A FORMAL APPROACH TO SECURE THE SEGMENTATION AND CONFIGURATION OF DYNAMIC NETWORKS

A FORMAL APPROACH TO SECURE THE SEGMENTATION AND CONFIGURATION OF DYNAMIC NETWORKS

BY

MOHAMMED ALABBAD, M.A.Sc., B.Sc.

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTING AND SOFTWARE

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

Doctor of Philosophy (2021) McMaster University

(Software Engineering) Hamilton, Ontario, Canada

TITLE:	A Formal Approach to Secure the Segmentation and Configuration of Dynamic Networks
AUTHOR:	Mohammed Alabbad M.A.Sc., (Software Engineering) McMaster University, Hamilton, Ontario, Canada B.Sc., (Computer Science) King Saud University, Riyadh, Saudi Arabia
SUPERVISOR:	Prof. Ridha Khedri
NUMBER OF PAGES:	xvi, 258

To my family

Abstract

Network segmentation and layered protection are critical strategies used in building and designing secure networks. Although they are recommended by security practitioners and agencies, they are defined vaguely and lack precise formal treatment. Implementing these strategies might be achievable for a small network with few resources; however, it is nearly an impossible task for a large network with a large number of resources and complex policies. The challenge is even harder for dynamic networks, where resources frequently join and leave the network. This case requires an adaptive approach for maintaining the implementation of these strategies.

In this thesis, we provide a formalism for the strategies of layered protection and network segmentation. The formalism is based on Product Family Algebra (PFA) and guarded commands. We use this formalism to assess whether a network satisfies these strategies. Furthermore, we articulate two implementation schemes for the layered protection strategy. Moreover, based on the introduced formalism, we propose two algorithms for structuring and configuring robust and secure networks. We then extend the formalism and the algorithms to handle networks with several entry points, where each entry point is intended to give access to a certain subnetwork. We employ the algorithms for the dynamic configuration and governance of Software Defined Networks (SDN). In addition to SDN data and control planes, we propose a plane in charge of the configuration and governance of SDN data planes. We call it the Dynamic Configuration and Governance (DCG) plane and it is intended to

give agility to dynamic networks. Moreover, we propose and assess three architectures that use the DCG plane. The assessment results identify an architecture that is suitable for dynamic networks and another for networks that are more stable regarding changes to policy and network topology.

The formalism presented in this thesis provides an automatic and adaptive approach for the segmentation and configuration of networks. It takes into consideration the security requirements of local resources as well as the global security situation. It constitutes a foundational framework for automated security solutions applicable to computer networks that use any type of connecting technology or topology.

Acknowledgements

First and foremost, I wish to start by expressing my sincere gratitude to Allah for the many innumerable blessings that I have given and for the completion of this work.

I would like to express my sincere gratitude to my supervisor Prof. Ridha Khedri for his substantial support and encouragement throughout my research and on a personal level. His patience, guidance, expertise, and motivation were fundamental for the completion of this thesis. I am extremely grateful for everything he has done through this journey.

I would also like to thank my supervisory committee members Dr. Spence Smith and Dr. Emil Sekerinski for their support and valuable feedback during my research.

I would like to thank my employer King Abdulaziz City for Science & Technology (KACST) for funding my graduate studies.

Finally, I would like to express my deepest acknowledgments to my dear family, my father, my mother, my wife, my daughters (Sarah, Jood, and Jana), and my siblings for their endless love, patience, support, and encouragement.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Network systems consist of many interconnected resources. Access to each resource is regulated according to an access control policy. A network security system needs to guarantee the overall network protection as well as individual resource protection. Moreover, it also needs to guarantee the consistency of policies enforced at the different access control points in the network. Furthermore, security design principles and best practices need to be followed when structuring and configuring networks such as least privileges and separation of concerns to ensure secure design structures. With the growth of network systems in scalability and complexity, it becomes a challenge to adhere to the design principles and best practices to preserve the security of resources. Therefore, a formal and automated approach is needed to tackle network governance.

This chapter presents the context of the problem and motivates the need for a network governance formalism that deals with securing networks through access control policies. Precisely, Section 1.1 gives an overview of computer security and computer networks. Afterward, it discusses network access control systems and their role in network security. Then, it presents security design principles and strategies to be followed in designing secure networks. Section 1.2 motivates the need for a mathematical formalism that deals with policies as related families. Section 1.3 states the research problem, objectives, and the

methodology that we take to achieve the objectives. Section 1.4 summarizes the contributions of this thesis. Section 1.5 lists the related publications. Finally, Section 1.6 outlines the structure of the remainder of the thesis.

1.1 General Background

The focus of this thesis is securing network resources. This section gives a general overview of the topics of computer security and computer networks and their relation.

1.1.1 Computer Security

The main pillars of computer security are *confidentiality*, *integrity*, and *availability* [Bis02]. Confidentiality refers to the protection of resources and information from unauthorized access or disclosure. The need for protection arose from the use of computers and technology in all areas of life including sensitive ones such as health care, military, government, and economy. The initial work of computer security started in the military to enforce the principle of "need to know" which restricts access to resources and information to only those who need it. Access control systems and security mechanisms such as cryptography and resource hiding support confidentiality. Integrity is the trustworthiness and correctness of data. It is the guarantee that data has not been modified in an unauthorized way. Integrity includes *data integrity* which is the trustworthiness of the content of data and *origin integrity* which is the trustworthiness of the source of data or what is known as *authentication*. Integrity mechanisms have two classes: prevention and detection. Prevention mechanisms aim to preserve the integrity of data by denying unauthorized access to change data or change the data in an unauthorized way. An access control system prevents integrity violation by blocking unauthorized access to data and resources. Detection mechanisms aim to report any violation of data integrity. Availability is the ability to use resources or access information when needed and that it is not denied for authorized users. An access control system

needs to guarantee resource availability.

We highlight the following elements of security: *authentication*, *access control* or *authorization*, and *nonrepudiation*. Authentication asserts establishing a relation between a user and an identity; it is the process of identifying users. Access control or authorization is the management of access to resources and information such that unauthorized users are denied access and only authorized users are allowed. Nonrepudiation is the prevention from denial of participating in an action or transaction either as a receiver or originator [Ben06].

When discussing security, it is important to specify what is secure and non-secure [Bis02]. A security policy is a statement of what is allowed and what is not allowed. More precisely, a security policy is defined as "a statement that partitions the states of the system into a set of *authorized*, or *secure*, states and a set of *unauthorized*, or *non-secure*, states" [Bis02]. For example, an access control security policy for a resource might be to block access requests coming from the internet. A security policy might refer to multiple meanings based on the context [GR95]. It could mean the high-level statement to ensure the overall security of the organization's data and resources. It also could refer to the low-level rules that implement the organization's security policy. A policy can be represented mathematically as a list of secure and non-secure states. The actual enforcer of a policy whether it is a tool, method, or procedure is called a *security mechanism*.

A *threat* is the potential of security violation [Bis02]. The violation does not need to happen to be called a threat. The actual occurrence of violation is called an *attack*. Threats can be external or internal. One of the internal threats is employees who have access to information they do not need [GR95]. A threat, which is the potential of attack, needs to be guarded against. Confidentiality, integrity, and availability aim to guard systems against threats.

Security mechanisms when enforcing security policies use three strategies: prevention, detection, and recovery [Bis02]. These strategies might be used separately or in combination. Prevention is the blocking or failing of an attack. For example, denying a certain source

access to a resource is a prevention mechanism to enforce confidentiality. Detection mechanisms assume that attacks will happen and attempt to discover an attack underway or a completed attack and report it. Such mechanisms typically monitor multiple aspects of the system to indicate if an attack is underway or has happened. Recovery is the repair of the damage caused by an attack.

Therefore, computer security [GR95] is the protection provided for an information system to preserve its confidentiality, integrity, and availability. It starts with threat assessment then countering them using policies and mechanisms that use prevention, detection, or recovery to maintain the security of the system.

Access Control

Access control is the protection of information and resources from unauthorized users, and at the same time, ensuring their availability for authorized users [SD01]. It is motivated by the need to limit access to resources and information to authorized users only [Ben06]. Access control systems are an essential tool to guard against security threats. They enforce policies to protect organizations' valuable assets from unauthorized access. Therefore, preserving the confidentiality and integrity of computer systems. When the enforced policies are well specified and implemented, they make a shield that allows only authorized users. By limiting the access to allowed users only, access control systems ensure the availability of information for authorized users.

Access control systems implement the organization's security policy written in a high-level language into security mechanisms that enforce access control policies. We refer to these security mechanisms as *access control points*. An access control point allows subjects to access the resources they are authorized to and denies unauthorized users.

Access control is based on identifying users (authentication) using trusted methods and establishing an *identity trust*. The goal of this is to base access control decisions on a secure foundation. An access control policy is a statement that specifies subjects that have access

to objects and the type of access that is permitted [GR95]. An access control policy contains an error if it allows unauthorized access or denies an authorized access [JGT+11]. A safe access control policy and mechanism should prevent unauthorized users from gaining access directly or indirectly to resources [Ben06].

An access control system decides on whether to permit a user to access a resource for the type of access requested based on multiple conditions that can be used in combination. These conditions could include user identity, role, location such as network address, time, transaction such as giving an employee access limited to achieving a transaction or mission and denying it afterward, and service constraints such as limiting the number of allowed users to access a server at the same time [GR95].

1.1.2 Computer Networks

A computer network [TW10] is a collection of independent computer resources connected using the same technology. Resources are connected if they can exchange information. The connection medium can be any kind of wired or wireless technologies.

Networks are built based on layers stacked over each other [TW10]. Networks can differ in the details of their layers. However, each layer is built to offer services to the layer above it while hiding implementation details in what is known as "information hiding".

A *protocol* is an agreement by which two corresponding layers on different machines communicate. In practice, layers on different machines do not communicate directly, each layer actually passes the information to the layer below it until the information reaches the physical layer where the communication takes place. *Interfaces* are used to define the services and functionalities offered by a layer for the layer above it, similar to function headers in programming languages.

Building networks in a layered fashion gives flexibility for replacing or updating layers as long as they offer the same services and have the same interface. Moreover, it allows for expansion by adding new layers. This is similar to hierarchical architecture in software

engineering.

Network architecture is the set of layers and protocols. Figure 1.1 shows a network architecture model presented in [TW10]. This model combines the Open Systems Interconnection (OSI) reference model and TCP/IP reference model. It has five layers starting, bottom-up, with physical layer, link layer (i.e., data layer), network layer (i.e., internet layer), transport layer, and application layer. The physical layer is the layer right above the physical transmission medium and the concern of this layer is transmitting bits as signals along different transmission mediums. The link layer is concerned with sending reliable messages with limited lengths that appear to be free of transmission errors to the network layer. It does this by breaking data into frames and transmitting them. The corresponding layer on the receiver machine confirms the receipt of frames by sending an acknowledgment frame. The network layer is concerned with the details of sending packets between distance machines. This includes routing and finding paths to send packets from source to destination. The packet format protocol Internet Protocol (IP) is defined at this layer. The transport layer is concerned with increasing the reliability of the network layer and allowing machines to create communication. Transmission Control Protocol (TCP) is a protocol defined at this layer. TCP segments bytes into messages and pass them to the lower layer. It is a reliable connection-oriented protocol that takes the task of delivering a stream of bytes without errors and in sequence. On the receiver machine, the opposite layer sorts the received messages into an output of a stream byte. The User Datagram Protocol (UDP) is the second protocol defined at this layer. It is unreliable, connectionless, and used by applications not requiring the sequencing of TCP. Finally, the application layer defines multiple high-level protocols needed by the user. For example, HyperText Transfer Protocol (HTTP) the protocol used for browsing web pages, File Transfer Program (FTP), Simple Mail Transfer Protocol (SMTP), and Domain Name System (DNS) are defined in this layer.

The initial use of computer networks was for academic purposes with no concerns for security [TW10]. Nowadays, the use of computer networks has expanded to be involved in many

5	Application
4	Transport
3	Network
2	Link
1	Physical

Figure 1.1: Network architecture reference model

critical fields ranging from personal affairs to governmental levels which include health care, banking, military, and more. Moreover, networks have grown in size from small networks to large networks including the internet. And from stationary terminals to mobile devices, home appliances, and health monitoring systems hooked to networks. Therefore, with the connectivity and scalability of networks and their involvement in critical fields, network security became a legitimate concern.

Security breaches in computer networks cover a wide range including but not limited to disclosure or modification of confidential information, unauthorized access to resources, damaging assets, identity theft, and denial of responsibility. Although most attacks are intentional for various motives, they can be unintentional such as disclosure of information by careless employees. Moreover, the focus of most security systems is the protection from outside threats, however, the reality is that most attacks are an insider job.

Network security mechanisms work on all layers of the network. On the physical layer, there are security measures to ensure the protection of the physical assets. Moreover, encryption mechanisms work on the data link layer. IP Security (IPsec) and firewalls work on the network layer. Authentication and nonrepudiation are dealt with on the application layer.

Network Access Control

Network access control is a very important aspect of network security. Its main concern is regulating access to resources in a network environment according to access control policy, which is a part of the overall security policy.

Connecting an organization's network to the internet is beneficial for the business, however, it brings security risks and threats [TW10]. The direction of the traffic in these threats includes outgoing traffic such as disclosed information leaving the organization's internal network to unauthorized parties. It also includes incoming harmful traffic, such as viruses, worms, or an attack on resources and assets. Therefore, there is a need to protect network resources from certain incoming and outgoing traffic. One way of protection is using IPsec by which a communication tunnel between two sites is protected, however, it does not protect resources from unauthorized access. Firewalls acting as access control points are used to achieve this goal.

Firewalls

Packet filtering firewalls, act as security checkpoints, inspect all traffic coming in or going out from networks. They allow or drop packets based on criteria or policies specified as sets of rules. These rules specify the conditions for which packets are allowed or dropped. The conditions are usually based on information in the packet header which can be source IP address, source port, destination IP address, destination port, or protocol. Port numbers are used to specify the services requested. For example, SMTP use TCP port 25 and HTTP uses TCP port 80. An organization that wishes to block *Telnet* services would deny traffic for TCP port 23. Stateful firewalls keep track of the state of connections. So, when a connection is established, the firewall does not need to check the packet filter rules for every packet exchanged that belongs to that connection. Therefore, improving connectivity and firewall performance. Application-level firewalls peak into the content of packets and go beyond checking packet headers. This enables firewalls to block traffic not only based on the attributes of packet headers but also on the application used. The topic of firewall technology will be explored in depth in Chapter 2.

An access control system consists of two components: access control policies and access control points. An access control policy can refer to the high-level policy or the low-level

rules derived from the high-level policy. An access control point is the security mechanism that enforces these policies and rules. For example, a high-level rule can be to block the email service for a certain resource, which then translates to a low-level firewall rule that blocks TCP connections on port 25. The rule gets enforced by the packet-filtering functionality of the firewall. One of the issues related to access control is the translation of high-level policy specification to low-level rules. It needs to be ensured that the low-level rules actually implement the high-level policy and that they are consistent and free of conflicts.

Firewalls and access control points deal with two levels of protection [CB94]: host-based and network levels. Host-based access control is concerned with the protection of a single host using its security mechanisms such as a local software firewall. This is an essential step in securing resources; however, it is not sufficient for many factors related to limited resource capabilities. Network-level access control is concerned with the protection of a cluster of resources. Network firewalls can be deployed at the entry point of the network and internally at multiple locations protecting subnetworks with sensitive resources. Internal firewalls can be used to implement a layered defense which is a manifestation of the principle of separation of concerns [Ben06].

Traditionally, firewalls are implemented at the entry point of networks with no control of internal traffic. This approach is not sufficient to protect network resources because it assumes every host in the internal network is trusted and therefore does not provide any protection from internal attacks. Moreover, an external attack that succeeds in passing the security at the entry point will gain free lateral movement within the network. Another issue is that firewalls cannot inspect encrypted packets. For these reasons, distributed firewalls were proposed. In the initial proposal [Bel99], the high-level policy is distributed on each resource. However, this solution was not sufficient because it relies only on host-based security and therefore it allows illegitimate traffic to go inside the network without any line of defense except the host firewall.

Challenges of Dynamic Networks

Modern networks and related technologies such as Big data, Internet of Things (IoT), Software Defined Networking (SDN), and Network Function Virtualization (NFV) have experienced exponential growth in the last few years and expected to grow further in the future. However, security challenges are some of the factors that limit the growth of such technologies. These challenges are the result of the nature of the vulnerabilities of the environment. One of the characteristics of such an environment is its dynamism. Resources in these networks join and leave the network frequently. Moreover, the size and scale of these networks are growing dramatically. To mitigate against the changing security threats and the dynamic nature of the environment, security solutions need to be adaptive [KG19]. The static security solutions of traditional networks are not sufficient to provide security for such a changing and evolving environment. Therefore, the focus of research efforts should be on adaptive security as suggested in [KG19].

A network access control system potentially consists of hundreds or thousands of access control points. The correct configuration and placements of these devices play a key role in the effectiveness of the system. In practice, network administrators design networks and configure devices following security design principles and best practices. This approach is prone to design errors and misconfiguration, especially for a large network. Moreover, it does not address the scalability and dynamicity aspects of modern networks.

Network Design

Some of the security design principles that should be adhered to when designing and configuring networks include least privileges and separation of concerns [SS94, CB94, Ben06, Sta07, Sta09, Pet01]. The principle of least privileges requires that the granted access for users to information and resources is limited to the ones that are necessary to perform their

job or task. The principle of separation of concerns/privileges/duties requires that for certain important tasks, no single user is allowed to perform all their subtasks and that access to resources should be based on the satisfaction for more than one condition. Two strategies that are recommended to achieve the above principles: layered defense and segmentation. Layered defense (i.e., layered protection) requires having multiple lines of defenses such that if one layer or line fails the other ones will not. Segmentation states that resources of similar security requirements should be located together in one cluster protected by a firewall. Therefore, sensitive resources are placed together and provided maximum security, and on the other hand, resources with loose security requirements are placed together. Demilitarized Zone (DMZ) is an example of applying segmentation. In the literature, these strategies are described and discussed intuitively without any formalism.

In this work, we aim at articulating a formalism for network governance. The formalism includes a mathematical framework that captures the network policies as a family of related security policies, a formal definition and implementation schemes for layered protection, and formally defining segmentation and an algorithm to achieve robust network design. The formalism is based on Product Family Algebra (PFA) [HKM11a, HKM06] where each of the resource policies is considered as a program specified using a formal policy modeling language. The mathematical framework is used to formalize the concepts of layered protection and segmentation. We also show the applicability of these approaches to modern networks by applying it to SDN.

1.2 Motivation

The strategies of layered protection and segmentation are effective in designing secure network architectures and mitigate security threats [Goo12, Cen15, Col09]. The existed discussion on these strategies is limited to best practices and guidelines with no formal approaches. The implementation of these strategies is left to the judgment of system administrators.

This approach is error-prone and hard to implement, especially for large networks. Moreover, the research studies that tackle these strategies are limited as will be explored in detail in Chapter 2. Therefore, there is a lack of formalism to define and achieve these strategies in network environments.

As discussed above, the goal of network access control is to prevent activities that could threaten the network's security. It does this by denying access of unauthorized users and limiting the access of authorized users to the needed resources [SS94, DFSJ07]. For a network security system to be effective, the network design has to adhere to the strategies mentioned above. Moreover, the security system needs to be adaptive to cope with changes in resource policies, resource availability, threats, or any other factor. These changes are dynamic in nature and thus require a quick and dynamic response, which is usually performed manually by system administrators. As networks consist of interconnected resources, a software product family approach [CN02, KD06, PBvdL05] is needed to model resource and global policies and to deal with changes in topology and policies. This ensures that changes in one dimension or a resource are reflected in the overall policies and topology of the network.

1.3 Problem Statement, Objectives, and Methodology

1.3.1 Problem Statement

To fill the research gap, this thesis aims to provide a formalism to address the network governance strategies of layered protection and segmentation. The basis of this formalism is Product Family Algebra (PFA) [HKM11a] and the concepts of feature modeling and software product line engineering. Moreover, we need to explore the question of transferring the formal results into real networks, such as Software Defined Network (SDN).

1.3.2 Objectives and Methodology

To achieve the goals of my research, the carried-out activities are mapped to several objectives described below.

Objective 1: A product family model to specify and analyze access control policies within a network system

The first objective is to adopt the commonly used mathematical formalism of guarded commands to specify access control policies. We model related access control policies as a product family. To achieve this, we give an interpretation to the PFA operators in terms of the mathematical model of guarded commands. Currently, PFA provides an abstract view of features used in a family of products. We aim at providing a model for the features of a family in terms of access control policies.

Objective 2: Articulating a formalism related to configuring access control points

The second objective is to formalize the concept of layered protection into the Defense in Depth (DD) strategy. Moreover, we aim at articulating implementation schemes to configure access control points according to the formalism of DD strategy. We also aim at formalizing a stricter form of DD. Moreover, we aim at implementing a prototype for an automated solution for managing and enforcing access control policies in a network applying the defense in depth strategy. The implementation uses a broker architecture where the management of policies is done centrally at the broker and the enforcement distributed at the access control points.

Objective 3: Articulating a formalism for network design and structure

The third objective is to articulate a formalism that allows us to design secure networks. Based on the results of Objects 1 and 2, we plan to propose formal definitions of network segmentation strategy and robust and secure network structure. Then, we seek to articulate a formal algorithm to systematically achieve a robust and secure network structure.

Objective 4: Usage of the obtained results in Software defined Networks (SDN)

The fourth objective is to build different architectures for implementing the segmentation algorithm proposed in Objective 3 into an SDN environment. We aim to assess the architectures and draw a conclusion on the most suitable architecture.

1.4 Contributions

Through the fulfillment of the objectives presented in Section 1.3, the research contributions are as follows:

1. A mathematical model for network policies that use a family approach enabling us to reason on large networks.

2. A formalism for the layered protection as DD strategy. Moreover, the thesis proposes multiple schemes for implementing the DD strategy to configure network devices. The strategy and schemes can be used to assess network configurations and for network audits. Furthermore, the thesis gives a formal definition for Strict Defense in Depth (SDD) which is a stricter form of DD strategy, and a set of results on when it is not possible to achieve SDD in a network. The SDD is later used to place segments when designing networks. The obtained results guide network designers to achieve a robust and secure network design.

3. An approach to include several security concerns into a weight function that assigns weight value for resource access control policies. The function takes into account security requirements that need to be taken into account to be used for the segmentation of network resources.

4. Formally define what is a segment and network segmentation based on the weight of commonality of policies. This leads to the formal definition of robust and secure network based on the network segmentation and SDD defined above.

5. Two algorithms to achieve robust and secure network architecture. One is exponential and the second is polynomial. The polynomial algorithm is obtained through collaboration with other researchers as indicated in [MAK21].

6. Implementing the algorithms into SDN to form a new plane that is specialized in structuring data plane topology. Moreover, the new plane is adaptive such that it restructures the data plane dynamically in response to changes in policies or resource availability. We have proposed three different architectures for the implementation and the assessment of the three architectures to reach a conclusion on which architecture is the most suitable.

1.5 Related Publications

This section lists the publications related to the research presented in this thesis.

1.5.1 Journal Articles

- [MAK21] N. Mhaskar, M. Alabbad, and R. Khedri. A Formal Approach to Network Segmentation. Computers & Security, page 102162, 2021.

1.5.2 Conference Papers

- [KJA17] R. Khedri, O. Jones, and M. Alabbad. Defense in Depth Formulation and Usage in Dynamic Access Control. In M. Maffei and M. Ryan, editors, Principles of Security and Trust: 6th International Conference, POST 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, pages 253–274, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg.

- [AK21] M. Alabbad and R. Khedri. Configuration and governance of dynamic secure SDN. In The 12th International Conference on Ambient Systems, Networks and Technologies (ANT 2021), Procedia Computer Science series, pages 1–8, Warsaw, Poland, March 23 – 26 2021. Elsevier Science.

1.5.3 Technical Reports

- [KMA19] R. Khedri, N. Mhaskar, and M. Alabbad. On the Segmentation of Networks. Technical Report CAS-19-01-RK, McMaster University, Hamilton, ON, Canada, March 2019.

1.6 Thesis Structure

The remaining chapters of this thesis are organized as follows:

Chapter 2 surveys the literature related to the network design strategies of layered protection and segmentation. Since our approach is based on access control perspective, the chapter presents access control and the research topics s related to it. Moreover, we are interested in implementing the strategies in SDN, therefore, this chapter introduces SDN and research approaches that implement firewalls in SDN.

Chapter 3 introduces the needed mathematical fundamentals including PFA, guarded commands, and tabular expressions.

Chapter 4 proposes a model for specifying network resource policies. It also discusses a family approach using this model and different usages of the model.

Chapter 5 gives formal definitions of DD and SDD. It also presents different schemes for achieving DD in a network and results related to SDD that helps to articulate a formal algorithm for segmentation in Chapter 6.

