc Janssen, Hans Keters, Ronald Rousseau, and André Vanhoutte. Similarity measures in scientometric research: The jaccard index versus salton's cosine formula. *In Information Processing & Management*, 25(3):315–318, 1989.

[145] Rui Zhuang, Scott A DeLoach, and Xinming Ou. Towards a theory of moving target defense. In *Proceedings of the ACM Workshop on Moving Target Defense*, pages 31–40, 7 Nov, Scottsdale, USA, 2014.

[146] Hamed Okhravi, Thomas Hobson, David Bigelow, and William Streilein. Finding focus in the blur of moving-target techniques. *IEEE Security & Privacy*, 12(2):16–26, 2013.

[147] Ping Chen, Jun Xu, Zhiqiang Lin, Dongyan Xu, Bing Mao, and Peng Liu. A practical approach for adaptive data structure layout randomization. In *Proceedings of the European Symposium on Research in Computer Security*, pages 69–89, 21-25 Sept, Vienna, Austria, 2015. Springer.

[148] Kanad Sinha, Vasileios P Kemerlis, and Simha Sethumadhavan. Reviving instruction set randomization. In *Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust*, pages 21–28, 01-05 May, Mclean, USA, 2017. IEEE.

[149] Jin-Hee Cho, Dilli Sharma, Hooman Alavizadeh, Seunghyun Yoon, Noam Ben-Asher, Terrence Moore, Dong-Seong Kim, Hyuk Lim, and Frederica Nelson. Toward proactive, adaptive defense: A survey on moving target defense. *IEEE Communications Surveys & Tutorials*, 22:709–745, 2020.

[150] Andrei Homescu, Todd Jackson, Stephen Crane, Stefan Brunthaler, Per Larsen, and Michael Franz. Large-scale automated software diversity—program evolution redux. *IEEE Transactions on Dependable and Secure Computing*, 14(2):158–171, 2015.

[151] Shohreh Hosseinzadeh, Sampsa Rauti, Samuel Laurén, Jari-Matti Mäkelä, Johannes Holvitie, Sami Hyrynsalmi, and Ville Leppänen. Diversification and obfuscation techniques for software security: A systematic literature review. *Information and Software Technology*, 104:72–93, 2018.

[152] Jafar Haadi Jafarian, Ehab Al-Shaer, and Qi Duan. An effective address mutation approach for disrupting reconnaissance attacks. *IEEE Transactions on Information Forensics and Security*, 10(12):2562–2577, 2015.

[153] Dilli Prasad Sharma, Dong Seong Kim, Seunghyun Yoon, Hyuk Lim, Jin-Hee Cho, and Terrence J Moore. Frvm: Flexible random virtual ip multiplexing in software-defined networks. In *Proceedings of the IEEE International Conference On Trust, Security And Privacy In Computing And Communications*, pages 579–587, 01-03 Aug, New York, USA, 2018. IEEE.

[154] Stefan Achleitner, Thomas La Porta, Patrick McDaniel, Shridatt Sugrim, Srikanth Krishnamurthy, and Ritu Chadha. Deceiving network reconnaissance using sdn-based virtual topologies. In *IEEE Transactions on Network and Service Management*, volume 14, pages 1098–1112. IEEE, 2017.

[155] Stephen Boyd and Angelos Keromytis. Sqlrand: Preventing sql injection attacks. In *Proceedings of the International Conference on Applied Cryptography and Network Security*, pages 292–302, 8 Jun, Berlin, Germany, 2004. Springer.