Chapter 6 presents a formal approach to tackle the notion of segmentation and robust network, then proposes a systematic algorithm to achieve a robust network. The algorithm considers the case when we have a network with a single entry point.

Chapter 7 extends the algorithm proposed for segmenting networks with a single entry point to segment networks with multiple entry points.

Chapter 8 presents three different architectures for implementing the segmentation algorithm presented in Chapter 6 in an SDN environment. Moreover, it presents an assessment to determine the most suitable architecture for an SDN network.

Chapter 9 concludes the thesis and points for future work.

Chapter 2

Literature Survey

The strategies of layered protection and segmentation are used to design secure network structures. This thesis formalizes these two strategies from an access control policy perspective. More precisely, layered protection is the first tackled strategy. Then, the results obtained from the first strategy are used to tackle the network segmentation strategy. Moreover, the results are implemented on SDN environment to show their applicability for real modern networks.

This chapter presents a literature survey related to access control, layered protection and segmentation strategies, and firewalls within SDN. Specifically, Section 2.1 gives an introduction to access control, its paradigms, and its models. Section 2.2 presents the layered protection strategy and related research. Section 2.3 presents the segmentation strategy and related research. Section 2.4 presents research studies attempting to apply the concept of product family into security and access control domain. Section 2.5 reviews other research topics related to access control. Finally, Section 2.6 presents the SDN and related research.

2.1 Access Control

In Chapter 1, we have introduced the concept of access control and established the importance of access control systems to preserve the security aspects of network resources: confidentiality, integrity, and availability. Access control systems control every access attempt and ensure that only authorized users are allowed while unauthorized users are denied [SS94, SD01]. It is important to differentiate the concepts of *access control policy*, *access control model*, and *access control point* [SD01, HFK06]. The access control policy is the high-level security policy that the access control system regulates traffic according to. The access control model is the formal representation of the policy which allows to prove properties on the policy. The access control point is the security mechanism that enforces the policy. These concepts separate multiple levels of abstraction. Specifically, the separation between the requirement in the high-level policy, the actual enforcement in the access control point, and the model level which closes the gap between the requirement and implementation. An access control point should be secured to be altered only by authorized personnel and using trusted channels. Moreover, the access control point should regulate every access attempt with no exceptions. Access control policies in reality are complex and their requirements come from different organizational perspectives such as laws, organizational regulations, and best practices. A policy must capture all these perspectives in a single consistent policy. This particular issue makes a family approach suitable for access control policies.

2.1.1 Access Control Paradigms

Access control paradigms are differentiated by who has the authority to specify access control policies. Below we present Discretionary Access Control (DAC) and Mandatory Access Control (MAC).

Access in DAC [SS94, SD01, HFK06] is regulated based on the identity of the user. It gives

subjects (i.e., users) the right to grant and revoke access to objects (i.e., resources) they own. Moreover, users can delegate permissions they have to other users. Granting and revoking permissions are managed by certain rules. In this paradigm, the emphasis is on information sharing which makes it suitable for commercial and business scenarios.

The access management in MAC [SS94, SD01, HFK06] is enforced by a central authority which is often the administrator of the system. Owners of resources do not have control over access to their resources. MAC paradigm is related to multi-level security models. In multi-level security models, users and objects are classified into different classifications. Bell-Lapadula [BLP76] and Biba [Bib75] are examples of mandatory multi-level security models. This paradigm is a military-based where the confidentiality and integrity of information and resources are the main concerns.

2.1.2 Access Control Models

Access Control Matrix

Access control matrix is a model or framework for describing DAC proposed by Lampson [Lam74] for protecting resources in operating systems, which was later formalized by Harrison et al. [HRU76]. An access control matrix model has a set of subjects, a set of objects, and a set of permissions. The authorization is represented as a matrix where the matrix rows are labeled by subject names and its columns by object names. Each cell contains a set of allowed access permissions. There are two implementations of access control matrix: Access Control List (ACL) and capability lists. In a capability list, resources and permissions of a subject are assigned to the subject. A capability list comes in a form of a table of pairs containing the resources and subject's permissions. A capability list can be compared to a ticketing system [TS06]. In this case, it is important to protect tickets from modification by the subject. On the other hand, in ACL, a list is attached to the resource. The list contains subjects that have access to the resource and their permissions.

Role-Based Access Control (RBAC)

Role-Based Access Control (RBAC) [FK92, SCFY96, SFK00, FSG+01] is a non-discretionary access control model where the administrator is responsible for enforcing the policies. In RBAC, the access control decisions are based on the roles of the users rather than on their identities. A user will obtain the allowed permissions by playing a role, in which case this user will inherit all the permissions associated with that role. Because roles within organizations are relatively persistent compared to users who switch positions within the organization or leave the organization, RBAC reduces the complexity, cost, and errors in policies establishment and maintenance. By limiting roles assignment to only the needed permissions, RBAC supports the principles of least privileges and separation of concerns.

Attribute Based Access Control (ABAC)

Attribute Based Access Control (ABAC) [HFK+14, HKFV15] grants or denies access of users to resources based on some attributes of the user, selected attributes of the objects, and environment conditions. Using ABAC an administrator can create rules without specifying a relation between a specific user and a specific resource. In this model, a user is assigned a set of subject attributes upon joining the organization. A resource is assigned a set of object attributes upon joining the system. An administrator creates a rule based on subject and object attributes. Upon an access request, an access control point matches the attribute values in the request against the policy it has to reach a decision. To change an access decision, all that is needed is changing attributes without changing the rules which gives a dynamic access control management and limits the maintenance of rules.

In addition to the above access control models, others found in the literature including the Chinese wall policy [BN89], Risk-Adaptive Access Control (RAdAC) [CRK+07], Policy-Based Access Control (PBAC) [ZJXsLx09], Organization Access Control (ORBAC) [KBB+03],

Administrative RBAC (ARBAC) [SBM99], Temporal RBAC (TRBAC) [BBF01], Task-Based Authorization Controls (TBAC) [TS98], Team-Based Access Control (TMAC) [Tho97], and Virtualized RBAC (VRBAC) [LLT+13].

The formalism presented in the thesis is a generalized formalism and therefore can be used to reason on any of the models presented above. However, our focus in the examples and implementation is on the firewall policies domain.

2.1.3 Firewalls

Firewalls are the first line of defense in network security. Due to this fact, we use firewall policies in the examples given in the thesis. In the following, we give an overview on firewalls. Firewalls are hardware or software artefacts that control traffic between networks with different security level requirements based on a predefined security policy [VE05, SH09, Zal10]. Every packet passing through a firewall is inspected and checked. Then it is either allowed to pass or rejected based on the firewall policy. A firewall policy is an ordered rule list, or an ACL, that is a translation of the organization's high-level policy. Therefore, writing firewall policies that reflect the organization's needs is critical in protecting resources. Firewall rules and network policies are domain (i.e., mechanism) specific access control policies. Firewalls are access control points enforcing the policy. The firewall rules define the actions to be taken for traffic matching their conditions. The firewall checks the rules sequentially and applies the first matching one.

Figure 2.1 shows an example of a firewall policy written in the language of *iptables*[1] [Rus02, np18]. Firewalls have two basic actions to either allow or drop traffic. However, in *iptables* firewalls have other actions such as `LOG`, `ACCEPT`, `REJECT`, or `DROP`. The action `LOG` does nothing to the traffic and records it in the Syslog file. The action `ACCEPT` allows traffic to pass. The action `DROP` blocks the traffic. The action `REJECT` blocks the traffic and sends an error message to the requester to acknowledge the receipt of the request and save the

[1] *iptables* is a command-line utility program used to configure Linux firewall

requester from reattempting a request.

```
1 -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
2 -A INPUT -m state --state INVALID -j DROP
3 -A INPUT -p tcp -m state --state NEW -m tcp --dport 80 -j ACCEPT
4 -A INPUT -p tcp -m state --state NEW -m tcp --dport 25 -j ACCEPT
5 -A INPUT -s 192.168.1.0/24 -j ACCEPT
6 -A INPUT -s 192.168.2.0/24 -j ACCEPT
7 -A INPUT -s 192.168.3.0/24 -j ACCEPT
8 -A INPUT -s 192.168.4.0/24 -j ACCEPT
9 -A INPUT -j DROP
```

Figure 2.1: Web server policy

As mentioned above, firewalls execute rules in a sequential manner, which results in a decrease in firewall performance. This is especially true if the matching rule is at the bottom of the rule list. The work in [AS14] enhances the performance of the firewalls by improving the order of the rule list. Another proposal is the use of parallel firewalls [WLWF15] which results in an improved performance especially if the rule domains are disjoint.

As discussed in Chapter 1, there are different types of firewalls [TS06, SH09]. Packet-filtering firewalls make the decision to pass or deny a packet based on its attribute values in the packet header. They inspect every incoming and outgoing packet and check it to the policy. Stateful firewalls keep track of the state of the connection and therefore not checking the policy for each packet. On the other hand, application-level firewalls check the content of packets.

Firewalls could be deployed at the organization's network gate or entry point. Although this approach provides some protection, it does not provide any protection from internal attacks or monitor internal traffic. Bellovin [Bel99] propose the initial idea of distributed firewalls where the centrally defined policy is distributed to hosts such that each host protects itself rather than a single firewall protecting the gate of the network. The common practice nowadays is that firewalls are deployed at the gate and different places in the network to

provide extra layers of protection.

2.2 Layered Protection

One of the security strategies that should be adhered to in an access control system is layered protection or layered defense which is also known as Defense-in-Depth [Pet01, FH06, Sta07, Sta09, Col09, U.S16]. The DD strategy asserts to implement layers of defenses such that if one layer is compromised the others will not. This makes it difficult for an attacker to succeed in gaining access to sensitive resources by the need to pass through multiple access control points. Although it is a recommended strategy [Goo12], there is no formal approach to implement it in the literature. We present here some work related to this topic.

In an early work, Smith and Bhattacharya [SB97] propose the use of a chain of firewalls or what is called a *cascade of firewalls* to enhance security and trust. Rubel et al. [RIHP05] present an attempt to construct a single specification from which the policies of the different mechanisms can be derived in a defense in depth system. The work of Lippmann et al. [LIS+06] analyzes firewall rules and generates attack graphs to produce recommendations to restore defense in depth. Huang et al. [HKH+05] present a technique using data mining to collect data from scattered Intrusion Detection Systems (IDS)/Intrusion Prevention Systems (IPS) and updates other IDS/IPS systems to restore DD in a network. Thames et al. [TAK08] present a proactive approach to couple distributed intrusion detection systems with an active response and distributed firewalls to secure networks. Mavroeidakos et al. [MMV16] define a multilayered security architecture based on defense in depth for applications in the cloud.

The formalism given in Chapter 5 provides a formal presentation of layered protection that can be used to verify the implementation of layered protection in networks. Moreover, we propose two implementation schemes that can be used to generate policies for network access control points satisfying the layered protection.

2.3 Network Segmentation

Network Segmentation is a security measure to protect resources [Sta07, Sta09, FH06, Ram02, U.S16]. It partitions the network into segments or zones such that traffic going from one zone to another is monitored and controlled by an access control point or a firewall which makes gaining access to valuable resources a challenging task. By partitioning the network resources into segments of different security levels and restricting movements, the principle of least privilege is applied such that users are given access to only the segments necessary to perform their tasks and limit their access to other segments. It is a protection measure against insider attacks. Moreover, proper network segmentation helps in creating an effective layered defense or DD [U.S16].

Segmentation is also referred to in the literature as compartmentalization, pertaining, aggregation, slicing, or zoning. Although network segmentation is a suggested practice [Goo12, Nat13, Cen15], there is no formal approach on how to achieve it. Several research studies have been tackling this issue such as [DZLL14, WŞPS19, WŞŞP$^+$17, WŞW$^+$16, JSBZ15].

Segmentation is closely related to security mechanism placement. For example, Smith and Bhattacharya in [SB97], besides the proposal of firewalls cascade, propose a firewall placement heuristic to decide the best placement for firewalls that offer maximum protection with minimal cost. Moreover, Rahman and Al-Shaer [RAS13, AS14] provide a framework for security configuration and placement of security devices within the network.

Clark et al. [CMV00] present compartmentalization as a way to secure data. Du et al. [DZLL14] propose an algorithm based on network segmentation to secure wireless sensor networks (WSN). And Jeuk et al. [JSBZ15] propose a network segmentation architecture in the cloud.

Wagner et al. [WŞW$^+$16] present an approach to select the best network segmentation based on heuristic search. In [WŞW$^+$17] they present a method of evaluating defense mitigations such as network segmentation and their impact on security and organizational

mission. In [WŞŞP+17] they propose a low-cost alternative method to evaluate network segments for security risks. And in [WŞPS19] a method is presented to generate network segmentations suited for security and performance, evaluate them, then suggest the most appropriate ones.

Network slicing (e.g., [CSL+20, NGM16]), for the purpose of isolating resources, decomposes a network based on one of several possible criteria such as shared resources, storage, security requirements, or bandwidth. Hence, segmentation is a form of slicing based on security requirements and access policies. It uses access control policies to specify the rules to access services/resources within each network segment. These rules are grounded in security requirements. As far as we know, there is an absence of literature on the segmentation of networks with multiple entry points. However, several results on network slicing exist, that could potentially help with the segmentation of networks with multiple entry points (e.g., [NGM16]). In [MDS+17], the authors states that flexible network design is needed for next generation networks. For example, dynamic networks often require flexibility in their designs. The authors of [MDS+17] argue that flexible design offers the possibility to cope with a diverse set of requirements. In the case of segmentation as a form of slicing, it helps cope with security access requirements.

he studies mentioned above fall into two categories. The first category is related to segmenting network resources using heuristic approaches from the perspective of certain security measures (e.g., risk assessment measures). The second category is related to hardening the security of existing networks. None of these studies use a formal approach to achieve network segmentation. Their approaches lack the ability to automate the derivation of the best solution for network segmentation and prove its correctness. The formalism presented in this thesis defines segmentation formally from the perspective of access control policies. Moreover, it proposes a formal and systematic approach to achieve it. More on this discussion in Section 6.3.3.

2.4 Product Family

In the literature, we find research studies that try to apply the concept of product family into security and access control systems. These research studies include [PGF+15, KKLS14, DSE12, LFP11, HPF16].

Pinto et al. [PGF+15] use dynamic software product lines (DSPLs) in wireless sensor networks (WSNs) setting to enable the modeling of variability and dynamic self-adaption. First, a feature model is used to model the whole system for all possible products or devices including their security policies. A product for a given device is a single configuration of the model. The model allows for variabilities that can change at run time, and the system will be able to adapt to such changes. The configuration of a given device changes or gets reconfigured accordingly. To reconfigure a security policy at run-time, the new policy is compared to the old one, and based on the difference self-adoption steps are generated. Each device has a monitoring component that monitors the environment and triggers the self-adaption process.

Kim et al. [KKLS14] use a family approach to generate a hybrid model of RBAC and MAC. In their approach, RBAC and MAC are defined in term of features. RBAC and MAC are modelled as feature models. Each model is configured for a given application. The configured RBAC and MAC models are composed into a single hybrid configuration for the application.

Horcas et al. [HPF16] use a family approach to model functional quality attributes (FQAs) such as security. An FQA is modeled separately from the application in a feature model following a software product line (SPL) approach with its concerns as features. Concerns have requirements (e.g., mandatory or optional) and dependencies. Multiple configurations are generated from the model. An aspect-oriented approach is used to insert the configuration into the application model.

Derakhshanmanesh et al. [Der15, DSE12] propose a model-centric approach for dynamic

access control systems based on the concepts from dynamic software product lines [HHPS08] and adaptive software [ST09]. In [DSE12], the authors apply the approach on physical access control where RBAC is the policy model. In the approach, a manufacturer has a variant of cyber-physical access control systems presented in a feature model. A customer selects the desired features for their product along with an access control policy. A product is shipped with an adaption strategy that is implemented to the whole product line so it can react during operation. Adaption rules can adjust the product in foreseen events during runtime automatically. At some point, an event occurs which enforces a new requirement. If the new requirement was foreseen in the product line, the adaption strategy could introduce new features that were not selected in the initial product or change the access control policy. The access control system architecture follows a layered architecture of four layers. The first layer is the access control core functionality layer which contains the product's artifacts with core functionality (e.g., granting or denying access as well as reading data from chips) and driver software. Second, the middleware layer has two interpreters one for the access control model and the other for the configuration model. The third layer is the runtime model layer which has the access control model (e.g., has access control policy), configuration model which has product line features, and state variable which represents sensed value. Finally, the adaption management layer has the adaption strategy to change rules and features. Moreover, rules can be changed manually by the administrator through this layer.

The studies presented above use a family approach in a top-down fashion, where a feature model for the whole system is specified and the policy of a certain device is a single configuration derived from the model or a specific product of the family. On the other hand, in this thesis, we use a family approach in a bottom-up fashion, where we use the resource policies to abstract the whole network and generate policies for internal access control points.

2.5 Access Control Research Topics

Distributed Environment Challenges

A network or a distributed system [TS06, TW10] consists of a collection of autonomous machines that require an advanced access control method. Moreover, enforcing an access control policy in such an environment is a challenge by itself. In this section, we discuss access control in a distributed environment and the research related to this area.

In a distributed environment, users request access to resources. Controlling access to resources is all about protecting those resources from unauthorized access. Protection is enforced by an access control point that stores policies and enforces them as well. Therefore, it is important to assume that the access control point is protected against compromise or unauthorized access.

Implementing access control in distributed systems can be done either using a centralized authority with a single or multiple access control points, distribution of policies to access control points or a hybrid approach of both where the management is centralized while the enforcement is distributed [HFK06]. An example of the first approach is the use of a perimeter firewall at the network's gateway to protect the distributed resources from external access. An example for the second approach is the use of distributed firewalls where each is assigned it is policy separately with no central authority over them. An example of the third approach is using distributed firewalls where there is a centralized unit managing those points and making sure policies are consistent and up to date.

Delessy et al. [DFLPW07] surveyed access control models for distributed systems. These models include high-level policy-based access control such as RBAC and ABAC and low-level implementation languages such as ACL and capability list. Dekker et al. [DCE08] present a formal model for a distributed RBAC. In the model, the policy is distributed among different reference monitors for the different subsystems. Any change or update in the policy is reflected to the affected reference monitors. Abadi et al. [ABLP93] present a

calculus model for delegating authority in access control systems in a distributed environment.

Virtual Machines (VMs) are software implementation of machines that run like isolated physical machines [RAAS14]. The virtualization technology is a software layer on top of the physical layer that allows for running multiple VMs with possibly different operating systems on the same hardware. Virtualization allows for servers' consolidation which helps in reducing operation and administration costs. VMs are gaining popularity and have been adopted by many organizations because of what they provide of flexibility to users and administrators. However, these benefits come with security risks and threats, one of which is managing access control.

Hirano et al. [HOKY07] present an approach to enforce an access control policy based on user identity in distributed virtual machines using secure Virtual Machine Monitor (VMM). An important component of secure VMM is MAC system. It consists of a policy decision point, policy enforcement point, and policy management to update and validate distributed policies. A security policy is distributed by a central policy server. The access control is based on virtual machine ID and user ID. Authentication and authorization are done on the VMM layer rather than the guest operating system using a user ID card.

Later, in [HSE+08] Hirano et al. present an architecture to enforce RBAC access control policy in distributed systems. It enforces the policy on distributed virtual machines using VMM. The ID management framework employs ID card system to authenticate and authorize users in VMM layer. A user is assigned a public key certificate (PKC) as an ID. User's roles are expressed as an attribute certificate (AC). The relationship between PKC and AC indicates the role assignment, both PKC and AC are stored in a user's ID card. Authentication is done based on certificate authority's PKC stored in the VMM layer. After that, authorization of a user is done by verifying its AC based on attributed authority's PKC in the VMM. Relationship between AC and PKC makes VMM decide the role of a user.

In addition to the above, access control policies in distributed virtual machines can be enforced using firewalls, thin-client system, or using VMM technology [HSE+08]. Another mechanism is to use a virtual firewall in the host's virtual machine monitor. Moreover, enabling the firewall of each guest machine is another mechanism. The dynamic nature of virtual machines requires the virtual firewall to be dynamic, flexible and having the ability to configure itself at run time according to the state of the machines [ZLKM11].

Usually, computer networks are distributed. In our work, the overall security policy is distributed to the resources and multiple access control points. Each of these entities enforces their local access control policy and there is a centralized unit that manages the overall policies of these points. We deal with this aspect from a family-based perspective. The distributed nature of the access control policies that we adopt can be observed in the SDN implementation and the prototype presented in Chapter 8 and Appendix B, respectively.

Testing Access Control Systems

One of the issues tackled in the literature of access control is generating test cases to test the implemented policy. Testing is tackled in [HXCL12, Brü12, BBKW10, BBW15] and more.

Hwang et al. [HXCL12] present an approach for firewall policies test generation.

In [Brü12, BBKW10, BBW15], Brucker et al. present a model for access control policies called Unified Policy Framework (UPF). The main goal of UPF is to be used for generating test cases. It is based on the interactive theorem prover Isabelle/HOL. UPF is a generic model that brings a lot of advantages. On the other hand, this comes with the challenge of bringing together different concepts from different domains and the need to be flexible. Security policies are usually dynamic such that their decision depends on the state of the system. Therefore, they model policies, system state, and state transition rather than only decisions.

UPF is based on four principles. First, it represents a policy as a decision function that

grant or deny access. Second, no conflicts are possible within a policy because a resolution strategy is applied when combining the rules of a policy. Third, the decision can be one of three values: allow, deny, or undefined. Fourth, the output is a decision accompanied with additional information. In UPF, a policy is modeled as a partial function that maps an input to a decision. The authors in [BBKW11] present a case study of applying a Model-based Testing (MBT) approach to generate test cases to the National Programme for IT (NPfIT) which is a large-scale project for the IT infrastructure for the National Health Service (NHS) in England. The challenges in modeling such a real-life application included the following: The access control rules for patient information are complex and represent conflicting issues such as confidentiality, usability, function, and legal constraints. Moreover, the security requirement abides by laws, ethical issues, official guidelines which are prone to changes that need to be reflected in the distributed system.

The formalism presented in this thesis is not specifically for test case generation. However, the implementation schemes presented in Section 5.1.2 for DD strategy can be used to generate the policies that should be enforced at an access control point and derive test cases from it.

Access Control Analysis

One of the problems that weaken an access control system and is addressed by analysis is conflicting or inconsistent policies. A conflict happens when two policies or rules cannot be satisfied at the same time. These conflicts can be at a high-level model, at the rules enforced at a single access control point (firewall), or between the rules of distributed access control points (firewalls). There is a lot of research tackling the challenge of classifying and finding these conflicts.

For finding errors in a high-level access control model, Lupu and Sloman [LS97, LS99] present an approach to find and resolve conflicts in a distributed system modeled with RBAC. There are two types of policies: authorization and obligation. Authorization policies specify

what privileges a user is permitted or denied performing while obligation policies specify what a user must or must not do to an object. Conflicts happen between allow and deny authorization policies, between deny authorization and obligation policies. conflicts are resolved by enforcing an execution order or precedence. Moreover, Jayaraman et al. [JGT+11] used an abstraction-refinement and bound model checking technique to find errors in ARBAC policies. In the beginning, the abstraction is extracted from policies then model checking is used to verify the abstracted policy against a set of security properties. Shu et al. [cSYA09] present an approach to find conflicts in ABAC policies in a distributed environment. It is used to find statically conflicting policies rather than evaluation at run time. The approach uses two techniques of rule reduction and binary search to find conflicts. The first is used to reduce rules into compact semantically equivalent rules while the second is used then to find conflicts.

On the other side, there is rich literature on the conflicts of a single firewall (e.g., [ASH03, YCM+06, BCL12, CSFM07, Cha13, HAK12, KH13, SSB14, VSB+15]). Moreover, there is literature on the conflicts between distributed firewalls (e.g., [ASH04, YCM+06, ASHBH05, Cha13, HNVC13, JS09, LS15, VSB+15, WCLC06]). There is also research that tackles finding firewall conflicts in the cloud [Mei15] and SDN [MCD15].

The presented tools above for conflict discovery and resolving are a simplified version of the real-world firewall rules. This is what motivated the work of Diekmann et al. [DHC15, DMHC16] to write a translator from real-world *iptables* to a simplified model that can be used in such tools. The translator is proven in Isabelle/HOL theorem prover.

Recall that a firewall policy is a list of ordered rules that are executed sequentially. Two rules of a firewall are said to be conflicted if they match the same traffic but have different actions. Moreover, conflicts happen between policies of different firewalls in a distributed environment. policies of two firewalls on the same path are conflicted if they have different actions for the same traffic.

In [ASH04], Al-Shaer and Hamed present a classification for the conflicts that can exist in a

single or between distributed firewalls. In a single firewall, the conflicts that may exist are shadowing, correlation, generalization, and redundancy. **Shadowing** is the conflict when the traffic of a lower rule is blocked completely by a proceeding rule with a different action. A shadowed rule is not reachable and never gets executed. **Correlation** is the conflict when two rules are correlated. Correlated rules are two rules that their domains intersect but not completely (i.e., they are not a subset nor a superset of each other) and they have different actions. **Generalization** is the conflict when the traffic of a rule is partially blocked by a proceeding rule with a different action. In this conflict, the lower rule's domain is a superset of the upper rule's domain. **Redundancy** is the conflict when we have a redundant rule. A rule is redundant if its domain is a subset of another rule's domain and having the same action. Such that removing the redundant rule will not result in any change in the policy's behavior.

Between distributed firewalls, the following conflicts can happen. **Shadowing** happens when an upper firewall blocks traffic accepted by a lower firewall. **Spuriousness** conflicts happen when an upper firewall accepts traffic denied by a lower firewall. **Redundancy** happens when a lower firewall denies traffic already blocked by an upper firewall. Finally, **correlation** which happen when we have two correlated rules one in the upper firewall and the other in the lower firewall.

Yuan et al. [YCM+06] have the same set of conflicts in a single firewall. Shadowing and redundancy are regarded as errors while generalization and correlation as warnings. In distributed firewalls, the only conflict considered an error is shadowing. Redundant accept rules are necessary to reach resources. Moreover, redundant deny rules are not necessary but it is not an error since they provide multiple layers of defense which is a good practice to improve security. Another conflict is introduced which is the **cross-path inconsistencies** which happens if there are multiple paths to a resource each has a different action for the same traffic.

Al-Shaer and Hamed [ASH04, AS14] present an algorithm to detect firewall conflicts in a

single firewall and distributed firewalls. Yuan et al. [YCM+06] present a toolkit for firewall modeling and analysis (FIREMAN). The tool uses static analysis techniques to find conflicts in firewall policies. Capretta et al. [CSFM07] present an algorithm to find conflicts between rules in a single firewall. Coq proof assistant is used to discover conflicts.

The formalism presented in this thesis guarantees the absence of conflicts in a single policy due to the integrability of the commands. Moreover, it guarantees the absence of conflicts between the distributed access control points. More discussion on this topic is presented in Section 4.3.

Policy Specification Languages

Defining a high-level specification language has the advantage of being easier to analyze than a low-level language. Low-level rules are derived from these specifications.

Zhang et al. [ZASJ+07, AS14] present a high-level firewall configuration policy language (FLIP). In the language, the firewall policies are defined at a high level using a user interface called policy designator. The high-level rules are then translated to firewall rules and distributed to appropriate firewalls. The language guarantees the absence of conflicts by performing checks.

The eXtensible Access Control Markup Language (XACML) [Ris13] is an informal policy modeling language based on XML for control over the internet. The model defines a policy language along with a process model on how policies should be evaluated to reach a decision for a request. The focus of this model is to combine different policies together which explains the number of combing algorithms it has. XACML mainly is an ABAC model which can be used to model RBAC as well.

The formalism presented in the thesis is a general formalism. It can be used to reason on policies specified in any of the above-mentioned high-level specification languages. Moreover, the use of a family approach allows us to abstract the distributed policies in the network as a family of resource policies enforced at the entry point of the network. The

topic of network abstraction is discussed in Section 4.3.

Generating Low-level Rules from High-level Policies

One of the topics tackled in the literature is the generation or translation from high-level policies to low-level rules to be implemented at the different access control points. This process is also referred to as policy refinement. It has been tacked by studies such as [Hin99, BLMR04, BLR+05, BCV04, CT08, RSC+06, CLL+09, CLL+10].

Adao et al. [AFGL16] present an algorithm that takes security goals and generate the actual configurations of distributed firewalls. Hinrichs [Hin99] described a technique for translation from high-level policy to low-level specific access control point Cisco devices.

The formalism of this thesis is not concerned about generating low-level policies from high-level policies, however, it is general to reason on both. Moreover, in this thesis, starting from the policies of the resources, we generate the policies of the internal access control points up to the entry point of the network.

Conformance of Low-level Rules to High-level Policy

One of the challenges in an access control system is validating that the low-level rules enforced by the access control points actually reflect the high-level policy of the organization [YCM+06].

Youssef and Bouhoula [YB10] present a formal method approach based on inference systems to check the conformance of distributed firewall rules to the global security policy. Moreover, the procedure checks for conflicts between the policies of the distributed firewalls. Tongaonkar et al. [TIS07] generate high-level policies from low-level firewall policies. This makes policies understandable, and easy to resolve errors and improve the rules set. Hachana et al. [HCC+13] present a mining technique to generate RBAC policies from existing firewall rules. The goal is that it is easier to manage RBAC than to manage firewall

rules. Montanari et al. [MCL+13] present a system for monitoring and validating compliance of low-level implemented policies to high-level security policies in network systems. It works by decomposing policies and delegates the validation to multiple machines to avoid single point failure and attacks.

The network abstraction discussed in Section 4.3 can be used to validate the overall policy of the network such as what traffic is denied or allowed by all resources in the network.

Policy Composition

Policies governing access to resources come from different sources and stakeholders covering laws, best practices, and organizational requirements. These policies are specified independently and need to be enforced as one policy at the resource while maintaining their independence. For example, the overall organizational policy needs to be combined with departmental policy. Moreover, resources can be subject to shared or collaborated projects between organizations with their requirements, and therefore their policies need to be composed to the implemented policy at the resource. To carry out the needs of both parties of information availability as well as confidentiality and integrity [SD01, WJ03]. There is a rich literature tacking this issue such as [BdS00, JSSS01, BDS02, WJ03, BDS04, PW05, ZB07, RLB+09, NF18].

Bonatti et al. [BdS00, BDS02] present an algebraic framework for access control policies. The goal for the framework is the composition of policies. Their work is motivated by the need for composing policies that are stated independently to generate a single implementation while maintaining the independence of the original policies. Prior to their work, policy specifications were thought of as a single monolithic and complete specifications. Moreover, one of the goals of their work is to allow policies that are not complete at specification time and the incomplete part will be provided at execution time.

The algebra is defined over ground or atomic terms of subjects, objects, and actions. Policies are sets of ground terms specifying the permitted actions. The algebra defines constraints

and inference rules. Constraints are used to restrict policies to special conditions on the subject, object, action, or combination of those. For example, an organization can define constraints to extract the part of a policy that is concerning a certain department. Inference rules are used to update terms when inference rules are satisfied and can be used, for example, to delegate authority. In the algebra, different composing operators are defined such as addition (union), conjunction (intersection), subtraction, closure, scoping restriction, overriding, and template. Addition is the union of basic terms. Conjunction is the intersection of policies. Subtraction removes ground terms from the first policy that are in the second policy. They argue it is advantageous to use subtraction than to allow for negative authorizations. Closure closes a policy using a set of inference or derivation rules. Scoping restriction uses a set of constraints to restricts a policy. Overriding replaces part of policy P_1 with parts of policy P_2 and P_3 is used to specify the parts to be replaced. Template is used to define partial policies that to be completed.

There are two strategies to evaluate policy specifications one is to generate implementations or ground terms before execution while the other is to evaluate requests at run-time. The first one requires pre calculations and the second is more expensive at run-time. A third approach which is supported by Bonatti et al. work is to generate ground terms for the complete part of the policy and leave the unknown or dynamic part to be evaluated at run-time.

Wijesekera et al. [WJ03] present a propositional policy algebra for the manipulation and composition of access control policies at a high level. Policies are modeled as nondeterministic relations or transformations of permission sets to subjects. Permission sets are pairs of object and permission. permissions are signed to indicate an allowed or denied permissions. Operations on policies are regarded as relational or set theory operators. They defined policy operators such as conjunction (intersection), disjunction (union), difference, negation, sequencing, closure under rules, provisioning, and scoping. Operators come in two flavors, internal and external. For example, internal union merge permission sets creating larger

sets with all possible combinations while external union just add sets together. The focus of such frameworks is that policies are a composition of other policies, however, we go further to use this fact to solve network security issues and define and proof properties.

In case of conflicts in a policy, that is; a policy that gives a positive and negative authorization for the same domain, max and min operators are used to select the positive or negative authorizations, respectively. And in the case of under specified policies, $oCom$ and $cCom$ operators complete the unspecified part of the policies by allowing all access that is not explicitly denied or denying access that is not explicitly allowed, respectively.

In [ZB07], Zhao and Bellovin propose a policy algebra framework to model security policies in a hybrid firewall environment. The policies are defined as a set of rules and operations are defined in the policy and the rules levels (external and internal). The operations are addition, conjunction, subtraction, and summation which is a list of additions. Each policy is associated with cost and risk functions and they have the concept of policy delegation where a resource can delegate the enforcement of its policy or part of it to other resources which in turn reduce cost and increase risk values. Moreover, they used this model on firewall rules. To transfer the sequential firewall rules to an unordered list they used a decorrelation algorithm [HSP00]. And instantiate the internal operations to the firewall rules. They assert that their algebra does not prevent conflicts, it only reveals them. The work extended to the integration of Ponder2 policies in [ZLB08].

Rao et al. [RLB$^+$09] present an algebra for the integration of XACML policies. In this model, a policy is a triple of three values: a set of the allowed request, a set of the denied requests, and a set of not applicable requests. A number of operators are defined including addition which allows the union of what the policies allow and denies what is denied by either policy one but not allowed by the other one, intersection, negation, and domain projection. They also present derived operators such as effect projection, subtraction, and precedence (i.e., sequential composition). They also show that the algebra expresses XACML policies and show its completeness.

Sabri and Hiary [SH16] present an algebraic model based on information algebra to specify and analyze policies. In the model, a policy is specified as an allowed privilege given to a user on an object (i.e., resource). Two ways are presented to combine policies: union and intersection. They differentiate between an elementary and composite policy. They specify enforcing predefined constraints and comparing policies. The model is later extended to RBAC policies [Sab18].

Recently, Neville and Foley [NF18] present a firewall algebra for constructing and reasoning about conflict-free firewall policies. The algebra is based on refinement define operators such as sequential composition, union, and intersection. It is used on *iptables* firewall rules. In the algebra, a filter condition is a tuple $(s, sport, d, dport, p)$ represent the sources IP, source port, destination IP, destination port, and protocol, respectively. A firewall policy is a tuple (A, D) where A are the allowed conditions and D are the denied conditions. A and D are disjoint and implicit default action is defined for the complement of the union of A and D. A policy P refines a policy Q if everything denied by Q is also denied by P and everything that is allowed by P is allowed by Q. The policy with the refinement forms a poset. Policy intersection is the policy that denies any condition denied by either P or Q and allows what they both allow, and therefore is the greatest lower bound w.r.t. refinement. Policy union is the policy that allows anything allowed by P or Q and denies what they both deny, therefore is the least upper bound w.r.t. refinement. They also define sequential composition based on these operators.

Golnabi et al. [GMKA06] present an approach to generate firewall rules by analyzing firewall log files using data mining. Garg et al. [GJD11] present an algorithm that checks the log for compliance with high-level policies.

The algebraic approaches presented above are limited to modeling and analyzing policies. They do not adopt a family approach to model policies.

Adaption to Resource Change, Policy Update, or Threats

As noted, resources and policies are changing continuously which requires access control systems to accommodate and react to these changes. Many research attempts have been made to tackle this issue.

Burns et al. [BCG+01] pointed out the need for an automated solution to reconfigure access control points in response to network changes to uphold a security policy. Moreover, they present a project is to tackle this issue. In [BCL11], Bailey et al. present a self-adaptive framework for distributed RBAC/ABAC policies. The framework depends on loop feedback that monitors the actions performed by users to analyze their behavior. Based on the analysis, it is decided if policies need to be updated.

The formalism presented in this thesis is intended to structure and configure networks either traditional or dynamic. In Chapter 8, we incorporated an algorithm derived using our formalism into an SDN environment as an adaptive component responsible for preserving the secure structure and configuration of networks.

2.6 Software Defined Networks (SDN)

In networking, a plane is an integral component of the telecommunication architecture. There are three components, the data plane, the control plane, and the management plane. Each plane has its functionality. The data plane is responsible for handling packets and performing actions of them. The control plane is responsible for deciding the actions to be taken by the data plane. The management plane is where the configuration and monitoring of the network devices happens.

In traditional networks, the control plane and data plane are combined in the same network device. Therefore, network administrators configure each device separately. This approach makes traditional networks hard to setup, maintain and manage. Moreover, traditional networks cannot cope with the demands of modern networks such as dynamic changes and

scalability. SDN separates or decouples the control plane from the data plane. The control or the management is centralized and has a global view of the network. Data plane devices (e.g., switches) are basically dummy forwarding devices. They forward traffic based on rules specified remotely. The rules could be coming from applications and are triggered by information extracted from traffic or traffic events [SKBJ20].

SDN architecture consists of three planes: application, control, and data planes [SH17, SKBJ20] as shown in Figure 2.2. Each plane has its own specific functionality. An SDN network has at least a single controller at the control plane, a northbound application programming interface (API) between the control plane and the application plane, and a southbound interface between the control and data planes.

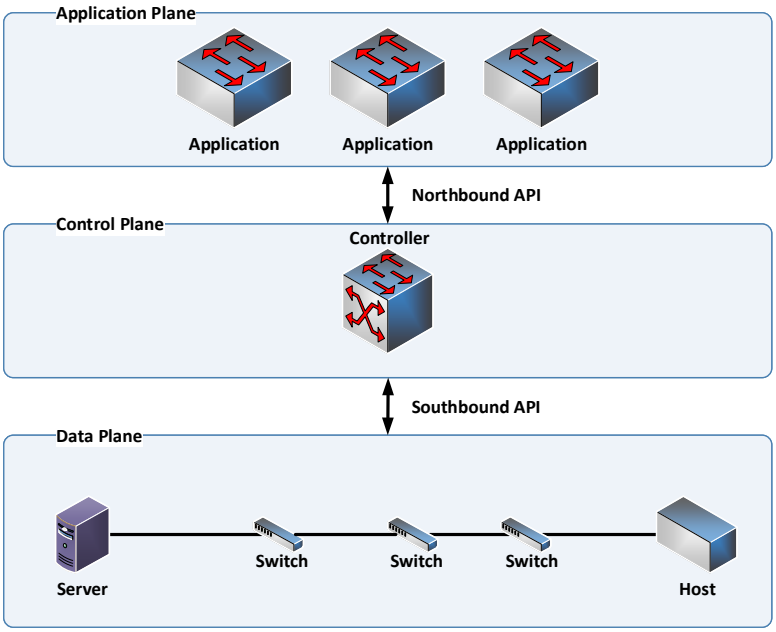


Figure 2.2: SDN architecture

Data plane has network devices (e.g., switches) that forward packets without taking decisions on their own, and they communicate with the controller using southbound APIs (e.g., *OpenFlow* protocol). Each *OpenFlow* enabled switch has a flow table. Each entry

in the flow table has matching fields, an action, and counters. The matching fields can be any of the packet header attributes such as source mac address, source IP, destination mac address, destination IP, destination port, protocol, etc. A flow table entry can execute many actions including forward packet on a specific port, forward packet to controller, or drop. Flow table entries have priority to specify the order of the evaluation. An entry with high priority gets evaluated first, and an entry with low priority is evaluated later. The development of SDN architecture allowed for the introduction of software switches. The most used of which is *open vSwitch* [PPK+].

The southbound API is the interface between control plane and data plane network devices. *OpenFlow* is the most accepted and used southbound API. There are multiple *OpenFlow* messages send from the data plane devices to the controller including *packet-in*, *switch features reply*, *flow remove*, etc. And the *OpenFlow* messages from the controller to data plane switches include *packet-out* and *add flow*.

SDN control plane runs the network operating system (NOS) which has a global view of the network and configures network devices based on policies and commands defined by applications [ANYG15, KMNH19, SKBJ20]. It also abstracts low-level/data plane network for application plane. There are two types of controller architectures: centralized and distributed. The centralized controller architecture has a single controller responsible for managing data plane devices. Examples of centralized controllers are Ryu [KFAS14], POX [ASM], and floodlight [flo]. SDN distributed controller architecture has multiple controllers with interfaces between them. ONOS [BGH+14] adopts an architecture that distributes the controller. For further information on SDN controllers, we refer the reader to [ZKS+19].

2.6.1 Stateful Data Plane

In the original proposal of SDN, data plane switches are stateless. Stateful processing is performed by the controller. For stateful communications, switches forward packets to

the controller which results in controller overhead and increased latency. Stateful data plane approaches are proposed to allow switches to handle state processing locally without the need to forward traffic to the controller and therefore improve the overall network performance [DCA+17, SKBJ20].

OpenState [BBCC14] is a stateful data plane abstraction extending *OpenFlow* that allows stateful processing at the data plane. The motivation behind *OpenState* is that some simple operations can be done based on the switch knowledge and can be delegated to the switches, therefore allowing the controller to focus on global network decisions. The controller is still informed of the delegated operations and in control of the switches. As shown in Figure 2.3, each data plane switch consists of two tables: a state table and an eXtensible Finite State Machine (XFSM) table. Communication states are stored and tracked by the state table. The actions to be taken by the switch for a packet and updating the state table is handled by the XFSM table. The XFSM table is the tabular representation of the Mealy Machine transition function.

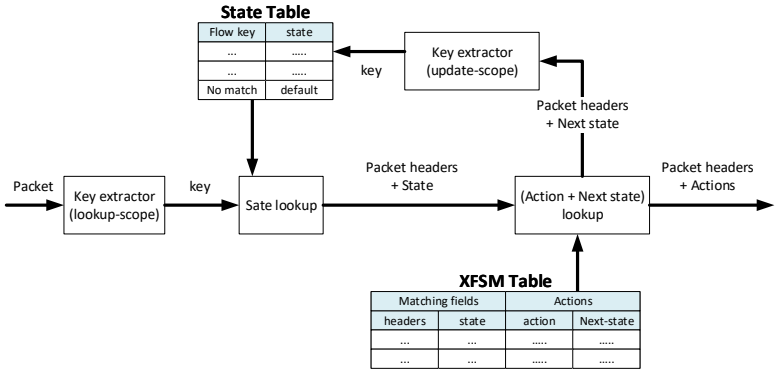


Figure 2.3: *OpenState* packet flow [BBCC14]

There are other approaches (e.g., Flow-level State Transitions (FAST) [MBG+14], Stateful Data Plane Architecture (SDPA) [SBC+17]) that use more than two tables. We refer the reader to [DCA+17] for other approaches for building a stateful data plane.

2.6.2 SDN Architecture

In the literature, there is an emphasis in the research on the whole SDN architecture (e.g., [SSJP17, CYY16]) and control plane structure (e.g., [BGH+14, HSM12]). While there is a large literature dealing with the controller placement at the control plane, the authors of [KMNH19] indicate a shortage of research work on the placement of resources and switches at the data plane. One of the topics discussed in regards to SDN data plane structure is data plane flexibility [KMNH19] which can refer to many issues including the adding and removing of resources (i.e., changing the topology). The issue of micro-segmentation and slicing is also one of the topics discussed in data plane structure [MHS+16]. As far as we know there are no clear approaches in the literature that guides us in placing the resources at the data plane.

2.6.3 Dynamic SDN

The nature of resources in modern networks is dynamic such that resources join and leave the network at any time. This is related to the topic of SDN data plane flexibility discussed above. Moreover, we find rich literature on SDN fault tolerance [YBÖE17]. Faults in SDN can be at the three places: data plane, control plane, and application plane. Failures in the data plane consist of link failure or switch failure, failures in the control plane can be controller failure or controller-switch link failure. Fault tolerance approaches in SDN are merely remedies for unexpected failures in the network. In our approach, we cover unexpected failures as well as changes that are done intentionally that affect the network topology. Moreover, fault-tolerant solutions both protection and restoration approaches focus on finding alternative paths rather than dynamically rearranging the network topology. We also find that many approach dynamic networks by focusing on routing and load-balancing in the control plane [SHBS19, WLGX16] and data plane [dSSdLPDF19]. It means that what is dynamic is the way to handle a packet. However, we are considering

a dynamic topology of the network: Resources get added and removed and the structure of the networks changes by changing the locations of the switches and their policies. The dynamism that we are considering is related to the network topology.

We find in [BDGPM20, KLG14] that SDN paradigm is mostly deployed to static data centers topology. The application of SDN into dynamic networks such as spontaneous Wireless Mesh Network (WMN) [BDGPM20] and Mobile Adhoc Network (MANET) [BK17, KLG14, YQR17] is recently emerging. In [BDGPM20], a middleware at every node is proposed for providing better management of WMN networks as well as for allowing the network to be flexible, dynamic, and scalable. The approach employs multiple SDN controllers; one for each collection of nodes or what is referred to as WMN islands. The focus of these studies is to use SDN in such dynamic networks. However, the focus of our work is to structure network topologies for security.

2.6.4 SDN Firewalls

There are many design architectures in the literature for implementing firewalls in an SDN environment. In these designs, firewalls are either centralized or distributed, stateless or stateful, resides at the control plane or the data plane.

The architectures in [JRR14, KS16, SPLY14] adopt the design of a single centralized stateless firewall resides at the control plane. For example, in [SPLY14] rules are inserted in the firewall individually and given a name by the user through a Command-line interface (CLI), which makes this architecture limited in handling a policy with a huge number of rules. Moreover, in this architecture, all traffic is handled by the firewall at the control plane except for limited deny rules which are inserted in the switch flow tables. This manual approach for entering the rules would have limited usage in a large dynamic network. It has a scalability limitation as all the traffic is assessed by the firewall at the control plane. Moreover, there is a coupling between the data plane switches and the firewall.

The architectures in [PY14, KKSG15, MAN16, TA15] use a design of distributed stateless

firewalls where a firewall application resides at the control plane that uses the data plane switches as distributed firewalls. In [PY14], the firewall application at the control plane assigns a firewall module for each switch. Each firewall inserts deny rules in the switch, such that the switch drops unwanted traffic without forwarding the packets to the controller. When a packet arrives at a switch with no entry in its flow table to handle the packet, it forwards the packet to the firewall at the controller which instructs the switch on how to forward the packet and inserts an entry to handle such packets in the future. The architectures in [KKSG15, MAN16, TA15] while they use only a single firewall at the control plane, they adopt the same approach of inserting deny rules in the switches at the data plane. The above-mentioned approaches use stateless firewalls that are not able to block packets that are not part of an established connection and do not protect from fake packets. Moreover, stateless firewalls block traffic in both directions while in real-world settings it is desired to allow traffic initiated from an internal resource and to allow the outer reply while blocking traffic initiating from an outer resource. Stateful firewalls store and track communication states and therefore able to block packets that are not part of an established connection. Moreover, they are able to allow for traffic to pass from one direction and to be blocked from the opposite direction. The architectures in [GKK16, TA16] use a stateful firewall. Architecture designs that are based on firewall applications at the control plane suffer from multiple issues related to scalability and controller overhead as they rely on forwarding a huge amount of traffic to the controller.

The architectures presented in [CRDP19, CHA$^+$18, ZE18] adopt a design of distributed stateful firewalls at the data plane. They use stateful data plane approaches to allow data plane switches to act as stateful firewalls without the need to forward traffic to the controller for stateful processing. The architecture in [CRDP19] uses *OpenState* [BBCC14] and [CHA$^+$18] uses *Open vSwitch conntrack* module [PPK$^+$] to enable stateful tracking in data plane switches. Such an approach takes the load off the controller and improves latency as traffic is handled at the switches; however, it has a setup and maintenance cost

associated with it as a huge number of rules need to be inserted in the switches.

In the implementation of the results of the formalism into SDN presented in Chapter 8, we propose three architectures: a single and centralized stateful firewall at the control plane, multiple distributed stateful firewall at the control plane, and multiple distributed stateful firewall at the data plane. These architectures cover all the architectures we found in the literature. However, we are only interested in stateful firewalls as indicated above.

2.7 Summary

This thesis tackles the strategies of layered protection and segmentation to structure and configure networks achieving robust and secure network design. I approach these strategies from an access control perspective. Therefore, this chapter started by surveying the literature related to access control policies. Then surveying the literature related to layered protection and network segmentation. Since the formalism presented in this thesis is based on a family approach, the chapter surveyed the literature related to the use of a family approach in security and access control systems. Afterward, it surveyed topics related to network access control. Finally, because the proposed formalism is implemented in SDN, the chapter introduced the literature related to SDN.

The formalism presented in this thesis is based on multiple assumptions. First, a policy is a special kind of program and therefore can be modeled using guarded commands. Second, the language is generic and not tied to a specific policy or firewall language. Third, the policies are enforced by distributed access control points and by a centralized management unit.

Chapter 3

Mathematical Background

This chapter introduces the mathematical background required for the rest of the thesis. In particular, Section 3.1 presents an illustrative example that is used throughout the thesis to relate the theory and concepts to the real-world network security context. Section 3.2 gives the necessary background on relations. It also presents tabular expressions as a means to represent policies and automate their analysis and transformation. Section 3.3 introduces guarded commands which are used to model access control policies as relations. Section 3.4 introduces PFA which is the basis for dealing with access control policies. Finally, Section 3.5 summarizes the chapter.

3.1 Illustrative Example

In this section, we present an example to illustrate the concepts and theory discussed throughout the rest of the thesis. We seek simplicity in presenting the example here and the concepts introduced later. Consider an organization with different departments, in which resources are connected in a network. We consider the following small subset of resources: A *Web server*, an *Email server*, two workstations belonging to the Engineering department (i.e., *Engineering1* and *Engineering2*), two workstations and a database belonging to the

Finance department (i.e., *Finance1*, *Finance2*, and *Finance DB*), and an internal *File server* available to all departments.

The above resources have different security requirements governing their access. These requirements stated in a high-level natural language as follows: *Web server* and *Email server* allow outer users for only the designated services (i.e., web pages and email) and deny any other services. For internal users, they allow full access. *File server* allows every user from inside the organization and blocks everyone else. Finance and engineering workstations allow only users belonging to their department and deny everyone else.

The requirements are translated to a network requirement as follow: all resources allow an established or related connection and drop every invalid packet. The *Web server* and *Email server* allow every HTTP (`TCP` port 80) and SMTP (`TCP` port 25) connection from any source, any access from all internal resources, and drop everything else. The *File server* rejects access requests made by the *Web server* and *Email server*, allows all internal resources, and drops every other access requests. The resources belonging to the engineering and finance departments have stricter policies, which allow access only by users and resources within their respective department, reject all requests made by any other internal resource, and drops every other access requests. These security requirements are written in the language of *iptables* and are given in Figures 3.1, 3.2, 3.3, and 3.4.

```
1 -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
2 -A INPUT -m state --state INVALID -j DROP
3 -A INPUT -p tcp -m state --state NEW -m tcp --dport 80 -j ACCEPT
4 -A INPUT -p tcp -m state --state NEW -m tcp --dport 25 -j ACCEPT
5 -A INPUT -s 192.168.1.0/24 -j ACCEPT
6 -A INPUT -s 192.168.2.0/24 -j ACCEPT
7 -A INPUT -s 192.168.3.0/24 -j ACCEPT
8 -A INPUT -s 192.168.4.0/24 -j ACCEPT
9 -A INPUT -j DROP
```

Figure 3.1: Web server policy

```
1 -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
2 -A INPUT -m state --state INVALID -j DROP
3 -A INPUT -s 192.168.1.0/24 -j ACCEPT
4 -A INPUT -s 192.168.2.0/24 -j ACCEPT
5 -A INPUT -s 192.168.3.0/24 -j ACCEPT
6 -A INPUT -s 192.168.4.0/24 -j REJECT
7 -A INPUT -j DROP
```

Figure 3.2: File server policy

```
1 -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
2 -A INPUT -m state --state INVALID -j DROP
3 -A INPUT -s 192.168.1.0/24 -j REJECT
4 -A INPUT -s 192.168.2.0/24 -j ACCEPT
5 -A INPUT -s 192.168.3.0/24 -j REJECT
6 -A INPUT -s 192.168.4.0/24 -j REJECT
7 -A INPUT -j DROP
```

Figure 3.3: Engineering workstation policy

```
1 -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
2 -A INPUT -m state --state INVALID -j DROP
3 -A INPUT -s 192.168.1.0/24 -j ACCEPT
4 -A INPUT -s 192.168.2.0/24 -j REJECT
5 -A INPUT -s 192.168.3.0/24 -j REJECT
6 -A INPUT -s 192.168.4.0/24 -j REJECT
7 -A INPUT -j DROP
```

Figure 3.4: Finance workstation policy

3.2 Brief Overview on Relations

This section presents briefly the necessary background on relations which can be found in many books on discrete mathematics (e.g., [GS93, Ros02]).

Definition 3.2.1 (Relation — e.g., [Khe08])**.** *Let Σ be a set of states. A (binary) relation R over Σ is a subset of the Cartesian product of $\Sigma \times \Sigma$.* ■

Let Q and R be relations on Σ, we present the following operations:

Empty Relation	$\emptyset \stackrel{\text{def}}{=}$	$\{(a, a') \mid \mathsf{false}\}$
Universal Relation	$\mathbb{L} \stackrel{\text{def}}{=}$	$\{(a, a') \mid \mathsf{true}\}$
Complement	$\overline{R} \stackrel{\text{def}}{=}$	$\{(a, a') \mid (a, a') \notin R\}$
Converse	$R^{\smile} \stackrel{\text{def}}{=}$	$\{(a, a') \mid (a', a) \in R\}$
Union	$R \cup Q \stackrel{\text{def}}{=}$	$\{(a, a') \mid (a, a') \in R \vee (a, a') \in Q\}$
Intersection	$R \cap Q \stackrel{\text{def}}{=}$	$\{(a, a') \mid (a, a') \in R \wedge (a, a') \in Q\}$
Composition	$R;Q \stackrel{\text{def}}{=}$	$\{(a, a'') \mid \exists(a' \mid a' \in \Sigma : (a, a') \in R \wedge (a', a'') \in Q)\}$
Domain	$dom(R) \stackrel{\text{def}}{=}$	$\{a \mid \exists(a' \mid a' \in \Sigma : (a, a') \in R)\}$
Range	$ran(R) \stackrel{\text{def}}{=}$	$\{a' \mid \exists(a \mid a \in \Sigma : (a, a') \in R)\}$

In this thesis, and to proceed with the model for policies, two relational operations need to be introduced: the demonic join $\sqcup$ and the demonic meet $\sqcap$.

A relation Q is said to refine relation R written as $Q \sqsubseteq_r R$ iff $((Q \cap R;\mathbb{L} \subseteq R) \wedge (R;\mathbb{L} \subseteq Q;\mathbb{L}))$ or, equivalently, iff $((Q \cup \overline{Q;\mathbb{L}} \subseteq R \cup \overline{R;\mathbb{L}}) \wedge (\overline{Q;\mathbb{L}} \subseteq \overline{R;\mathbb{L}}))$ [Khe08]. In other words, a relation Q refines a relation R if Q agrees with R on whatever in the domain of R and that the domain of R is a subset of the domain of Q.

Definition 3.2.2 (Demonic join of relations— e.g., [Khe08, DBS$^+$95])**.** *The demonic join of two relations R and Q is defined as $R \sqcup Q \stackrel{\text{def}}{=} (R \cup Q) \cap R;\mathbb{L} \cap Q;\mathbb{L}$. We also have $(R \sqcup Q);\mathbb{L} = R;\mathbb{L} \cap Q;\mathbb{L}$* ■

In other words, the demonic join of two relations is the union of their common domain. For example, for a set $K = \{(1, 2), (1, 3), (2, 5)\}$ and $L = \{(1, 4), (1, 5), (3, 6)\}$ their demonic join $K \sqcup L = \{(1, 2), (1, 3), (1, 4), (1, 5)\}$. So, we can say $(a, a') \in (K \sqcup L) \iff ((a, a') \in K \vee (a, a') \in L) \wedge a \in dom(K) \wedge a \in dom(L)$.

Definition 3.2.3 (Demonic meet of relations – [Khe08, DFKM98])**.** *Let R and Q be two relations on Σ. The demonic meet of R and Q denoted by $R \sqcap Q$ is equal to $R \sqcap Q \stackrel{\text{def}}{=} (R \cap Q) \cup (R \cap \overline{Q;\mathbb{L}}) \cup (Q \cap \overline{R;\mathbb{L}})$ only when $R;\mathbb{L} \cap Q;\mathbb{L} = (R \cap Q);\mathbb{L}$.* ■

The condition $R;\mathbb{L} \cap Q;\mathbb{L} = (R \cap Q);\mathbb{L}$ can be written as $dom(R \cap Q) \subseteq dom(R) \cap dom(Q)$. It means that R and Q has to map every element in their common domain to at least one

common element in the range. For example, for the relation $S = \{(1,2),(1,3),(2,4)\}$ and $T = \{(1,2),(1,4),(3,5)\}$ satisfy the condition by having 2 in their range for the common domain 1. Therefore, $S \sqcap T = \{(1,2),(2,4),(3,5)\}$. While for the relations K and L in the above example, their demonic meet is not defined. For more details on relations and their demonic operations, the reader is referred to [BKS97] and [SS93].

Let S be a set, and let $+$ and $\cdot$ be two binary operations on S (i.e., $+, \cdot : S \times S \to S$), and $\leq$ be a partial order. We say that:

- The operation $\cdot$ is associative $\iff (\forall\, x, y, z \mid x, y, z \in S \cdot x \cdot (y \cdot z) = (x \cdot y) \cdot z\,)$.
- The operation $\cdot$ is commutative $\iff (\forall\, x, y \mid x, y \in S \cdot x \cdot y = y \cdot x\,)$.
- The operation $\cdot$ is idempotent $\iff (\forall\, x \mid x \in S \cdot x \cdot x = x\,)$.
- The element 1 is the identity element for $\cdot \iff (\forall\, x \mid x \in S \cdot x \cdot 1 = x = 1 \cdot x\,)$.
- The element 0 is an annihilator for $\cdot \iff (\forall\, x \mid x \in S \cdot x \cdot 0 = 0 = 0 \cdot x\,)$.
- The operation $\cdot$ right-distributes over the operation $+ \iff (\forall\, x, y, z \mid x, y, z \in S \cdot (x + y) \cdot z = x \cdot z + y \cdot z\,)$.
- The operation $\cdot$ left-distributes over the operation $+ \iff (\forall\, x, y, z \mid x, y, z \in S \cdot x \cdot (y + z) = x \cdot y + x \cdot z\,)$.
- The operation $\cdot$ is right-isotone with respect to $\leq \iff (\forall\, x, y, z \mid x, y, z \in S \cdot x \leq y \implies x \cdot z \leq y \cdot z\,)$.
- The operation $\cdot$ is left-isotone with respect to $\leq \iff (\forall\, x, y, z \mid x, y, z \in S \cdot x \leq y \implies z \cdot x \leq z \cdot y\,)$.
- The relation $\leq$ is reflexive $\iff (\forall\, a \mid a \in B \cdot a \leq a\,)$.
- The relation $\leq$ is antisymmetric $\iff (\forall\, a, b \mid a, b \in B \cdot a \leq b \wedge b \leq a \implies a = b\,)$.
- The relation $\leq$ is transitive $\iff (\forall\, a, b, c \mid a, b, c \in B \cdot a \leq b \wedge b \leq c \implies a \leq c\,)$.

3.2.1 Tabular Expressions

Relations are used in documenting software requirements and specifications [JPZ97, JK01, JP10]. However, such relations tend to be long and tedious which lead to the use of *tabular expressions* to improve their readability and ease calculations on relations. Tabular expressions [JK01] are mathematical expressions in tabular form. They are easier to read and comprehend and therefore bring a practical advantage to mathematical expressions. Moreover, they aid in the automation and verification of mathematical expressions. They have been proven to be effective and used in the documentation of the requirements of many projects such as the A-7E aircraft for the U.S. Navy and the Darlington Nuclear Power Generation in Ontario, Canada [JPZ97]. Tabular expressions have been formalized by Parnas [JP10]. Parnas formalized them in multiple classes. A single class presented in [JK01] that covers all classes. We use one class of tables called *normal tables*.

In this thesis, policies and rules are modelled using guarded commands which are essentially relations, as will be discussed in the next section. Therefore, tabular expressions can be used to represent them.

A table is an organized sets of cells, where each cell contains a mathematical expression [JK01]. The table contains a collection of headers and a grid indexed by these headers. A header H, is an indexed set of cells, $H = \{h[i]|i \in I\}$, where $I = \{1, 2, \ldots, k\}$ is a set of indexes and $k \in \mathbb{N}$. A grid G is a set of cells indexed by headers $H_1, \ldots, H_n$ where $n \in \mathbb{N}$ and $H_j = \{h^j[i]|i \in I^j\}$, $j = 1, \ldots, n$ is a set of cells indexed by I^j, therefore, $G = \{g[\alpha]|\alpha \in I\}$ where $I = \Pi_{i=1}^{n} I^i$. The set I is the index of G. A collection of headers $H_1, \ldots, H_n$ and a grid G indexed by them is the table skeleton. A table is intended to represent a relation (a policy in the context of this thesis) R which is composed of simpler relation (rules) $R[\alpha], \alpha \in I$. Every $R[\alpha]$ is specified by one grid cell $g[\alpha]$ and header cells $h^j[\alpha|j]$, where $\alpha|j$ is the jth element of α. Table 3.1 shows a table skeleton. $H_1 = \{h^1[i]|i = 1, 2, 3\}$, $H_2 = \{h^2[i]|i = 1, 2\}$ and $G = \{g[i, j]|i = 1, 2, 3 \wedge j = 1, 2\}$. Therefore, $R[2, 2]$ is defined

by the expressions in $g[2,2]$, $h^1[2]$, and $h^2[2]$. Every relation (rule) $R[\alpha]$ is defined of the form: **if** P_α **then** E_α, where P_α is a predicate that defines the domain of the relation (i.e., starting states) and E_α is a predicate that defines the range of the relation (i.e., the ending states). Note that in our case, P_α is split between headers and the grid holds E_α.

	$h^1[1]$	$h^1[2]$	$h^1[3]$
$h^2[1]$	$g[1,1]$	$g[2,1]$	$g[3,1]$
$h^2[2]$	$g[1,2]$	$g[2,2]$	$g[3,2]$

Table 3.1: Schematic table of a tabular expression

Consider the relation R:

$$R = \{\big((x,y),(x',y')\big) \mid (x >= 0 \wedge y >= 0 \wedge y' = x) \vee (x < 0 \wedge y < 0 \wedge y' = y)\}$$

We represent R as a two-dimensional tabular expression as shown in Table 3.2. This table has headers H_1 and H_2 to represent the domain of the relation. It has the grid G for the range of R. The relation $R[1,1]$ is defined by the expressions in $g[1,1]$, $h^1[1]$, and $h^2[1]$ as follows: $R[1,1] = \{\big((x,y),(x',y')\big) \mid (x >= 0 \wedge y >= 0 \wedge y' = x)\}$. The relation $R[2,2]$ is defined by the expressions in $g[2,2]$, $h^1[2]$, and $h^2[2]$ as follows: $R[2,2] = \{\big((x,y),(x',y')\big) \mid (x < 0 \wedge y < 0 \wedge y' = y)\}$.

	$x >= 0$	$x < 0$
$y >= 0$	$y' = x$	false
$y < 0$	false	$y' = y$

Table 3.2: R as a tabular expression

Note that in a tabular expression, all table entries represent relations. For example, in Table 3.2, the first cell in H_1 represent the relation $h^1[1] = \{\big((x,y),(x',y')\big) \mid x >= 0\}$. And the cell in the first row of the first column in G represent the relation $g[1,1] = \{\big((x,y),(x',y')\big) \mid$

$y' = x\}$. In fact, as shown in [Wu01], the whole table can be interpreted as the below relation,

$$T \stackrel{\text{def}}{=} \bigcup_{i=1}^{n} \bigcup_{j=1}^{m} H_1[i] \cap H_2[j] \cap G[i,j],$$

where n and m are the number of cells in H_1 and H_2, respectively. Which unfolds to the relation R.

$$R = \{\big((x,y),(x',y')\big) \mid$$
$$(x >= 0 \wedge y >= 0 \wedge y' = x)$$
$$\vee\ (x >= 0 \wedge y < 0 \wedge \mathsf{false})$$
$$\vee\ (x < 0 \wedge y >= 0 \wedge \mathsf{false})$$
$$\vee\ (x < 0 \wedge y < 0 \wedge y' = y)\}$$

As an example of a policy represented using tabular expressions, Table C.2 is the tabular expression of the *File server* policy shown in Figure 3.3. The table has two headers, H_1 which specifies the source IP numbers and protocols and H_2 which specifies the states and destination ports, and an internal grid G which specifies actions.
For the tabular expression representation of all the resource policies for the illustrative example, see appendix C. And for more details on tabular expressions, we refer the reader to [JK01].

Tabular Expressions to Calculate Demonic Join In this section, we demonstrate the use of tabular expressions to calculate the demonic join tabular expression T of tabular expressions T_1 and T_2. To construct T, we use the procedure ComputeDemonicJoin, which computes the demonic join defined in Section 3.3.
Similar to what we find in [Wu01] for calculating the demonic meet, this procedure, first transforms the two input tables T_1 and T_2 to tables T_1' and T_2' such that both have identical

```
procedure ComputeDemonicJoin(T1, T2)
    Transform T1 and T2 to T1' and T2' having identical headers.
    TH1 = H1'                          ▷ first header for T = the first header of T1' and T2'
    TH2 = H2'                          ▷ second header for T = the second header of T1' and T2'
    for all i between 1 and the number of cells in H1' do
        for all j between 1 and the number of cells in H2' do
            if G1'[i, j] ≠ false ∧ G2'[i, j] ≠ false then
                G[i, j] = G1'[i, j] ∪ G2'[i, j]
            else G[i, j] = false
            end if
        end for
    end for
    return T
end procedure
```

Figure 3.5: This procedure takes two tables T_1 and T_2 as input and returns their demonic join table T

headers. This transformation possibly gives rise to new cells in both tables. For the cells for which no action is specified, we set their values to $false$ – which indicates that no rule exists corresponding to such cells in the policy. We then construct the demonic join table T by first having its headers identical to the headers of tables T_1' (or T_2' as they have same headers). Then, we compute Grid G of T as follows: Loop through each cell in G; if the corresponding cells in T_1' and T_2' are not set to $false$, then set its value equal to their union; otherwise set it to $false$.

As an example, consider Table D.7 which is the demonic join of the finance workstations and database policy (Table C.3) and the engineering workstations policy (Table C.4) represented as tabular expressions. Since the two tables have identical headers the procedure ComputeDemonicJoin skips the first step. It constructs the demonic join table by first having identical headers as those seen in Table C.3 (Table C.4). Then Grid G of this table is computed as follows by the procedure: it loops through each cell of G starting with the first one. Since the cell $G[1, 1]$ in Table C.3 and Table C.4 both have a value of $a' =$ `ACCEPT`,

then the cell $G[1,1]$ in Table D.7 is set to $a' = \texttt{ACCEPT}$. While computing G, if the corresponding two tables' cells have different values as seen in the cell $G[1,5]$ of Table C.3 with value $a' = \texttt{ACCEPT}$, and Table C.4 with value $a' = \texttt{REJECT}$, then the value at this cell is the union of their values; that is, the value at Table D.7 is $a' = \texttt{ACCEPT} \vee a' = \texttt{REJECT}$. All the other cells are computed analogously.

3.3 Guarded Commands

In this section, we present a variant of Dijkstra's guarded commands which are used to model access control policies. The following material on guarded commands is based on [MS06, HKM11b]. Essentially, a command is a transition relation from starting states to possible ending states, and a set of states that do not lead to failure.

Definition 3.3.1 (Command — e.g., [HKM11b])**.** *Consider a set Σ of states; the exact nature of its elements does not matter.*

1. *A command over Σ is a pair (R, P) where $R \subseteq \Sigma \times \Sigma$ is a transition relation and P is a subset of Σ.*

2. *The restriction of a transition relation $R \subseteq \Sigma \times \Sigma$ to a subset $Q \subseteq \Sigma$ is ${}_Q{\downarrow}R \stackrel{\text{def}}{=} R \cap (Q \times \Sigma)$.*

The set P is intended to characterize those states from which the command cannot lead to abortion/failure. ■

Hence, a command is modelled as a pair (R, P) to avoid the possibility of leading to abortion [Par83, Par97].

Definition 3.3.2. *The following are basic commands and command-forming operators that correspond to program constructs [HKM11b].*

1. *The command* abort *is the one that offers no transitions and does not exclude abortion of any state:* $\mathsf{abort} \stackrel{\text{def}}{=} (\emptyset, \emptyset)$.

2. *The command* skip *does not do anything; it leaves the state unchanged and cannot lead to abortion for any state:* $\mathsf{skip} \stackrel{\text{def}}{=} (\mathbb{I}, \Sigma)$, *where* $\mathbb{I} \stackrel{\text{def}}{=} \{(s, s) \mid s \in \Sigma\}$ *is the identity relation on states.*

3. *The command* fail *does not offer any transition but guarantees that no state may lead to abortion:* $\mathsf{fail} \stackrel{\text{def}}{=} (\emptyset, \Sigma)$.

4. *The command* chaos *is unpredictable:* $\mathsf{chaos} \stackrel{\text{def}}{=} (\Sigma \times \Sigma, \emptyset)$.

■

Definition 3.3.3 (e.g., [HKM11b])**.** *Let* $C = (R, P)$ *and* $D = (S, Q)$ *be commands. The command* $C \,[]\, D$ *is intended to behave as follows. For a starting state* s*, non-deterministically a transition under* R *or* S *is chosen (if there is any). Absence of aborting is guaranteed for* s *iff it can be guaranteed under both* C *and* D*, i.e., iff* $s \in P \cap Q$*. The command* $[]$ *is defined as:* $(R, P) [] (S, Q) \stackrel{\text{def}}{=} (R \cup S, P \cap Q)$. ■

The operation $[]$ is associative, commutative, and idempotent and the command fail is the neutral element. The placement for the union in the first part and intersection in the second is due to the fact that if the choice of starting states gets greater, then the set of states which no abortion is guaranteed gets smaller.

Definition 3.3.4 (Feasible Commands — e.g., [HKM11b])**.** *Let* (R, P) *be a command, then it is feasible when* $P \subseteq dom(R)$. ■

Definition 3.3.4 is used to distinguish subclass of commands which guarantee the absence of abortion only for those states that offer transitions.

Definition 3.3.5 (Guarded Commands — e.g., [HKM11b])**.** *For the command* (R, P) *and* $Q \subseteq \Sigma$*, the guarded command* $Q \longrightarrow (R, P)$ *(where* Q *is called the guard) is defined as:* $Q \longrightarrow (R, P) \stackrel{\text{def}}{=} ({}_{Q}{\downarrow}R, \overline{Q} \cup P)$*, where* $\overline{Q}$ *is the complement of* Q *w.r.t.* Σ. ■

For the set of states from which the guarded command does not lead to abortion, we augment the set P by the complement of Q; that is, by $\overline{Q}$. The reason is that outside of the set of states Q (i.e., $\overline{Q}$) the command cannot be executed and therefore there is no possibility of abortion or failure. In a starting state, the guarded command $Q \longrightarrow (R, P)$ can lead to a transition only if the starting state is in Q and the domain of R. Abortion is excluded if the state is not in Q or P. Hence, $Q \longrightarrow (R, P)$ is not a feasible command even if (R, P) is. This is solved using the if_fi-statement.

Definition 3.3.6 (if_fi-statement — e.g., [HKM11b])**.** *The* if_fi*-statement for a command* (R, P) *is defined by:* $\mathsf{if}\ (R, P)\ \mathsf{fi} \stackrel{\text{def}}{=} (R, P \cap \mathsf{dom}(R))$. ■

The command is surrounded with if_fi to convert it to a feasible command.

When modelling access control policies using guarded commands. The guard ensures that no changing of the state of the system is done unless the condition of the rule is met. The state of the system changes according to the transition relation of the command.

For a firewall rule attributes $S, P, \mathrm{St}, \mathrm{Dport}$, and A, where S is the set of all possible source IP numbers, P is the set of all possible protocols, St is the set of all possible connection states, Dport is the set of all possible destination ports, and A is the set of all possible actions. $\mathbb{L}$ denotes the universal relation on this space. Then we have $\Sigma = S \times P \times \mathrm{St} \times \mathrm{Dport} \times A$. Then, for example, the rule given in Line 5 in Figure 3.1, which we call C can be written as follows:

$$C = [Q \longrightarrow (R, P)], \text{ where}$$

$Q \subseteq \Sigma$ is the guard and written as follows:

$$\{(s, p, st, dport, a) \mid\ s = 192.168.1.0/24\}.$$

The relation R is written as:

$$R = \{\big((dr, s, p, st, ds, a), (dr', s', p', st', ds', a')\big) \mid a' = \texttt{ACCEPT}\},$$

and we take the set of states that does not lead to abortion as $P = \emptyset$; this states that since the domain of R is Σ, the relation does not guarantee the absence of the abortion for any the sates in its domain (i.e., Σ).

Then, the guarded command $Q \longrightarrow (R, P)$ is equal to $({}_Q\!\downarrow\!R, \overline{Q} \cup P) = (R \cap (Q \times \Sigma), \overline{Q} \cup \emptyset) = (R \cap (Q \times \Sigma), \overline{Q}) = (R \cap (Q \times \Sigma), \overline{Q})$. Therefore, the first element of the tuple of the command C can be written as:

$$R' = \{\big((s, p, st, dport, a), (s', p', st', dport', a')\big) \mid (s = 192.168.1.0/24 \wedge a' = \texttt{ACCEPT})\}.$$

And the second element of the tuple $P' = \overline{Q}$ states that with the guard, the states that are guaranteed not to lead to abortion are the states in the complement of the guard. And the command C corresponding to the rule is written as $C = (R', P')$.

The policy of the *Web server* presented in Figure 3.1 can be represented as a set of rules or as a single rule as follow:

$$\begin{aligned}
&R_{web} = \{((s, p, st, dport, a), (s', p', st', dport', a')) \mid \\
&(st \in \{\texttt{RELATED}, \texttt{ESTABLISHED}\}) \wedge a' = \texttt{ACCEPT} \\
&\vee (st \notin \{\texttt{RELATED}, \texttt{ESTABLISHED}, \texttt{NEW}\} \wedge a' = \texttt{DROP}) \\
&\vee (p = \texttt{TCP} \wedge st = \texttt{NEW} \wedge dport \in \{80, 25\} \wedge a' = \texttt{ACCEPT}) \\
&\vee (s \in \{192.168.1.0/24, 192.168.2.0/24, 192.168.3.0/24, 192.168.4.0/24\} \\
&\wedge a' = \texttt{ACCEPT}) \\
&\vee (s \notin \{192.168.1.0/24, 192.168.2.0/24, 192.168.3.0/24, 192.168.4.0/24\} \\
&\wedge p \neq \texttt{TCP} \wedge st = \texttt{NEW} \wedge a' = \texttt{DROP}) \\
&\vee (s \notin \{192.168.1.0/24, 192.168.2.0/24, 192.168.3.0/24, 192.168.4.0/24\} \\
&\wedge p = \texttt{TCP} \wedge st = \texttt{NEW} \wedge dport \notin \{80, 25\} \wedge a' = \texttt{DROP})\}
\end{aligned}$$

Definition 3.3.7 (Refinement on Commands — e.g., [MS06, HKM11b])**.** *The refinement relation on commands is defined as* $(R, P) \sqsubseteq (S, Q) \overset{\text{def}}{\Leftrightarrow} Q \subseteq P \wedge {}_{Q}\downarrow R \subseteq S$ ■

The refinement relation is reflexive, transitive, and not antisymmetric. The equivalence relation $\equiv$ associated with $\sqsubseteq$ is defined as $C \equiv D \iff^{\text{def}} C \sqsubseteq D \wedge D \sqsubseteq C$. Therefore, the equivalence relation on commands is defined as: $(R, P) \equiv (S, Q) \iff^{\text{def}} P = Q \wedge {}_{P}\downarrow R = {}_{P}\downarrow S$. The equivalence relation partition a set into equivalent classes. All the elements of a class are equivalent. Two classes are related by $\sqsubseteq$ if their class representatives are. Therefore, there is a partial order on equivalence classes of commands.
For example, for the relations R and R' presented above, we have $R \sqsubseteq R'$ as the domain of R' is a subset of the domain of R and restricting R to the domain of R' is equal to R'.

Definition 3.3.8 (Demonic meet — e.g., [DBS+95, HKM11b])**.** *The greatest lower bound of commands* (R, P) *and* (S, Q) *w.r.t.* $\sqsubseteq$ *is the demonic meet which is defined as* $(R, P) \sqcap (S, Q) = ((R \cap S) \cup ({}_{\overline{P}}\downarrow S) \cup ({}_{\overline{Q}}\downarrow R), P \cup Q)$. ■

The $\sqcap$ operation is commutative, associative, and the command abort is its neutral element. Let R and S be two relations and let the commands $(R, dom(R))$ and $(S, dome(S))$ be two feasible commands. The meet of these two commands is said to be feasible iff $dom(R \cap S) = dom(R) \cap dom(S)$. In other words, the meet is feasible if and only if R and S give the same actions for their common domain. This allows for the integration of R and S or what is called **integrability** property.

Definition 3.3.9 (Integrability — e.g., [HKM11b])**.** *Two relations* R *and* S *are* ***integrable*** *iff* $dom(R \cap S) = dom(R) \cap dom(S)$. ■

Let R_1 and R_2 be the relations of policies P_1 and P_2 respectively:

$$R_1 = \{\big((s, p, st, dport, a), (s', p', st', dport', a')\big) \mid$$

$$(s = 192.168.1.0/24 \wedge a' = \texttt{ACCEPT})$$

$\vee\ (s = 192.168.2.0/24 \wedge a' = \texttt{REJECT})$

$\vee\ (s = 192.168.3.0/24 \wedge a' = \texttt{ACCEPT})\}$

$R_2 = \{\big((s, p, st, dport, a), (s', p', st', dport', a')\big)\ |$

$(s = 192.168.1.0/24 \wedge a' = \texttt{ACCEPT})$

$\vee\ (s = 192.168.2.0/24 \wedge a' = \texttt{ACCEPT})$

$\vee\ (s = 192.168.4.0/24 \wedge a' = \texttt{ACCEPT})\}$

Then, the demonic meet of R_1 and R_2 is as follows:

$R_1 \sqcap R_2 = \{\big((s, p, st, dport, a), (s', p', st', dport', a')\big)\ |$

$(s = 192.168.1.0/24 \wedge a' = \texttt{ACCEPT})$

$\vee\ (s = 192.168.3.0/24 \wedge a' = \texttt{ACCEPT})$

$\vee\ (s = 192.168.4.0/24 \wedge a' = \texttt{ACCEPT})\}$

The demonic meet $R_1 \sqcap R_2$ in the above example is not feasible. The reason for this is that R_1 and R_2 do not agree on the action for the common domain (i.e., $s = 192.168.2.0/24$). Therefore, we can say R_1 and R_2 are not integrable.

Definition 3.3.10 (Demonic Join — e.g., [DBS+95])**.** *The least upper bound of commands* (R, P) *and* (S, Q) *w.r.t.* $\sqsubseteq$ *is the demonic join which is defined as* $(R, P) \sqcup (S, Q) = ({}_{(P \cap Q)}\!\downarrow (R \cup S),\ P \cap Q)$. ■

The demonic join coincides with the Greatest Common Divisor (GCD) of policies discussed in Chapter 4.

Then, the demonic join of R_1 and R_2 is as follows:

$R_1 \sqcup R_2 = \{\big((s, p, st, dport, a), (s', p', st', dport', a')\big)\ |$

$(s = 192.168.1.0/24 \wedge a' = \texttt{ACCEPT})$

$\vee\ (s = 192.168.2.0/24 \wedge (a' = \texttt{ACCEPT} \vee a' = \texttt{REJECT}))\}$

Note that the operator $\sqcup$ is commutative and associative and has fail as its neutral element. Also, $\sqcup$ and $\sqcap$ distribute over each other and therefore the commands form a distributive lattice.

For further discussion on guarded commands, we refer the reader to [MS06, HKM11b].

3.4 Product Family Algebra

Product family algebra (PFA) [HKM11a, HKM06] is a mathematical structure of product families. It is based on the mathematical structure of idempotent and commutative semiring. It aids in the capture and analysis of the commonalities and variabilities of product families. Moreover, it allows for mathematical description and manipulation of product family specifications. In the following, we introduce the mathematical foundations of PFA.

Definition 3.4.1 (Semigroup — e.g., [GS93, Hun80])**.** *A semigroup is an algebraic structure* $(S, \cdot)$ *such that* S *is a set, and* $\cdot$ *is an associative binary operation. A semigroup is commutative if* $\cdot$ *is commutative, and idempotent if* $\cdot$ *is idempotent.* ■

Definition 3.4.2 (Monoid — e.g., [GS93, Hun80])**.** *A monoid is an algebraic structure* $(S, \cdot, 1)$*, such that* $(S, \cdot)$ *is a semigroup and* 1 *is the identity element. A monoid is commutative if* $\cdot$ *is commutative, and idempotent if* $\cdot$ *is idempotent.* ■

Definition 3.4.3 (Semiring — e.g., [HW98])**.** *A semiring is an algebraic mathematical structure* $(S, +, 0, \cdot, 1)$*, such that* $(S, +, 0)$ *is a commutative monoid and* $(S, \cdot, 1)$ *is a monoid such that* $\cdot$ *distributes over* $+$ *and* 0 *is an annihilator for* $\cdot$*. A semiring is idempotent if* $+$ *is idempotent and commutative if* $\cdot$ *is commutative. In an idempotent semiring the relation,* $a \leq b \iff^{\text{def}} a + b = b$ *is a partial order (i.e., a reflexive, antisymmetric and transitive*

relation) called the natural order on S. It has 0 *as its least element. Moreover,* $+$ *and* $\cdot$ *are isotones with respect to* $\leq$*.* ■

Definition 3.4.4 (Product Family Algebra — e.g., [HKM11a])**.** *A product family algebra is a commutative idempotent semiring* $(S, +, \cdot, 0, 1)$*, where:*

a) S *corresponds to a set of product families;*

b) $+$ *is interpreted as the alternative choice between two product families;*

c) $\cdot$ *is interpreted as a mandatory composition of two product families;*

d) 0 *corresponds to an empty product family;*

e) 1 *corresponds to a product family consisting of only a pseudo-product which has no features.*

■

The term $a+1$ is the product family offering the choice between a and the identity product. Let p_1 and p_2 be the policies enforced at nodes N_1 and N_2, respectively. The policy p_1 consists of the composition of rules r_1, r_2, and r_3 (i.e., $p_1 = r_1 \cdot r_2 \cdot r_3$). While the policy p_2 is the composition of r_1 and r_2 (i.e., $p_2 = r_1 \cdot r_2$). Let N_0 be the immediate parent node of N_1 and N_2 on the graph that represent the network. Assuming N_0 has only N_1 and N_2 as children, then the family of policies enforced at N_0 is $F = p_1 + p_2 = r_1 \cdot r_2 \cdot r_3 + r_1 \cdot r_2 = r_1 \cdot r_2 \cdot (r_3 + 1)$. Hence, p_1 and p_2 share the rules r_1 and r_2. However, p_1 has only one extra rule r_3. Therefore, the term $(r_1 \cdot r_2)$ denotes the commonality of the products in the family F.

Set Model In [HKM11a], concrete models are presented to give a meaning for the semiring structure. Each model consists of sets of products where each product is a set of features.

Let $\mathbb{F}$ be a set of arbitrary features. A product is a set or collection of features which is an element of the power set of F. The power set of $\mathbb{F}$ (i.e., $\mathbb{P} \stackrel{\text{def}}{=} \mathcal{P}(\mathbb{F})$) is the set of all products of $\mathbb{F}$. A product family is a subset of $\mathbb{P}$ that is a collection of products. A special family $1 = \{\emptyset\}$ is the family that has one product with no features. The family 0 is the empty set of products.

A formal definition of the semiring operations is given in [HKM11a]. The operation $\cdot$ is the composition of families on products.

$$\cdot : \mathcal{P}(\mathbb{P}) \times \mathcal{P}(\mathbb{P}) \to \mathcal{P}(\mathbb{P})$$
$$P \cdot Q \stackrel{\text{def}}{=} \{p \cup q \mid p \in P \wedge q \in Q\}.$$

The operation + offers a choice between families.

$$+ : \mathcal{P}(\mathbb{P}) \times \mathcal{P}(\mathbb{P}) \to \mathcal{P}(\mathbb{P})$$
$$P + Q \stackrel{\text{def}}{=} P \cup Q,$$

where $\cup$ denotes the set union.

The structure of the set model is $\mathbb{P}\text{FS} \stackrel{\text{def}}{=} \big(\mathcal{P}(\mathbb{P}), +, \emptyset, \cdot, \{\emptyset\}\big)$. Note, the set model does not allow products to have multiple occurrences of the same feature.

Bag Model Sometimes, it is desired to have multiple occurrences of the same feature in a product. To achieve this, we use the bag model. The definition of 1, 0, and the operation + are the same as in the set model. However, the operation $\cdot$ as is redefined to allow for multiple occurrences of features.

$$\cdot : \mathcal{P}(\mathbb{P}) \times \mathcal{P}(\mathbb{P}) \to \mathcal{P}(\mathbb{P})$$
$$P \cdot Q \stackrel{\text{def}}{=} \{p \uplus q \mid p \in P \wedge q \in Q\},$$

where $\uplus$ denotes the bag sum.

Products, Features, Refinement, and Constraints

Definition 3.4.5 (Product — e.g., [HKM11a])**.** *Let $\mathcal{A} = (A, +, \cdot, 0, 1)$ be a product family algebra and let $a \in A$. We say that a is a product if it satisfies the following:*

$$(\forall\ b \mid b \in A \cdot b \leq a \implies (b = 0 \vee b = a)), \tag{3.1}$$

$$(\forall\ b, c \mid b, c \in A \cdot a \leq b + c \implies (a \leq b \vee a \leq c)). \tag{3.2}$$

The element 0 is a product. A product is said to be proper if $a \neq 0$. ■

Equation 3.1 indicates that a product does not have a subfamily except the empty family and itself. Equation 3.2 states that if a product a is a subfamily of a family formed by c and b, it must be a subfamily of one of them. The policy p_1 defined above is an example of a product according to Definition 3.4.5.

Definition 3.4.6 (Feature — e.g., [HKM11a])**.** *Let $\mathcal{A} = (A, +, \cdot, 0, 1)$ be a product family algebra. We say that $a \in A$ is a feature if it is a proper product different than 1 satisfying:*

$$(\forall\ b \mid b \in A \cdot b \,|\, a \implies b = 1 \vee b = a), \tag{3.3}$$

$$(\forall\ b, c \mid b, c \in A \cdot a \,|\, (b \cdot c) \implies a \mid b \vee a \mid c), \tag{3.4}$$

where the division operator $\mid$ is defined for $x, y \in A$ by $(x \mid y) \overset{\text{def}}{\iff} (\exists\ z \mid z \in A \cdot y = x \cdot z)$. ■

Equation 3.3 states that if we have a product b that divides a, then either b is 1 or $b = a$. Equation 3.4 states that for all product families b and c, if a is mandatory to form $b \cdot c$, then it is mandatory to form b or it is mandatory to form c. Essentially, a feature is a product that is indivisible with regards to $\cdot$ operator.

Definition 3.4.7 (Subfamily — e.g., [HKM11a]). *Let $\mathcal{A} = (A, +, \cdot, 0, 1)$ be a product family algebra. For every $a, b \in A$, we say that a is a subfamily of b denoted by $a \leq b$ iff* $(\forall\, a, b \mid a, b \in A \cdot a + b = b)$. ■

Let a and b be two product families, a is a subfamily of b written as $a \leq b$ iff all the products in family a are at the same time products of the family b. For example, the policy p_1 is a subfamily of F, since $p_1 \leq F = p_1 + F = p_1 + (p_1 + p_2) = p_1 + p_2 = F$.

Definition 3.4.8 (Refinement — e.g., [HKM11a]). *Let $\mathcal{A} = (A, +, \cdot, 0, 1)$ be a product family algebra and $a, b \in A$. We say that a refines b written as $a \sqsubseteq b$ iff* $(\forall\, a, b \mid a, b \in A \cdot (\exists\, c \mid c \in A \cdot a \leq b \cdot c))$. ■

This refinement relation is a preorder (i.e., reflexive and transitive).

For two product families a and b, a is a refinement of b written as $a \sqsubseteq b$ iff every product in a has at least all the features of some products in b. For example, $p_1 \sqsubseteq p_2$ as $p_1 \sqsubseteq p_2 \iff (\exists\, c \mid \cdot \quad p_1 \leq p_2 \cdot c) \iff (\exists\, c \mid \cdot \quad p_1 + p_2 \cdot c = p_2 \cdot c) \iff (\exists\, c \mid \cdot \quad r_1 \cdot r_2 \cdot r_3 + r_1 \cdot r_2 \cdot c = r_1 \cdot r_2 \cdot c)$ which is satisfied for $c = r_3$ as p_1 has all the rules of p_2 and more (i.e., rule r_3). Moreover, we also have $p_1 \sqsubseteq F$ as $p_1 \sqsubseteq F \iff (\exists\, c \mid \cdot \quad p_1 \leq F \cdot c) \iff (\exists\, c \mid \cdot \quad p_1 + F \cdot c = F \cdot c) \iff (\exists\, c \mid \cdot \quad r_1 \cdot r_2 \cdot r_3 + (r_1 \cdot r_2 \cdot (r_3 + 1)) \cdot c = (r_1 \cdot r_2 \cdot (r_3 + 1)) \cdot c)$, which is true for $c = 1$.

Constraints in feature models [KCH+90] are captured in PFA using the requirement relation. The relations of *subfamily* and *refinement* are used to define requirement relation.

Definition 3.4.9 (Requirement — e.g., [HKM11a]). *Let $\mathcal{A} = (A, +, \cdot, 0, 1)$ be a product family algebra. For elements $a, b, c, d \in A$, and product $p \in A$ we define the requirement constraint, in a family-induction style,*

$$a \xrightarrow{p} b \quad \stackrel{\text{def}}{\Leftrightarrow} \quad (p \sqsubseteq a \implies p \sqsubseteq b),$$
$$a \xrightarrow{c+d} b \quad \stackrel{\text{def}}{\Leftrightarrow} \quad a \xrightarrow{c} b \wedge a \xrightarrow{d} b,$$

where $a \xrightarrow{p} b$ *means that if product* p *has feature* a *then it also has feature* b*.* ■

Constraints have two main usages: inclusion or exclusion. Consider the following rules for a policy: r_1 and r_2. An example of an inclusion constraint is: $r_1 \xrightarrow{F} r_2$, which means if we have the rule r_1 in a product of the family F then we must have the rule r_2. On the other hand, an example of an exclusion constraint comes in the form: $r_1 \cdot r_2 \xrightarrow{F} 0$, which states that we cannot have the rules r_1 and r_2 at the same time in any product of F.

In the context of access control policies, the constraints are used to express policies on the specified policies. This is similar to the concept of *metapolicy* in ABAC or conflict classes in the Chinese wall model. For example, constraints can be used to express that if a user a is allowed access to resource x, then it should be denied access to resource y. These constraints ensure the enforcement of global policies that limit the implemented local policies. Using PFA, policies can be defined and constrained using the requirement relation. For example, constraints can be specified by administrators and resource owners are allowed to define their policies. Any rule that violates the constraints is eliminated. We say that a family F satisfies a constraint $(a \xrightarrow{q} b)$, and we write $((a \xrightarrow{q} b) \vdash F)$, iff $(\forall\, p \mid p \leq F \wedge q \sqsubseteq p \cdot a \xrightarrow{p} b\,)$.

We have the divisibility relation among families $(a \mid b) \iff (\exists\, c \mid \cdot \quad b = a.c\,)$[1], which allows us to find divisors of families. And therefore, find the GCD. The GCD is the common divisor that is divided by all other common divisors. Hence, the following property holds: $\texttt{gcd}(a, b) = d$ such that the following condition is satisfied, $(\, d \mid \cdot \quad (d \mid a) \wedge (d \mid b) \wedge ((\forall\, c \mid \cdot \quad (c \mid a) \wedge (c \mid b)\,) \implies (c \mid d))\,)$. Finding the commonalities of two families is formalized by finding the GCD. The annihilating element of `gcd` is 1 such that for any family a the $\texttt{gcd}(1, a) = 1$.

[1]In this thesis, we adopt the notation used by Gries and Schneider in [GS93] for quantifiers. The general form of the notation is $\star(x \mid R : P)$ where $\star$ is the quantifier, x is the dummy or quantified variable, R is predicate representing the range, and P is an expression representing the body of the quantification. An example of the notation is $(+\, i \mid 1 \leq i \leq 3 \cdot i^2\,) = 1^2 + 2^2 + 3^2$.

Definition 3.4.10 (Coprime families — e.g., [KJA17])**.** *Let a and b be two product families, we say they are coprime families if* $\gcd(a, b) = 1$. ■

In the context of this thesis, a policy is a set of rules or a single rule obtained by combining other rules. An atomic rule (discussed in Chapter 4) is an indivisible modelled as a guarded command.

Using product family algebra, it is possible to perform calculations on features, products, and product families. Through calculus, one can verify the equivalence of two specifications, give the number of possible products of a product family, list the common features, or verify whether a family refines another family. In addition, it is possible to build new families, find new products, and exclude unwanted feature combinations. The PFA theory presented in this section is a general representation which can be interpreted in many domains. It has been interpreted in the domain of requirement scenarios [HKM11b]. In the next chapter, we give an interpretation of PFA in the domain of access control policies where a feature is interpreted as a rule modelled in guarded commands, and a product is interpreted as a deterministic policy.

For more details on the product families and the use of this mathematical framework to specify them, we refer the reader to [HKM06, HKM08, HKM11a].

3.5 Summary

In this chapter, we gave the necessary mathematical background for the rest of the thesis. Essentially, policies are modelled as relations using guarded commands. A policy is a set of rules or a single rule obtained by combing rules. A policy specification at an access control point is a family of policies. Tabular expressions are used to represent policies and aid in automating the manipulation and analyses of policies.

Chapter 4

Access Control Policies as Product Families

The first step to analyze and reason on a network access control system consisting of distributed resources is to specify the system formally. This chapter tackles this issue. First, Section 4.1 introduces the mathematical framework to specify access control systems, where a policy at an access control point is modeled as a family of related policies. Section 4.2 presents the concepts of policy and atomic rule. It also gives interpretations of the concepts of subfamily, refinement, constraint, GCD, and coprime families in terms of policies and rules. Section 4.3 presents further usages of the framework. Finally, Section 4.4 summarizes the chapter.

4.1 Policy Family Model

Based on guarded commands and PFA presented in Chapter 3, a model for access control policies can be defined. Let G be a set of rules (i.e., commands). Let $\mathbb{P} \stackrel{\text{def}}{=} \mathcal{P}(G)$, a set of policies. An element of $\mathcal{P}(\mathbb{P})$ is a set of policies.

Definition 4.1.1 (Policy Family Model — e.g., [KJA17]). *A policy family model* $\mathcal{F} = (\mathcal{P}(\mathbb{P}), \oplus, \odot, 0_{\mathcal{F}}, 1_{\mathcal{F}})$ *is a product family algebra, where for* $A, B \in \mathcal{P}(\mathbb{P})$

1. $A \oplus B \stackrel{\text{def}}{=} A \cup B$
2. $A \odot B \stackrel{\text{def}}{=} \{a \sqcap_{\mathbb{P}} b \mid a \in A \wedge b \in B\}$
3. $0_{\mathcal{F}} \stackrel{\text{def}}{=} \emptyset$
4. $1_{\mathcal{F}} \stackrel{\text{def}}{=} \{ \{\mathsf{abort}\} \}$

■

The operation $\oplus$ is the union of policy families representing the choice of families, and the operation $\odot$ is the extended notion of demonic meet presented in Chapter 3 to policies representing the composition of families. The constant $0_{\mathcal{F}}$ is a policy family that is not executable, and the constant $1_{\mathcal{F}}$ is the policy family that has a policy with a single rule abort. It enforces nothing and therefore allows all traffic to pass. Hence, $0_{\mathcal{F}}$ is the annihilator element for $\odot$ and $1_{\mathcal{F}}$ is the neutral element for $\odot$. $A \odot 0_{\mathcal{F}} = \{a \sqcap_{\mathbb{P}} b \mid a \in A \wedge b \in 0_{\mathcal{F}}\} = \{a \sqcap_{\mathbb{P}} b \mid a \in A \wedge b \in \emptyset\} = \{a \sqcap_{\mathbb{P}} b \mid a \in A \wedge \mathsf{false}\} = \{a \sqcap_{\mathbb{P}} b \mid \mathsf{false}\} = \emptyset = 0_{\mathcal{F}}$. Also, we have $A \odot 1_{\mathcal{F}} = \{a \sqcap_{\mathbb{P}} b \mid a \in A \wedge b \in 1_{\mathcal{F}}\} = \{a \sqcap_{\mathbb{P}} b \mid a \in A \wedge b \in \{ \{\mathsf{abort}\} \} \} = \{a \sqcap_{\mathbb{P}} \{\mathsf{abort}\} \mid a \in A\} = \{a \mid a \in A\} = A$. Therefore, we can define the natural order that comes with the semiring structure (i.e., the subfamily relation) for $\mathcal{F}$ as $a \preceq_{\mathcal{F}} b \stackrel{\text{def}}{\Leftrightarrow} a \oplus b = b$. Let N_1, N_2 be two nodes represent one engineering workstation and one finance workstation, respectively, and let N_0 be an access control point governing access to these two resources only. Let p_1 be the policy of the engineering workstation consisting of rules $r_{1.1}, r_{1.2}$, and $r_{1.3}$ presenting the Lines 3-5 of the policy shown in Figures 3.3 and p_2 be the policy of the finance workstation consisting of the Lines 3-5 of the policy shown in Figures 3.4.

$$r_{1.1} = \{\big((s, p, st, dport, a), (s', p', st', dport', a')\big) \mid (s = 192.168.1.0/24 \wedge a' = \texttt{REJECT})\}$$

$$r_{1.2} = \{\big((s, p, st, dport, a), (s', p', st', dport', a')\big) \mid (s = 192.168.2.0/24 \wedge a' = \texttt{ACCEPT})\}$$

$$r_{1.3} = \{\big((s, p, st, dport, a), (s', p', st', dport', a')\big) \mid (s = 192.168.3.0/24 \wedge a' = \texttt{REJECT})\}$$

Recall that a policy is a set of rules or a single combined rule. Therefore, the policy p_1 can be represented as set of rules $r_{1.1}$, $r_{1.2}$ and $r_{1.3}$ or a combined rule r_1.

$$r_1 = \{((s, p, st, dport, a), (s', p', st', dport', a')) \mid (s = 192.168.2.0/24 \wedge a' = \texttt{ACCEPT})$$
$$\vee\ (s \in \{192.168.1.0/24, 192.168.3.0/24\} \wedge a' = \texttt{REJECT})\}$$

In the policy family model, p_1 is the family of policies that has one policy $p_1 = r_{1.1} \odot r_{1.2} \odot r_{1.3} = r_1$ or in a set representation as a set of one element $p_1 = \{\{r_{1.1}, r_{1.2}, r_{1.3}\}\} = \{\{r_1\}\}$. Similar treatment for p_2, the policy of the finance workstation. Let F_0 be the policy family at N_0, and let it equal to $F_0 = p_1 \oplus p_2$ or a set of sets $F_0 = \{p_1, p_2\}$. Because firewall rules are executed sequentially, they need to be transformed into disjoint rules before being represented in the policy family model. The reason for this transformation is that rules in our model are executed without necessary following and order.

4.2 Notions and Properties

In this section, we define the notions of policy and atomic rule and give an interpretation for the concepts of subfamily, refinement, and requirement in the policy family model $\mathcal{F}$. A policy in the model $\mathcal{F}$ is the family that cannot be decomposed using the $\oplus$ operator and defined formally below.

Definition 4.2.1 (Policy)**.** *An element p is called a policy if it satisfies:*

$$(\forall\, q \mid q \cdot q \preceq_{\mathcal{F}} p \implies (q = 0_{\mathcal{F}} \vee q = p)\,), \tag{4.1}$$

$$(\forall\, q, t \mid q, t \cdot p \preceq_{\mathcal{F}} q \oplus t \implies (q \preceq_{\mathcal{F}} p \vee t \preceq_{\mathcal{F}} p)\,). \tag{4.2}$$

The element $0_{\mathcal{F}}$ is a policy. A policy is said to be proper if $p \neq 0_{\mathcal{F}}$. ■

Equation 4.1 states that there is no subfamily for a policy except $0_{\mathcal{F}}$ and itself. Equation 4.2 indicates that if a policy is a subfamily of the policy family $q \oplus t$ then it has to be a subfamily of q or t. For example, the policy p_1 defined above is an example of a policy according to Definition 4.2.1.

Recall that demonic meet to be defined on guarded commands, they need to be integrable. Therefore, since a policy is formed using $\odot$, its rules need to be integrable. To solve this issue for firewall rules, they need to be transformed into disjoint ones as discussed above using a decorrelation algorithm that is applied to the rules. Hence, the members of a policy family do not need to be integrable.

A policy is formed of a set of rules or a single rule modeled using guarded commands. Therefore, it is important to differentiate an atomic or a singleton rule from a combined rule.

Definition 4.2.2 (Atomic rule)**.** *We say that a policy r is an atomic rule if it satisfies:*

$$(\forall\, s \;\mid\; s \cdot s \,|_{\mathcal{F}}\, r \implies s = 1_{\mathcal{F}} \vee s = r\,), \tag{4.3}$$

$$(\forall\, s, t \;\mid\; s, t \cdot r \,|_{\mathcal{F}}\, (s \odot t) \implies r \,|_{\mathcal{F}}\, s \vee r \,|_{\mathcal{F}}\, t\,), \tag{4.4}$$

where the division operator for policies $|_{\mathcal{F}}$ is defined by $(x \,|_{\mathcal{F}}\, y) \overset{\text{def}}{\iff} (\exists\; z \;\mid\cdot\; y = x \odot z\,)$. ■

Equation 4.3 states that if we have a policy s that divides r, then either s is $1_{\mathcal{F}}$ or $s = r$. Equation 4.4 states that for all policy families s and t, if r is mandatory part of $s \odot t$, then it is mandatory part of s or it is mandatory part of t. For example, rules $r_{1.1}$, $r_{1.2}$ and, definitely, r_1 are not atomic rules as they can be decomposed further using division. An atomic rule is the one that has a single value for each starting state attribute. Therefore, the rule r_a is an atomic rule.

$$r_a = \{\big((s, p, st, dport, a), (s', p', st', dport', a')\big) \mid (s = 192.168.1.0/32 \wedge p = \texttt{TCP} \wedge st = \texttt{NEW}$$

$\wedge\ dport = 80 \wedge a = \texttt{ACCEPT} \wedge a' = \texttt{REJECT})\}$.

Note that an atomic rule that is mapped to multiple ending states is still an atomic rule because it cannot be decomposed further according to the division operator defined above. therefore, the rule r_{a2} is still an atomic rule.

$$r_{a2} = \{\big((s, p, st, dport, a), (s', p', st', dport', a')\big) \mid (s = 192.168.4.0/32 \wedge p = \texttt{TCP} \wedge st = \texttt{NEW}$$
$$\wedge\ dport = 80 \wedge a = \texttt{ACCEPT} \wedge a' \in \{\texttt{ACCEPT}, \texttt{REJECT}\})\}.$$

In a similar way to the definitions presented in Chapter 3, we have the notion of subfamily, refinement, and requirement constraints. For that we need to substitute $\leq$ with $\preceq_{\mathcal{F}}$. We denote the refinement on families of policies by $\sqsubseteq_{\mathcal{F}}$ and we use a similar notation for the requirement relation.

Definition 4.2.3 (Subfamily)**.** *The subfamily relation for $\mathcal{F}$ ($\preceq_{\mathcal{F}}$) is defined as $p \preceq_{\mathcal{F}} q \overset{\text{def}}{\Leftrightarrow} p \oplus q = q$.* ■

Let p and q be two policy families, p is a subfamily of q written as $p \preceq_{\mathcal{F}} q$ iff all the policies in family p are at the same time policies of the family q. For example, the policy p_1 is a subfamily of F_0, since $p_1 \preceq_{\mathcal{F}} F_0 = p_1 \oplus F_0 = p_1 \oplus (p_1 \oplus p_2) = p_1 \oplus p_2 = F_0$.
Now, we can define the refinement relation for $\mathcal{F}$ as follows:

Definition 4.2.4 (Policy Family Refinement)**.** *Let p and q be two policy families, we say p refines q iff $p \sqsubseteq_{\mathcal{F}} q \overset{\text{def}}{\Leftrightarrow} (\exists\, s \mid \cdot \quad p \preceq_{\mathcal{F}} q \odot s\,)$* ■

For two policy families p and q, the policy family p refines q written as $p \sqsubseteq_{\mathcal{F}} q$ iff every policy in p has at least all the atomic rules of some policies in q. This asserts that p is a refined version (i.e., superset or equal) of some policies in q. For example, $p_1 \sqsubseteq_{\mathcal{F}} F_0$ as $p_1 \sqsubseteq_{\mathcal{F}} F_0 \iff (\exists\, s \mid \cdot \quad p_1 \preceq_{\mathcal{F}} F_0 \odot s\,)$, which is true for $s = 1_{\mathcal{F}}$.
Also, let $p_3 = r_{1.1}$, then $p_1 \sqsubseteq_{\mathcal{F}} p_3$ as $p_1 \sqsubseteq_{\mathcal{F}} p_3 \iff (\exists\, s \mid \cdot \quad p_1 \preceq_{\mathcal{F}} p_3 \odot s\,) \iff (\exists\, s \mid \cdot \quad (r_{1.1} \odot r_{1.2} \odot r_{1.3}) \oplus (r_{1.1} \odot s) = (r_{1.1} \odot s)\,)$ which is satisfied for $s = r_{1.2} \odot r_{1.3}$ as p_1 has

all the rules of p_3 and more (i.e., rule $r_{1.2}$). $A \equiv_{\mathcal{F}} B \overset{\text{def}}{\iff} A \sqsubseteq_{\mathcal{F}} B \wedge B \sqsubseteq_{\mathcal{F}} A$ defines the equivalence relation $\equiv_{\mathcal{F}}$ associated with $\sqsubseteq_{\mathcal{F}}$.
Since $\mathcal{F}$ is a model for PFA, then we have

$$p \preceq_{\mathcal{F}} q \Rightarrow p \sqsubseteq_{\mathcal{F}} q, \tag{4.5}$$

$$p \odot q \sqsubseteq_{\mathcal{F}} q, \tag{4.6}$$

$$p \sqsubseteq_{\mathcal{F}} p \oplus q, \tag{4.7}$$

$$p \sqsubseteq_{\mathcal{F}} q \Rightarrow p \oplus r \sqsubseteq_{\mathcal{F}} q \oplus r, \tag{4.8}$$

$$p \sqsubseteq_{\mathcal{F}} q \Rightarrow p \odot r \sqsubseteq_{\mathcal{F}} q \odot r, \tag{4.9}$$

$$p \sqsubseteq_{\mathcal{F}} 0_{\mathcal{F}} \Leftrightarrow p \preceq_{\mathcal{F}} 0_{\mathcal{F}}, \tag{4.10}$$

$$0_{\mathcal{F}} \sqsubseteq_{\mathcal{F}} p \sqsubseteq_{\mathcal{F}} 1_{\mathcal{F}}, \tag{4.11}$$

$$p \mid_{\mathcal{F}} q \Leftrightarrow q \sqsubseteq_{\mathcal{F}} p, \tag{4.12}$$

$$p \oplus q \sqsubseteq_{\mathcal{F}} r \Leftrightarrow p \sqsubseteq_{\mathcal{F}} r \wedge q \sqsubseteq_{\mathcal{F}} r, \tag{4.13}$$

$$p \sqsubseteq_{\mathcal{F}} q \oplus r \Leftrightarrow p \sqsubseteq_{\mathcal{F}} q \vee p \sqsubseteq_{\mathcal{F}} r \tag{4.14}$$

Now, we can define the requirement relation presented in Section 3.4 for $\mathcal{F}$ as follows.

Definition 4.2.5 (Policy Requirement)**.** *For elements q, r, s, t and a policy p in $\mathcal{F}$, the requirement relation ($\rightarrow$) is defined*[1] *in a family-induction style as:*

$$q \xrightarrow{p} r \quad \overset{\text{def}}{\Leftrightarrow} \quad p \sqsubseteq_{\mathcal{F}} q \implies p \sqsubseteq_{\mathcal{F}} r$$
$$q \xrightarrow{s \oplus t} r \quad \overset{\text{def}}{\Leftrightarrow} \quad q \xrightarrow{s} r \wedge q \xrightarrow{t} r.$$

■

The first argument states that the relation $q \xrightarrow{p} r$ implies that if the policy p satisfies all the policies in the family q, then it must satisfy all the policies in r. This relation is called

[1] *As it is a simple instantiation in a model of PFA of that of Section 3.4, we use the same notation.*

a Policy Requirement Constraint (PRC) and usually used to express constraints on the policies. These constraints can be of the requirement of inclusion, for example, $q \xrightarrow{p} r$ means that if we have q in any member of p then we must have r at the same time. While if we want to express the exclusion such that two elements q and r cannot coexist in any member of p we write $q \cdot r \xrightarrow{p} 0_{\mathcal{F}}$.

For example, if we have the requirement that system administrators' machines from the subnetwork 192.168.5.0/24 are to be allowed to every resource in F_0, then we write the policy family p_4 with the rule r_4

$$r_4 = \{\big((s, p, st, dport, a), (s', p', st', dport', a')\big) \mid (s = 192.168.5.0/24 \wedge a' = \texttt{ACCEPT})\},$$

and we define the constraint $1_{\mathcal{F}} \xrightarrow{F_0} p_4$. This constraint states that every policy of the family F_0 should have the policy p_4 as $1_{\mathcal{F}}$ is refined by all policies. Moreover, if we have the requirement that every resource an engineer has access to, then his manager from machine IP 192.168.6.0/24 should have access to at the same time, then $p_{1.1}$ has the rule $r_{1.1}$ and we write the policy p_5 with one rule r_5

$$r_5 = \{\big((s, p, st, dport, a), (s', p', st', dport', a')\big) \mid (s = 192.168.6.0/24 \wedge a' = \texttt{ACCEPT})\},$$

and we define the constraint $p_{1.1} \xrightarrow{F_0} p_5$. Finally, if we have the requirement that an engineer and a financial analyst cannot have access to the same resource, then we write the constraint $(p_{1.1} \odot p_{2.1}) \xrightarrow{F_0} 0_{\mathcal{F}}$.

We now define the GCD operation mentioned in Section 3.4 for $\mathcal{F}$ as it is a model of PFA.

Definition 4.2.6 (Greatest Common Divisor for Policies)**.** *The GCD of families of policies is defined as the demonic join of the policies as follow:*

$(\forall A, B \mid A, B \in \mathcal{P}(\mathbb{IP}) \cdot \texttt{gcd}(A, B) \stackrel{\text{def}}{=} \{a \sqcup_{\mathbb{IP}} b \mid a \in A \wedge b \in B\})$. ■

For example, the GCD of the two policy families p_1 and p_2 is a policy family $p_3 = r_3$

$$r_3 = \{((s, p, st, dport, a), (s', p', st', dport', a')) \mid (s = 192.168.3.0/24 \wedge a' = \texttt{REJECT})$$
$$\vee (s \in \{192.168.1.0/24, 192.168.2.0/24\} \wedge a' \in \{\texttt{ACCEPT}, \texttt{REJECT}\})\}$$

Calculating the GCD is more comprehensive when we use tabular expressions. This will be evident in the automation of Chapter 5.

Since GCD is a divisor of a policy family and from implication 4.12, we can state that $\texttt{gcd}(p, q) \Rightarrow p \sqsubseteq_{\mathcal{F}} \texttt{gcd}(p, q) \wedge q \sqsubseteq_{\mathcal{F}} \texttt{gcd}(p, q)$.

Definition 4.2.7 (Coprime Policies)**.** *Let A and B be two families of policies, we say they are coprime policies iff $\texttt{gcd}(A, B) = 1_{\mathcal{F}}$.* ■

For example, the policies $p_{1.1}$ and $p_{2.1}$ are an example of coprime policies as they do not have any common domain.

4.3 Policy Family Model Usage

In this section, we discuss multiple usages of the policy family model. These usages include network policy abstraction, measuring the security requirement level of a policy family, and policy conflicts.

4.3.1 Policy Abstraction

Using this model, it is possible to abstract a network policy as the family of policies of its resources. This family is implemented at the entry point of the network. For example, resources within a lab can be abstracted as the node implementing the family policy consisting of the resource policies. Using such an approach allows us to reason on network policies at multiple levels.

One way to handle large networks is by abstracting subnetworks into one node that has a family of policies and not one policy as we would do for a single resource. Consider a room including a large number of computers that have very similar access control policies. We can abstract this room into one node in the global network that has a family of policies. Moreover, we can have our global network distributed in many buildings. In designing the subnetwork for a building, we consider the other resources located at other buildings as nodes each with a family of policies. Furthermore, in real-world networks, a security policy governing an access control point consists of rules, possibly coming from different stakeholders (for example management, security officers, or users). Clearly, the policies derived from the perspectives of the stakeholders, share common rules and differ on others. Therefore, we can think of these policies as families of policies. Hence, we need to adopt a family approach to manage similar cases illustrated above. For this purpose, we adopt PFA as the formalism that links the access control policies to the network.

4.3.2 Measuring Security Requirement Level

In this section, we discuss one way of comparing policy families by measuring their security requirement level or strictness. This measurement is based on assigning a weight value to the atomic rules. Such that, the weight of a policy is the sum weight of its atomic rules and the weight of a policy family is the weight of the internal union of the policies.

Recall that an atomic rule is modeled as a guarded command, which is essentially a transition relation from a starting state to a possible ending state(s). The weight of an atomic rule is calculated based on the weights of it is ending state(s). One way of computing the weight of an ending state is done by assigning weights to the different values of a chosen state attribute. The higher the security requirement the higher the weight assigned to the value. Moreover, since a chain is strong as its weakest link, an atomic rule that maps a starting state to multiple ending states has a weight equal to the minimum of the weights of the ending states.

Formally, let $\texttt{SA}_i$, where $1 \leq i \leq m$, be the different state attributes, and let $V_{\texttt{SA}_i} = \{a_{i1}, a_{i2}, \ldots, a_{in}\}$ be the set of all possible values assigned to $\texttt{SA}_i$. Then $w_{V_{\texttt{SA}_i}} : V_{\texttt{SA}_i} \rightarrow \mathbb{Z}$ is the weight function that assigns an integer value to each element in $V_{\texttt{SA}_i}$, such that for any two elements $a_{ik}, a_{il} \in V_{\texttt{SA}_i}$ if the security requirement of a_{ik} is less than that of a_{il}, then $w_{V_{\texttt{SA}_i}}(a_{ik}) < w_{V_{\texttt{SA}_i}}(a_{il})$. Let $\mathbf{R}$ be the set of all atomic rules for all the resources in the organization. Then $w_{\mathbf{R}} : \mathbf{R} \rightarrow \mathbb{N} \cup \{-1\}$, is the weight function which assigns to an atomic rule $r \in \mathbf{R}$ its corresponding weight. The weight of $1_{\mathcal{F}}$ is taken to be -1, as it is a rule that does not bring any security constraints. Furthermore, the weight of a rule (r) with its domain mapped to multiple end states (say p), is the minimum of the weights assigned to its end states. For each end state s_i, where $1 \leq i \leq p$, we take $v_{s_i} = eval(w_{V_{\texttt{SA}_1}}, w_{V_{\texttt{SA}_2}}, \ldots, w_{V_{\texttt{SA}_p}})$, where $w_{V_{\texttt{SA}_i}}$ is the weight of the assigned value to the state attribute $\texttt{SA}_i$. For a state s_i, the *eval* function takes the weight of the attributes of s_i and assigns to it an overall security weight v_{s_i}. Then to compute the weight of such an atomic rule, we take the minimum of all the values v_{s_i}. Therefore, for an atomic rule r that has p end states, we have $w_{\mathbf{R}}(r) = min(v_{s_1}, \cdots, v_{s_p})$.

Recall from the illustrative example presented in Chapter 3. For the firewall policy, we compute the weights of our atomic rules based on the ACTION (`AC`) state attribute, where $\texttt{AC} = \{\texttt{ACCEPT}, \texttt{REJECT}, \texttt{DROP}\}$. The three actions in `AC` contribute differently to the confidentiality of the resources of the network, and so `DROP` has a higher weight than `REJECT`, which in turn has a higher weight than `ACCEPT`. Based on this we assign the following weights to each element in `AC`: $w_{\texttt{AC}}(\texttt{ACCEPT}) = 0$, $w_{\texttt{AC}}(\texttt{REJECT}) = 1$, and $w_{\texttt{AC}}(\texttt{DROP}) = 2$. Then, for example, a rule that maps an initial state to two end states (s_1 and s_2) where the action attribute in s_1 is assigned the value `ACCEPT` and the action attribute in s_2 is assigned the value `REJECT`, then the rule has a weight equal to the minimum weight of the two; that is, 0.

The *weight of a policy* or a combined rule r that is formed by the set A_r of atomic rules is the sum of the weights of the atomic rules in A_r. Let $\mathbf{P}$ be the set of all policies composed

of the atomic rules in $\mathbf{R}$. Then $w_{\mathbf{P}} : \mathbf{P} \to \mathbb{N} \cup \{-1\}$, is the weight function that assigns an integer value to each element in $\mathbf{P}$ based on $w_{\mathbf{R}}$. We assign -1 as the weight of the $1_{\mathcal{F}}$ policy, and $+\infty$ as the weight of the $0_{\mathcal{F}}$ policy.

In a typical firewall policy, rules are executed sequentially. Such rules usually are not atomic. For example, rule 9 in the policy in Figure 3.1 can be divided by all other rules in the policy. Therefore, a rule might be followed by another that includes part or all of its domain. If we apply the weight function to these rules, we can have a double-counting of weights. Therefore, to avoid any problem we transform policies into a set of rules that are (relative) atomic (or, relative prime). A *relative atomic/prime rule* in this context is a rule that is indivisible by any other given rule (i.e., it is only divisible by 0 and itself). The weight function takes such a rule and assigns a weight to it. For example, consider the policy shown in Figure 3.3 which is transformed to a relative atomic policy with 270 rules: two rules with `ACCEPT` actions, three rules with `REJECT` actions, and 265 rules with `DROP` actions. Therefore, the weight assigned to the policy is $2 * 0 + 3 * 1 + 265 * 2 = 533$. Figure F.20 shows a snippet of the relative atomic policy of the engineering workstation presented in Figure 3.3 relative to the other policies in the network.

The weight of a policy family is the weight of the internal union of its policies. Then, $w_{\mathbf{F}} : \mathbf{F} \to \mathbb{Z}$ is the weight function that assigns an integer value to a policy family.

The union of the policy family F is defined as $union(F) = (\cup_{\mathbb{P}}\, p_i \mid p_i \in F \cdot p_i)$, and the weight of F is equal to $w_{\mathbf{F}}(F) = w_{\mathbf{P}}(union(F))$.

As can be seen we measure the level of security requirements of a resource by the weight of the policy governing it, such that its security requirements are directly proportional to the weight of its policies. For example, consider two resources v_1 and v_2, where the weights of their policies are $w_{\mathbf{P}}(p(v_1))$ and $w_{\mathbf{P}}(p(v_2))$, respectively. If $w_{\mathbf{P}}(p(v_2)) < w_{\mathbf{P}}(p(v_1))$, it means that v_1 has higher level of security requirement than v_2.

Let $w_{\mathbf{R}}$ be the partial order on the atomic rules based on their weights and $w_{\mathbf{P}}$ be the total order on the policies based also on their weights. Then there exists an order preserving map

$f : w_{\mathbf{R}} \to w_{\mathbf{P}}$ satisfying the following condition:

$$(\forall\, i \mid 1 \leq i \leq n \wedge x_i, y_i \text{ are atomic rules} \cdot x_i <_{w_{\mathbf{R}}} y_i\,)$$
$$\Leftrightarrow f(P(x_1, x_2, \ldots, x_n)) <_{w_{\mathbf{P}}} f(P(y_1, y_2, \ldots, y_n)),$$

$P(x_1, x_2, \ldots, x_n)$, and $P(y_1, y_2, \ldots, y_n)$ are the policies consisting of atomic rules $x_1, x_2, \ldots, x_n$ and $y_1, y_2, \ldots, y_n$ respectively. Since $w_{\mathbf{R}}$ is a partial order, some values of the atomic rules might not have a weight assigned. In this case, the mapping f assigns zero, the neutral value for addition.

4.3.3 Access Request

Checking the inclusion of a policy family R within a policy family F is defined using refinement as follows:

Definition 4.3.1 (Policies Inclusion)**.** *Let R and F be two policy family, we say R is in F iff* $R \in_{\mathcal{F}} F \stackrel{\text{def}}{=} (\forall\, r \mid r \in R \cdot (\exists\, f \mid f \in F \cdot f \sqsubseteq_{\mathcal{F}} r\,))$. ■

Definition 4.3.1 checks that a policy family R is defined within a policy family F. Let r_6 as follow:

$$r_6 = \{\big((s, p, st, dport, a), (s', p', st', dport', a')\big) \mid (s = 192.168.6.0/24 \wedge a' = \texttt{REJECT})\}.$$

Let p_6 be a policy consisting of a single rule r_6 (i.e., $p_6 = r_6$). We retake rules r_4 and r_5 given on Page 77 to have $p_7 = r_4 \odot r_6$ and $p_8 = r_4 \odot r_5$. Let F_1 be a policy family consisting of the policies p_7 and p_8 (i.e., $F_1 = p_7 \oplus p_8$).

Using this operation $\in_{\mathcal{F}}$ we can check the existence of the policy p_6 within the policy family F_1. Note, as policy families are not necessary integrable, we find in F_1 a policy p_8 that leads to a different ending state than p_6 for the same domain. Then, because F_1 has the policy p_2, which has the rule r_5, we cannot guarantee that the behavior of F_1 will be `REJECT`

for the domain of p_6. However, since a rule with multiple ending states has a weight equal to the minimum of its ending states, a policy family that has conflicting policies behaves according to the one with the least security requirement level. Therefore, a policy family is guaranteed if it satisfies an allow access request.

On the other side, if we want to extract the part of a policy family that deals with a certain domain $D \subseteq \Sigma$, we do this by extending the guard command defined in Chapter 3 to policy families.

Definition 4.3.2 (Policy Family Restriction)**.** *Let $D \subseteq \Sigma$ and F be a policy family, the restriction of F to D is defined as:* ${}_{D}\!\downarrow^{\mathcal{F}} F \stackrel{\text{def}}{=} (\oplus P_i \mid P_i \preceq_{\mathcal{F}} F \cdot (\odot R_j \mid R_j \in P_i \cdot D \longrightarrow R_j))$. ■

Let $D \subseteq \Sigma$ defined as follows:

$$\{(s, p, st, dport, a) \mid s = 192.168.1.0/24\}.$$

Then we can extract the part of F_0 that deals with D using ${}_{D_a}\!\downarrow^{\mathcal{F}} F_0$.

$${}_{D_a}\!\downarrow^{\mathcal{F}} F_0 = \{\big((s, p, st, dport, a), (s', p', st', dport', a')\big) \mid (s = 192.168.1.0/24 \wedge a' = \texttt{REJECT})\}$$
$$\oplus\{\big((s, p, st, dport, a), (s', p', st', dport', a')\big) \mid (s = 192.168.1.0/24 \wedge a' = \texttt{ACCEPT})\}.$$

4.3.4 Conflicts

Conflicts between two rules, policies, or policy families arise when they have different actions to the same request. The single firewall conflicts presented in Chapter 2; that is, shadowing [ASH04], correlation, and generalization are formalized in $\mathcal{F}$ as a single conflict called inconsistency of policies. We say that a policy p is inconsistent iff $(\exists r, q \mid r, q \in p \cdot (dom(r) \cap dom(q) \neq dom(r \cap q)))$

From the above, the integrability (i.e., $dom(a \cap b) = dom(a) \cap dom(b)$) guarantees the absence of inconsistency (i.e., shadowing, generalization and correlation) in a policy.

A policy in $\mathcal{F}$ as defined above is a set of rules that are combined using the operator $\odot$. Recall that demonic meet of two commands to be defined these commands need to be integrable. Therefore, rules in a policy need to be integrable. And this requirement guarantees the absence of inconsistency conflict in a single policy.

For a policy family, it is allowed to have conflicting policies. However, integrability checks can be performed and conflicting rules are highlighted to the system administrator. A resolution strategy can be performed in case of conflict, we adopt to allow the most permissive policy, as a link is secure as its weakest point.

In the case of distributed access control points, it is possible to have conflicting policy families on a path from the root to a resource. The conflicts include shadowing and spuriousness as presented in Chapter 2. The refinement relation guarantees the absence of conflicts on a path from the root to a resource. Therefore, if $B \sqsubseteq_{\mathcal{F}} A$, then there are no conflicts in the path from A to B. This will be evident and explain in detail in the next Chapter. Moreover, in the case of many paths from the root to the resource, the refinement guarantees the absence of cross-path inconsistency. The policy faced by a request from the root to a resource is the $\odot$ of these policy families.

4.3.5 Completeness

A policy is complete is if its domain equal to Σ and a policy family is complete if the union all of its policies is complete. The *gcd* of two complete policies is complete.

A policy family is evaluated in response to an access request as follows. If one rule is applicable, then that rule action is executed. If two or more rules are applicable and they have the same action, then that action is executed. If they have different actions, then the most permissive one is executed.

4.4 Summary

In this chapter, we have presented the model for policy family which models policies as related families. We have also defined different operations such as policy family union, policy family composition, subfamily, refinement, requirement, and the concepts of policy and atomic rule based on that model. We have also shown how to measure the security requirement level for a policy family, how to evaluate access requests in a policy family, how to restrict a policy family to a certain domain, and discussed conflicts within this model.

Chapter 5

Defense in Depth

Layered protection or DD is one of the strategies used in securing access to network resources. This chapter presents a formal understanding of this strategy based on the material presented so far. Section 5.1 presents the concept of DD based on the mathematical framework defined in Section 4.1. Moreover, it presents formal approaches to generate access control policies for networks such that DD is satisfied. Section 5.2 present the stricter form of DD. Finally, Section 5.3 summarizes the main results of the chapter.

5.1 Defense in Depth Strategy and its Usage

Defense in depth offers layers of defenses such that traffic traveling the network from the root to a resource is faced with multiple defenses each is equal or stronger than the one preceding it. From the perspective of resources, threats and known untrusted sources are blocked by multiple layers and kept as close to the outer edge as possible.

In this section, we give a formal definition for DD. Moreover, we use the definition to assess whether a given network satisfies the DD strategy. We also present different approaches to assign access control policies to internal access control points such that the DD strategy is satisfactorily implemented.

A network with an outer entry point r can be represented as a connected directed acyclic graph rooted at r. The network resources are represented by the graph leaves while the access control points are represented by the internal vertices. An edge represents a link between nodes. All vertices of the graph represent nodes that execute policies. For example, Figure 5.1 shows a graph representing a network that is rooted at r. The leaves are the vertices v_6 to v_{10}, and the internal vertices are v_1 to v_5. The edge (v_1, v_6) represent the traffic link between the nodes v_1 and v_6.

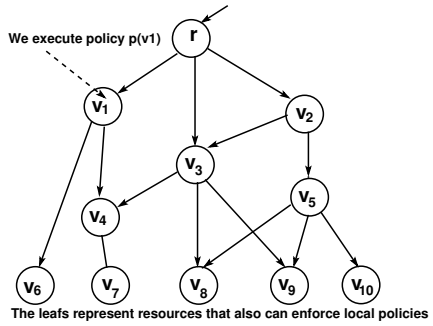


Figure 5.1: A network as a rooted connected directed acyclic graph (figure borrowed from [KJA17])

Let $G \stackrel{\text{def}}{=} (V, E, r)$ be a rooted connected directed acyclic graph representing a network, where:

- V is a set of vertices that represents the set of access control points and resources that enforce access control policies;
- E is a set of edges (i.e., ordered pairs of vertices) that represent links between the network access control points and resources;
- r is the root of the graph and it represents the outer entry point between the network and the external world.

To guarantee the absence of conflicts in the policies of a network G, any edge between two vertices access control points should respect the refinement property discussed in Chapter 4.

For example, if the edge $(v_5, v_{10}) \in E$ then the family of policies implemented at v_5 is refined by the policy implemented at v_{10}. Therefore, the family of policies at a node (e.g., v_{10}) is a refinement of its ancestors v_5, v_2, and r.

Definition 5.1.1 (Defense in Depth Law (DDL) — e.g., [KJA17]). *Let* $G \stackrel{\text{def}}{=} (V, E, r)$ *be a network. Where* $p(v)$ *is family of policies enforced by vertex* v *in* G*. The network* G *employs a DD strategy if* $p(r) \neq 0_{\mathcal{F}} \ \wedge \ (\forall\, a, b \mid (a, b) \in E \cdot p(b) \sqsubseteq_{\mathcal{F}} p(a))$ ■

The $0_{\mathcal{F}}$ family is not allowed as it is a non-executable policy family. The second condition requires that every policy family enforced at a node is more deterministic or at least as deterministic as the family of policies at the parent node in the graph. This is articulated in Proposition 5.1.1(a). Note, if the desired requirement is that $p(b)$ to be more restrictive than $p(a)$, then the condition for $w_{\mathbf{F}}(p(b)) > w_{\mathbf{F}}(p(a))$ is added. Also note that by the refinement condition $p(b) \sqsubseteq_{\mathcal{F}} p(a)$, $p(b)$ is at least as restrictive as $p(a)$ if not more restrictive but never less restrictive. Moreover, the refinement means that every policy in b refines at least a policy in a, the weight of the policy family at b is equal or more to the policy family at a. Moreover, as the weight of a family is equal to the weight of the internal union of its policies, then the weight of the family b is equal or more than the weight of a. Therefore, by the refinement condition, it is ensured that it is as deterministic and as restrictive or more.

Definition 5.1.1 does not prevent the instance where all or some nodes in a path enforce the same policy. For example, in the network given in Figure 5.1, if all nodes in the path $\langle r, v_3, v_8 \rangle$ enforce the same policy, the network would still implement the DD strategy given in the definition. Enforcing the same policy is a waste of network resources as it adds cost without adding value. Moreover, the enforced policy could be the policy family $1_{\mathcal{F}}$ that enforces no control over the traffic. For example, the nodes from the root r to the resource v_8 could enforce the policy family $1_{\mathcal{F}}$. This case hinders the overall security of the network as discussed later in the chapter.

Proposition 5.1.1 (e.g., [KJA17]). *Let G be a network that employ a DD strategy. Let $P = \langle v_1, v_2, ..., v_m \rangle$ be a path of P. We have*

a) $(\forall\, i \mid 1 \leq i \leq m \cdot p(v_m) \sqsubseteq_{\mathcal{F}} p(v_i))$

b) $(\forall\, v \mid v \in E \cdot p(v) \sqsubseteq_{\mathcal{F}} p(r))$

Proof. The proof for item (a) uses the reflexivity and transitivity of $\sqsubseteq_{\mathcal{F}}$ and some basic quantifier rewriting rules. While the proof for item (b) is done by induction on $Q(m) \stackrel{\text{def}}{\Leftrightarrow} (\forall\, i \mid 1 \leq i \leq m \cdot p(v_i) \sqsubseteq_{\mathcal{F}} p(v_1))$. The detailed proof is given in the Appendix A.1. □

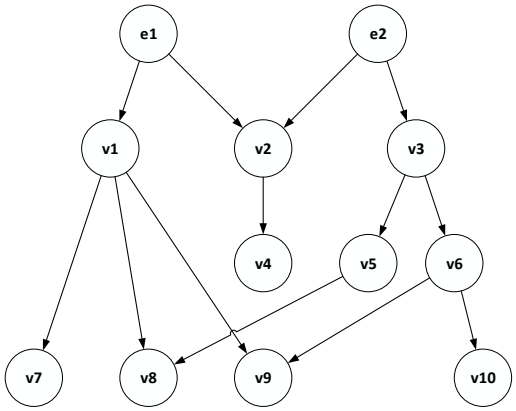
The first result 5.1.1(a), states that if a network implements a DD strategy, then the deeper we go in any path from the root, we are faced by policy families that are as deterministic if not more deterministic than the one proceeds it. For example, if the network shown in Figure 5.1 implements the DD strategy. Then, it is guaranteed that if we take the path $\langle r, v_1, v_4, v_7 \rangle$ starting from r to v_7, then we are faced with policies that are as deterministic or more deterministic as the one before it. The second result 5.1.1(b) states that all the policy families of all the nodes in the network are as deterministic or more deterministic as the policy family enforced at the root. Therefore, if the network of Figure 5.1 implements DD, then the policies of all the nodes $v_1, \ldots v_{10}$ are as deterministic or more deterministic that the policy at the root r.

5.1.1 Defense in Depth in Networks with Multiple Entry Points

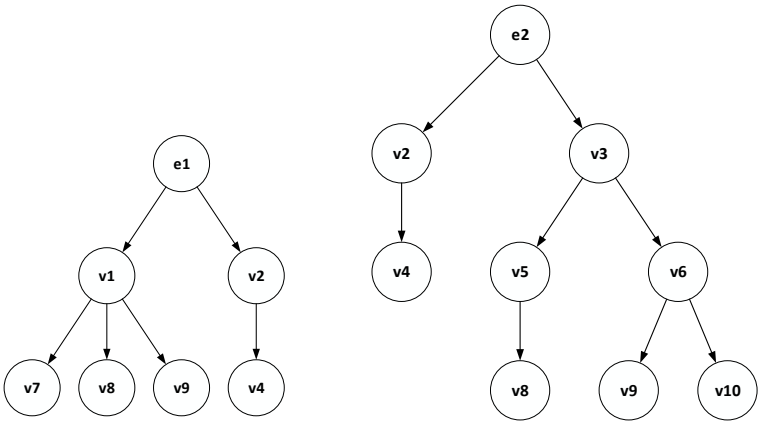
A network with several entry points can be represented as $G \stackrel{\text{def}}{=} (V, E, I)$, where I is a set of roots/entry points, V is the set of vertices, and E is the set of edges. For every $r \in I$, we can derive a network graph $G_r \stackrel{\text{def}}{=} (V_r, E_r, r)$ from the G as follows:

- $V_r = \{v \mid (r, v) \in E^*\} = \{v \mid v \text{ is reachable from } r\}$, where E^* is the reflexive transitive closure of E.
- $E_r = (V_r \times V_r) \cap E$ is the set of edges in G belonging to paths starting at r.

Hence, we notice that a network with several entry points can be decomposed into several networks each with a single entry point. For example, the network shown in Figure 5.2a can be decomposed to the networks in Figures 5.2b and 5.2c. The algorithms presented later in Chapter 7, use this fact to reason on a network with several entry points by reducing it into several networks with only one entry point each. Therefore, we restrict our attention to theoretical background on networks with a single entry point.



(a) Graph G



(b) Graph G_{e_1} (c) Graph G_{e_2}

Figure 5.2: Decomposition of a graph G with two entry points e_1 and e_2 to two graphs G_{e_1} and G_{e_2} with single entry point each

5.1.2 Generating Lower-Level Policies from Higher-Level Ones

Let $G \stackrel{\text{def}}{=} (V, E, r)$ be a graph representing a network. The root r is assigned a level 0, the lowest level. Let v_i a node at level n, then a node v_j such that $(v_i, v_j) \in E$ has a level $n+1$. A node can belong to more than one level as it could be reached from multiple paths of different lengths. The node will belong to a unique level when the graph forms a tree.

Proposition 5.1.2 (e.g., [KJA17])**.** *Let $G \stackrel{\text{def}}{=} (V, E, r)$ be a network. Let T be a directed spanning tree of G rooted at r and having a set L of leaves. For every $l \in L$, we are given $p(l)$. If we have $p(v) \stackrel{\text{def}}{=} (\oplus v_i \mid (v, v_i) \in E \wedge p(v_i) \neq 1_{\mathcal{F}} \cdot p(v_i))$ for every $v \in V$ that is an ancestor of an $l \in L$, then G employs a DD strategy.*

Proof. The condition of the Definition 5.1.1 is satisfied because for each $v \in V$ that is not a leaf, enforces a family of policies generated using the operator $\oplus$ from the families of the nodes that are at a higher level and attached to it. Then, the families of policies at these nodes refine the family of policies at v. □

The scheme presented by Proposition 5.1.2 allows for the generation of the families of policies for the nodes starting from the resources. The process starts by assigning policies for the resources by system administrators. Using these families of policies and this approach, the families of policies for the rest of the network are generated level by level till the root is reached.

Recall the illustrative example presented in Section 3.1. Suppose those resources are placed in the topology architecture shown in Figure 5.3, and we wish to generate the policies for $r, F_1, F_2,$ and F_3 such that the policies enforced implement the DD strategy.

We generate the policies for the internal access control points using the scheme presented in Proposition 5.1.2. Using $\oplus$ is straight forward. The policy family at F_2 will be $p(F_2) = p(Eng_1) \oplus p(Eng_2)$ and F_3 will have the policy family $p(F_3) = p(Fin_1) \oplus p(Fin_2) \oplus p(FinDB)$. The node F_1 will have a policy family $p(F_1) = p(File) \oplus p(F_3) \oplus p(F_2) =$

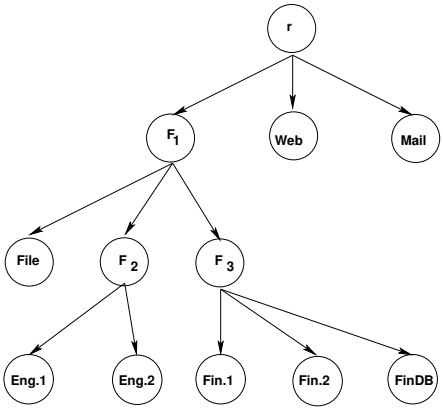


Figure 5.3: Illustrative example topology

$p(File) \oplus p(Eng_1) \oplus p(Eng_2) \oplus p(Fin_1) \oplus p(Fin_2) \oplus p(FinDB)$. Finally, the root r will have the policy family $p(r) = p(F_1) \oplus p(Web) \oplus p(Mail) = p(File) \oplus p(Eng_1) \oplus p(Eng_2) \oplus p(Fin_1) \oplus p(Fin_2) \oplus p(FinDB) \oplus p(Web) \oplus p(Mail)$. We can see that Definition 5.1.1 is satisfied for every edge in the topology graph. For example, $p(Eng_1) \sqsubseteq_{\mathcal{F}} p(F_2) \Leftrightarrow p(Eng_1) \sqsubseteq_{\mathcal{F}} p(Eng_1) \oplus p(Eng_2) \Leftrightarrow p(Eng_1) \sqsubseteq_{\mathcal{F}} p(Eng_1) \vee p(Eng_1) \sqsubseteq_{\mathcal{F}} p(Eng_2)$, which is true. We also see that every policy family refines the policy family at the root. Moreover, on every path from the root to a resource, policy families at the higher level refines the ones at lower level. Note, we can notice that policy families at internal nodes are not integrable.
In the following, a different scheme is presented for the generation of policies.

Proposition 5.1.3 (e.g., [KJA17])**.** *Let $G \stackrel{\text{def}}{=} (V, E, r)$ be a network. Let T be a directed spanning tree of G rooted at r and having a set L of leaves. For every $l \in L$, we are given $p(l)$. If we have $p(v) = (\texttt{gcd} v_i \mid (v, v_i) \in E \wedge p(v_i) \neq 1_{\mathcal{F}} \cdot p(v_i))$ for every $v \in V$ that is an ancestor of an $l \in L$, then G employs a DD strategy.*

Proof. The proof is based on the fact that in PFA, $a \cdot c \sqsubseteq a$. This is true for the current model as it is a model for PFA as discussed in Chapter 4. For a vertex v enforcing a policy that is the `gcd` for the vertices $v_i, \ldots, v_j$, then each of the vertices has a policy equal to $p(v) \odot c$ for some c and therefore refines $p(v)$ and satisfy the condition for DD. □

If two vertices $v_i, v_j \in G$ have a common parent v and are enforcing coprime families of policies (i.e., $\texttt{gcd}(v_i, v_j) = 1_{\mathcal{F}}$) then the families of policies enforced at v equals to $1_{\mathcal{F}}$. It indicates that two coprime families have no common domain. If the families have two rules with the same domain but different action, then their `gcd` is different than $1_{\mathcal{F}}$.

For the resources of the illustrative example presented in Section 3.1 placed in the topology shown in Figure 5.3. We generate the policies for the internal access control points r, F_1, F_2, and F_3 such that the policies enforced implements the DD strategy using the scheme of Proposition 5.1.3. The scheme gives the following policy families. For the node F_2, its policy family is equal to the policy family of the engineering workstations as they have the same policy and the same applies for F_3 for the finance workstations.

Similar policy will be applied at F_3 which changing attributes to the ones of the financial department. The node F_1 will have a policy family with a single policy having the following rule:

$$
\begin{aligned}
&R_{F_1} = \{((s, p, st, dport, a), (s', p', st', dport', a')) \mid \\
&(st \in \{\texttt{RELATED}, \texttt{ESTABLISHED}\} \wedge a' = \texttt{ACCEPT}) \\
&\vee (st = \texttt{INVALID} \wedge a' = \texttt{DROP}) \\
&\vee (s \in \{192.168.1.0/24, 192.168.2.0/24, 192.168.3.0/24\} \wedge a' \in \{\texttt{ACCEPT}, \texttt{REJECT}\}) \\
&\vee (s = 192.168.4.0/24 \wedge a' = \texttt{REJECT}) \\
&\vee (s \notin \{192.168.1.0/24, 192.168.2.0/24, 192.168.3.0/24, 192.168.4.0/24\} \wedge a' = \texttt{DROP})\}
\end{aligned}
$$

The policy of the root r is generated using `gcd` as well. We can see that Definition 5.1.1 is satisfied for every edge in the topology graph. We also see that every policy family refines the policy family at the root. Moreover, on every path from the root to a resource, policy families at the higher level refine the ones at the lower level.

Obviously, a network graph G can have multiple spanning trees $T_1 \cdots T_n$, where $n \leq$

$|V|^{(|V|-2)}$ as stated by Cayley's formula that for m vertices the number of spanning trees is m^{m-2}. For each tree T_i, $1 \leq i \leq n$, the family of policies at an internal node is computed using the `gcd` or $\oplus$ operator schemes. Also note that, an internal node could be a node in multiple spanning trees, say $T_i \cdots T_k$. In this case, the family of policies implemented at such a node v is $p(v) = (\texttt{gcd}\ i \mid 1 \leq i \leq k \cdot p_i(v))$ or $p(v) = (\oplus\ i \mid 1 \leq i \leq k \cdot p_i(v))$, where $p_i(v)$ is the family of policies for the node v in the spanning tree T_i.
The proposition below is about preserving the DD strategy when applying PRCs.

Proposition 5.1.4 (e.g., [KJA17])**.** *Given a network G that employs a DD strategy, where each node v has a family of policies $p(v)$ assigned to it. Let C be a given set of PRCs. The following scheme gives a network that employs a DD strategy.*
For every $v \in V$ that is an ancestor of an $l \in L$, we assign a family of policies $p'(v)$ such that

1. *$p'(v) \leq p(v)$, and*
2. *$(\forall\, c, v, w \mid c \in C \wedge (v, w) \in E \cdot (p(w) \leq p(v)) \wedge (c \vdash p'(v)) \wedge (c \vdash p'(w)))$.*

Proof. Because $p'(v) \leq p(v)$ and the refinement relation between a the family of policies at vertices w and the family of policies at v (i.e., $p(w) \sqsubseteq_{\mathcal{F}} p(v)$) is reduced to the subfamily relation (i.e., $p(w) \leq p(v)$), applying the constraint preserve the refinement relation and therefore the DD strategy. The refinement relation is not guaranteed to be preserved without the condition $(p(w) \leq p(v))$. □

In the network given in Figure 5.3, if the network implements a DD strategy and we have a set of constraints C. The network is guaranteed to preserve the DD only when the refinement relation is reduced to the subfamily relation. If we have constraints and we wish to generate the internal nodes' policies, then the the DD preserved only using the scheme of Proposition 5.1.2.

5.1.3 Calculating GCD using a Prototype Tool

In Appendix B, a prototype tool is presented to show the use of the above theories. The prototype tool calculates the policies of the internal nodes of a network.

5.2 Strict Defense in Depth

This section defines the Strict Defense in Depth strategy SDD which is a stricter form of DD than that of Definition 5.1.1.

Definition 5.2.1 (Strict Defense in Depth Law (SDDL))**.** *Let* $G \stackrel{\text{def}}{=} (V, E, r)$ *be a network. The network* G *employs a SDD strategy if* $p(r) \neq 0_{\mathcal{F}} \ \wedge \ (\forall\, a, b \mid (a, b) \in E \cdot p(b) \sqsubset_{\mathcal{F}} p(a))$, *where* $p(b) \sqsubset_{\mathcal{F}} p(a) \iff (p(b) \sqsubseteq_{\mathcal{F}} p(a) \ \wedge \ p(a) \neq p(b))$. ■

The SDD is a stronger form of DD obtained by replacing the refinement relation $\sqsubseteq_{\mathcal{F}}$ by its stricter form $\sqsubset_{\mathcal{F}}$. Using SDD excludes the instance discussed above of equal policy families on successive nodes in a path. The SDD strategy is used later in Chapter 6 to place resources in the network.

Let $G \stackrel{\text{def}}{=} (V, E, r)$ be a network and $S \stackrel{\text{def}}{=} \{v \mid v \in V \wedge (r, v) \in E\}$ be the set of vertices connected to the root r. Let T be a spanning tree of G rooted at r. For every leaf node $l \in L$ assigned a family of policies $p(l)$. If we want to generate the families of policies of the internal vertices such that G implements SDD, the following lemmas present cases when SDD is not achievable.

Lemma 5.2.1. *Let* T *be a spanning tree of* G *having a height greater than* 2*. If* T *has a leaf* $l \notin S$ *, such that* $p(l) = 1_{\mathcal{F}}$*, then it is impossible to have SDD implementation using*

`gcd` *scheme. Formally,*

$$(\exists\, l \mid l \in L \cdot p(l) = 1_{\mathcal{F}})$$
$$\Longrightarrow$$
$$\neg(p(r) \neq 0_{\mathcal{F}} \wedge (\forall\, a, b \mid (a, b) \in E \cdot p(b) \sqsubset_{\mathcal{F}} p(a)))$$

Proof. The proof is based on the fact that the policies of l ancestors will be equal to $1_{\mathcal{F}}$ using the `gcd` scheme. This violates the requirements of SDD. The detailed proof is in Appendix A.2 □

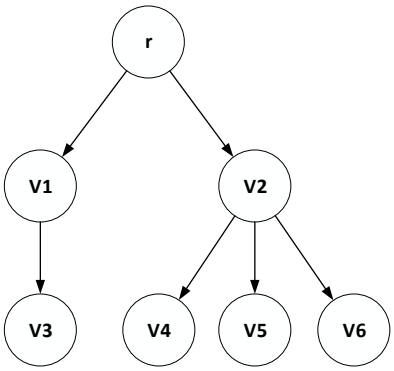


Figure 5.4: A network graph

Lemma 5.2.1 states that if a leaf node, that is not attached to the root, has a family of policies equals to $1_{\mathcal{F}}$ and the implementation scheme is `gcd`, then the family of policies executed at its parent and ancestors will be equal to $1_{\mathcal{F}}$. For example, in Figure 5.4 if $p(v_6) = 1_{\mathcal{F}}$ then calculating $p(v_2)$ and $p(r)$ using the `gcd` scheme will result in their families of policies equal to $1_{\mathcal{F}}$. This case does not meet the requirements of the SDD strategy which states that each child node should strictly refine its parent.

Having a leaf node with a family of policies equal to $1_{\mathcal{F}}$ deep in the network will hinder the overall security of the network. It will result in no security control in the chain of firewalls

from the root to that node. Therefore, allowing unauthorized users to access deep into the network without any line of defense. To prevent this case, the solution is to isolate those nodes from the network or move them closer to the edge. Since those nodes are not meant to be protected, we isolate them from the nodes that need protection. Therefore, by segregating and moving the nodes with $1_{\mathcal{F}}$ policy close to the root we can efficiently protect the resources with stricter policies.

Lemma 5.2.2. *Let T be a spanning tree of G having a height greater than 2. If T has two leaf nodes l_1, l_2 belonging to the same subtree having coprime policy families, then it is impossible to have a SDD implementation using the* `gcd` *scheme. Formally,*

$$
\begin{aligned}
&(\exists\, l_1, l_2, s, n, m \;\mid\; s \in S \wedge n \geq 1 \wedge m \geq 1 \wedge (s, l_1) \in E^n \\
&\wedge\, (s, l_2) \in E^m \;\cdot\; \texttt{gcd}(l_1, l_2) = 1_{\mathcal{F}}\,) \\
&\implies \\
&\neg(p(r) \neq 0_{\mathcal{F}} \;\wedge\; (\forall\, a, b \;\mid\; (a, b) \in E \;\cdot\; p(b) \sqsubset_{\mathcal{F}} p(a)\,)),
\end{aligned}
$$

where $(a, b) \in E^x$ is a path from node a to node b with x edges.

Proof. The proof is based on the fact that the policies of the common ancestors of l_1 and l_2 will be equal to $1_{\mathcal{F}}$ using the `gcd` scheme. This violates the requirements of SDD. The detailed proof is in Appendix A.3 □

Lemma 5.2.2 states that if a subtree T of G has two nodes such that their policy families have no common domain. Then calculating the families of policies of their common ancestors including the root of T and the root of G using the `gcd` scheme will result in the family of policies equal to $1_{\mathcal{F}}$. In this case, the SDD strategy is not satisfied. For example, in Figure 5.4 if $p(v_4)$ and $p(v_6)$ are coprime policy families such that $\texttt{gcd}(p(v_4), p(v_6)) = 1_{\mathcal{F}}$ then calculating $p(v_2)$ and $p(r)$ using the `gcd` scheme will result in $p(v_2) = 1_{\mathcal{F}}$ and $p(r) = 1_{\mathcal{F}}$. Having nodes with coprime policy families at one subtree will result in no security control

in the chain of firewalls from their common ancestor to the root. This affects the overall security of the network. Therefore, a well-designed network groups nodes that share common policies (e.g., belong to the same department) in one segment, and places resources having nothing in common into different segments/subnetworks. This allows for maximum security and protection than grouping nodes randomly. The solution to avoid this case is to place nodes that have different domains into different subnetworks.

Lemma 5.2.3. *If we have a family of policies at a node that is equal to the GCD of the policies of its siblings, then it is impossible to have a SDD implementation using* `gcd` *scheme. Formally,*

$$(\exists\, v \mid (u,v) \in E \cdot p(v) = (\texttt{gcd}\, v_i \mid (u,v_i) \in E \cdot p(v_i)))$$

$$\implies$$

$$\neg(p(r) \neq 0_{\mathcal{F}} \ \wedge \ (\forall\, a,b \ \mid \ (a,b) \in E \ \cdot \ p(b) \sqsubset_{\mathcal{F}} p(a)\,))$$

Proof. If the policy of a node v is equal to the `gcd` of all the members of the segment, then the policy of the root of the segment using the `gcd` scheme is equal to the policy of the node v. This violates the requirements of SDD. The detailed proof is in Appendix A.4. □

Lemma 5.2.3 states that if we have a node such that its policy family is equal to the commonality of its siblings. Then calculating the family policies of its parent using the `gcd` scheme will equal to the policy of this node. Therefore, not satisfying the SDD strategy. For example, in Figure 5.4 if $p(v_4) = \texttt{gcd}(p(v_5), p(v_6))$ then calculating $p(v_2)$ using the `gcd` scheme will result in its policy $p(v_2) = p(v_4)$.

Lemma 5.2.4. *It is impossible to have a SDD implementation using $\oplus$ scheme. Formally,*

$$(\exists\, v \mid (u,v) \in E \cdot p(u) = (\oplus\, v_i \mid (u,v_i) \in E \cdot p(v_i)))$$
$$\implies$$
$$\neg(p(r) \neq 0_{\mathcal{F}} \ \wedge\ (\forall\, a,b \ \mid\ (a,b) \in E \ \cdot\ p(b) \sqsubset_{\mathcal{F}} p(a)\,))$$

Proof. The proof is based on the fact that calculating a family of policies using the $\oplus$ for a set of policies scheme does not strictly refine any of them which violates the requirements of SDD. The detailed proof is in Appendix A.5 □

Lemma 5.2.4 states that calculating the family of policies for a node using the $\oplus$ scheme does not satisfy the SDD strategy. For example, in Figure 5.4 if $p(v_2) = p(v_4) \oplus p(v_5) \oplus p(v_6)$. The condition of the SDD strategy is that each child strictly refines the policy family $p(v_2)$. Therefore, $p(v_4) \sqsubset_{\mathcal{F}} p(v_4) \oplus p(v_5) \oplus p(v_6) \Leftrightarrow p(v_4) \sqsubset_{\mathcal{F}} p(v_4) \vee p(v_4) \sqsubset_{\mathcal{F}} p(v_5) \vee p(v_4) \sqsubset_{\mathcal{F}} p(v_6)$ should be true for $p(v_4)$, $p(v_5)$, and $p(v_6)$. Which is false and therefore the SDD strategy is not satisfied.

Lemma 5.2.5. *If we have a node that has only one child, then it is impossible to have a SDD implementation using $\oplus$ or* `gcd` *schemes.*

Proof. The proof is based on the fact that $\oplus$ or `gcd` are binary operations, and therefore we cannot calculate for a node with one child. □

Lemma 5.2.5 states that if we have a node that has one child. Then calculating its family of policies using the $\oplus$ or `gcd` schemes is not achievable. Therefore, not satisfying the requirements for SDD strategy. For example, in Figure 5.4 if calculating $p(v_1)$ using the `gcd` or $\oplus$ scheme is not possible. The solution is either we do not allow a node to have only one child or have a special case to handle this situation.

Based on the above results, we propose in the following chapter an approach to network segmentation that implements SDD.

5.3 Summary

In this chapter, we have presented a formal definition for the of DD strategy. Which can be used to assess the implementation of the strategy in a given network. Moreover, we have presented different approaches to generate policies for internal access control points from the resources policies based on the definition of DD. We have also introduced the SDD strategy and presented results related to the cases when it is not achievable.

Chapter 6

Network Segmentation

One of the most important strategies for building a secure network is *segmentation* or *compartmentalization*. Segmentation is essentially grouping resources in a way that provides maximum protection for the resources. This chapter presents a formal discussion of network segmentation and robust network architecture. Based on the network segmentation formalism we propose an algorithm that uses the SDD strategy to group resources systematically to achieve maximum protection. This results in a network configuration that is by construction secure from an access control perspective. Section 6.1 introduces the concept of network segmentation and its relevance to network security. Section 6.2 gives a formal definition for network segmentation and robust secure network. Section 6.3 presents an exponential and another polynomial algorithm to segment resources to reach a robust and secure network structure. Finally, Section 6.4 summarizes the chapter.

6.1 Network Segmentation

Organizations use access control points (e.g., firewalls) to protect their resources from threats, attacks, and untrusted sources. Access control points inspect network traffic (packets) and perform actions on it based on specified policies [CB94, VE05]. They control traffic

between networks with different security requirement levels. These networks can be separate networks such as the internet and the organization's internal network or internal subnetworks. Traditionally, organizations deploy a single access control point at the perimeter of their network which is not sufficient to protect internal resources. Therefore, organizations currently place access control points at multiple places in the network protecting subnetworks or segments [VE05].

In Chapter 1, we have introduced the concept of segmentation and its relation to network security. In Chapter 2, we have presented the research related to network segmentation. Essentially, segmentation [MHMR06] or compartmentalization [Sta07, Sta09] is partitioning the network into segments or subnetworks based on their security requirements such that resources are grouped with the ones that have similar security requirements. Traffic between these segments is controlled by access control points. The concept of segmentation has no formal definition in the literature [Sta07, WŞW$^+$16]. This chapter builds on the results presented in previous chapters to propose a formal approach for network segmentation.

6.1.1 Related Work and Motivation

Research, industry, and government security agencies recommend segmentation as a measure to protect resources and mitigate threats [Goo12, Nat13, Cen15]. However, network resources can be segmented in many possible ways, even for a small set of resources. The possibilities grow exponentially with the increase in the number of resources. With the absence of a formal approach to achieve segmentation, the task of finding the best segmentation is left to the expertise of network administrators. Moreover, research on the topic of segmentation is limited as presented in Chapter 2.

We discuss the relation between the research presented by Wagner et al. [WŞŞP$^+$17, WŞPS19, WŞW$^+$16] and our proposed approach in this chapter. In [WŞW$^+$16], they present a heuristic approach to segment network resources based on risk evaluation to reach optimal or near-optimal segmentation. Moreover, this proposal is high in cost and impractical for

large networks. However, our approach is a policy-based segmentation and fully automated to reach the best segmentation. In [WŞŞP+17], they develop a Markov chain model to evaluate a given segmentation architecture. This model can be used by network administrators to evaluate different segmentation architectures. Their solution is not for building network architectures but only to evaluate them. In [WŞPS19], they present an approach to segment network resources that is optimized for multiple objectives. These objectives are security, cost, and mission performance. This solution is fully automated. Our focus is on formalizing policy-based network segmentation, build an algorithm to achieve segmentation, and implement it in an actual setting.

Further, Rahman and Al-Shaer in [RAS13] propose an automated solution that decides the physical placements of security devices within the network and generates network security configurations. Given a network topology, security requirements, and business constraints as inputs, the solution writes the security design problem as a formula and solves it using the Satisfiability Modulo Theories (SMT) [dMB09]. This solution supports placing security devices in an existing network. In contrast to [RAS13], we build a network topology from scratch that achieves security goals by design rather than placing security devices on existing topologies.

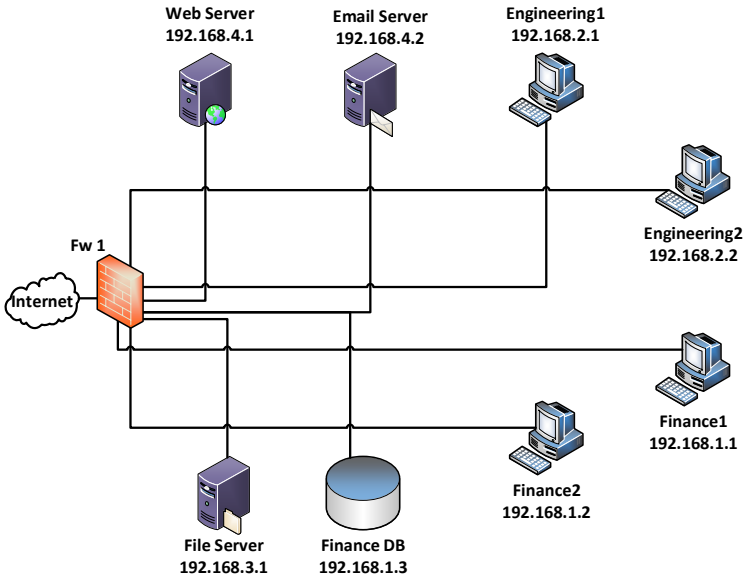
Therefore, although there exist some approaches to segment network resources and place network security devices on existing topologies, there is no formal approach to achieve it. Moreover, the notion of 'similarity' or 'different security levels' between resource policies has not been *precisely* articulated, which results in the inability to define, automate, and prove the correctness of the best segmentation. The above has motivated us to present a formal approach to network segmentation.

6.1.2 Illustrative Example Revisited

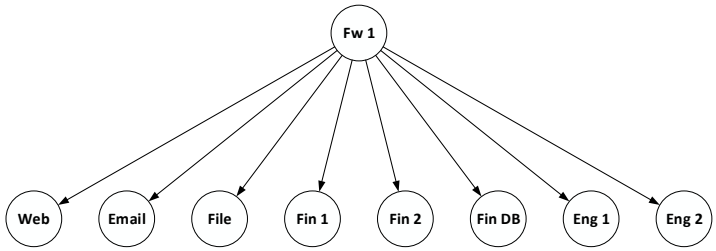
Recall the organization resources presented in the illustrative example in Chapter 3. Now, we consider three different network topological structures for these resources. The first

structure, shown in Figure 6.1, has an outer firewall *Fw1* that protects all the resources. Observe that this structure does not provide sufficient protection for resources as it suffers from many drawbacks. First, it keeps internal resources (e.g., *Engineering1*) vulnerable for outer attacks as it has a single layer of defense which the firewall *Fw1*. A successful attack that bypasses *Fw1* can attack other sensitive resources easily. Second, it does not protect from internal attacks initiated from within the organization. This is due to the fact that this structure assumes complete trust of internal traffic. In conclusion, this kind of structure and segmentation of resources is not sufficient to protect resources, and therefore there is a need for a better network structure to achieve a maximum protection for resources in which resources are segmented in a way that provides protection from external and internal threats.

As discussed in previous chapters, it is important to adhere to segmentation and DD strategies when building and designing a network structure. The second structure, shown in Figure 6.2, attempts to implement these strategies. It does this by placing resources into different segments protecting each segment with a firewall and by having multiple layers of firewalls to reach internal resources. This approach is not based on a formal approach and does not guarantee maximum protection. We observe, in Figure 6.2 that the outer firewall *Fw1* and internal *Fw3* would have the policy that allows traffic from the internet to *Web server* and *Email server*. Therefore, this structure makes the engineering workstations vulnerable to outer threats. An outside attacker can easily gain access to the segment and attack the engineering workstations. Therefore, although this structure implements segmentation and DD strategies, it does not provide sufficient protection for resources with strict security requirements. Therefore, simply adding extra layers of firewalls does not always guarantee extra protection without proper placement and configuration of network resources. More importantly, we are concerned with the ability to reach these conclusions for large networks, something not readily possible via only general guidelines and human judgment.



(a) A topological representation of network (1)

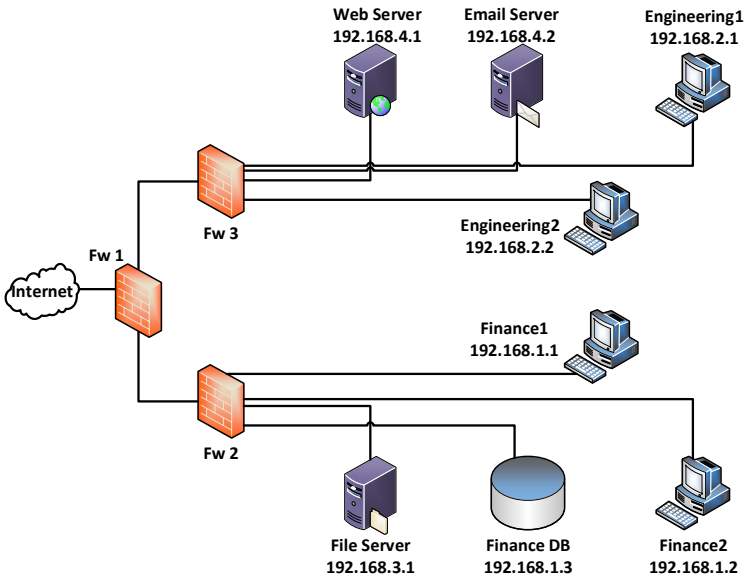


(b) A graph representation of network (1)

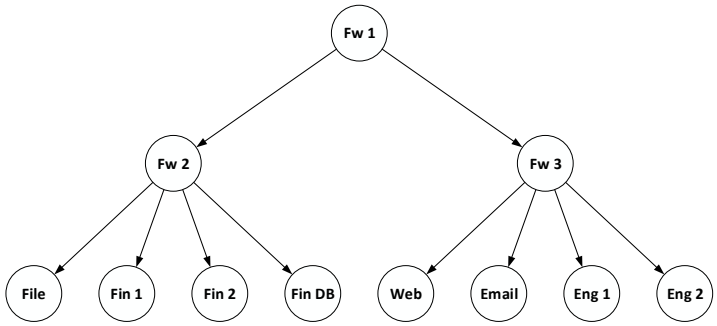
Figure 6.1: Network structure (1)

Finally, the third network structure, shown in Figure 6.3, correctly follows and implements the guidelines and best practices for network design [Pet01]. This structure is the ideal structure for the resources. However, following the guidelines and reaching such a structure might be achievable for a small network, it is unattainable for a large network with a huge number of resources with different policies using only human judgment, motivating the use of a formal and rigorous approach.

In this chapter, we present a formal approach to segment network resources into nested



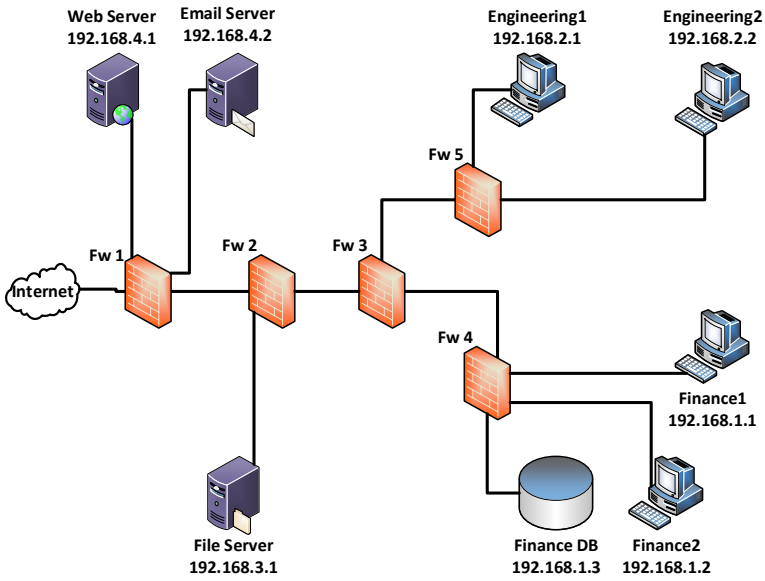
(a) A topological representation of network (2)



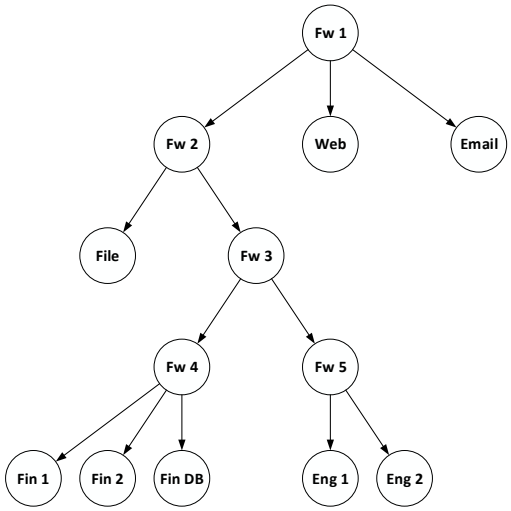
(b) A graph representation of network (2)

Figure 6.2: Layered network structure (2)

segments to provide maximum protection for these resources and the overall network. The strategy addresses the question of where to place these segments within the network to achieve the goal of reaching an optimal robust and secure structure. It also generates the policies to be enforced at the internal access control points in the network. The approach is based on the theory of PFA and the theory of *Guarded Commands* presented in Chapter 3. We also use the strategy of DD presented in Chapter 5.



(a) A topological representation of network (3)



(b) A graph representation of network (3)

Figure 6.3: Ideal network structure (3)

6.2 Segmentation and Robust Network Architecture

To assess whether an approach achieves the desired goal, it has to be defined formally. In this section, the definitions of network segmentation and robust network architecture are presented. In network segmentation, resources are grouped into segments or zones such that this segmentation provides maximum protection. Maximum protection for a resource is achieved when its placed within a segment that is protected by an access control point that enforces a policy family with the highest possible security level. The concept of network segmentation is defined based on the weight function introduced in Section 4.3. The weight function is used to quantify and measure the different security requirements of an access control policy.

Definition 6.2.1 (Segment)**.** *Let R be a set of resources. A set $S \subseteq R$ is said to be a segment of R iff*

$(\forall\, r, r' \mid r \in S \wedge r' \in (R - S) \cdot w_{\mathbf{P}}(\texttt{gcd}(p(r), p(r'))) \leq w_{\mathbf{P}}(\texttt{gcd}(\, r \mid r \in S \cdot p(r)\,))\,)$ ■

Definition 6.2.2 (Segmentation)**.** *Let R be a set of resources, and let F be a set of subsets of R such that $(\cup A \mid A \in F \cdot A) = R$. Then F is a segmentation of R iff*

$(\forall\, A \mid A \in F \cdot A$ *is a segment of* $R)$. ■

In our definition of segmentation, a resource r is placed in a segment S if and only if the `gcd` (i.e., commonality) of the policies of the resources in the segment S and r has more or equal weight than the `gcd` of r and any other resource out of S. Therefore, the `gcd` of the segment S, which is the policy enforced at the firewall controlling access to resources in the segment, provides maximum protection for the resources in the segment than any other policy specification. In Definition 6.2.1, a segment is forced to have at least two resources as the GCD is a binary operator. Note that segments can be nested and therefore a node can belong to more than one segment if the smaller one is within the other one.

Lemma 6.2.1.

(a) Let $S \subseteq R$ be a set of resources formed by only mutually co-prime policies that are $\neq 1_{\mathcal{F}}$. S forms a segment only if $(\forall\, r \mid r \in (R - S) \cdot p(r) = 1_{\mathcal{F}})$

(b) Let $S \subseteq R$, and there exists resources $r, r' \in S$ such that $p(r) = 1_{\mathcal{F}}$ and $p(r') \sqsubset 1_{\mathcal{F}}$, and

$$(\forall\, r, r' \mid r \in S \wedge r' \in (R - S) \cdot \texttt{gcd}(p(r), p(r')) = 1_{\mathcal{F}}).$$

Then S is a segment.

Proof. The detailed proof is given in Appendix A.6. □

By requiring DD, resources with $1_{\mathcal{F}}$ policy are directly attached to the root having $1_{\mathcal{F}}$ policy. If a segment satisfying the condition in Lemma 6.2.1(a) exists, then the root of this segment has $1_{\mathcal{F}}$ policy. Consequently, the root of the global network has $1_{\mathcal{F}}$ policy. To achieve SDD all the resources and internal nodes attached to the root of the segment S is attached to the global root. By Lemma 6.2.1(b), S is not a segment if $S \subseteq R$, there exists resources $r, r' \in S$, such that $p(r) = 1_{\mathcal{F}}$ and $p(r') \sqsubset 1_{\mathcal{F}}$, and there exists resources $r \in S$ and $r' \in (R - S)$ such that $\texttt{gcd}(p(r), p(r')) \neq 1_{\mathcal{F}}$. In other words, for a set of resources S that has some resources with $1_{\mathcal{F}}$ policy and others resources with policies different than $1_{\mathcal{F}}$. If the commonality of each resource in S with every resource out of S is $1_{\mathcal{F}}$, then S forms a segment. Otherwise, S is not a segment.

Superfluous Firewall Chain is a chain of firewalls where each firewall has only one child, that is a firewall, ending with a firewall that has resources or multiple firewalls attached to it. In this case, each firewall in the chain protects only one firewall aside from the last firewall. A superfluous firewall chain is a waste of network resources and therefore it is possible to remove all firewalls except the last one as it has the most restrictive policy.

Below, the definition of a robust network is presented. The definition is based on the segmentation defined above and the SDD strategy presented in Chapter 5. In a robust

network, the SDD strategy is used to place optimal segments within the network such that the generated topology is guaranteed to provide maximum access control protection employing compartmentalization and layered defense.

Definition 6.2.3 (Robust Network)**.** *A network graph G is said to be robust if the following criteria hold:*

1. *G satisfies SDD strategy in every path from the root to the parent of a resource,*
2. *G implements segmentation as defined in Definition 6.2.2, and*
3. *G has no superfluous firewall chain.*

■

The first criterion for a robust network ensures that taking any path from the root, the traffic is faced by stricter policies the deeper it gets into the network. This is achieved by forcing a strict refinement at every edge in the network graph. However, this condition is not required at an edge with a leaf node. It allows a leaf node (i.e., a resource) and its parent to have the same policy. This is seen when the `gcd` of resources in a segment equals a policy of one of the resources. The second criterion ensures that a resource belongs to the segment that is protected by the strongest possible policy (i.e., `gcd`). Then, using the SDD strategy the segments are placed at the appropriate depth based on their security requirement level that is the weight of their `gcd`. Such that, resources with high-security level requirements are placed deep in the network protected by multiple layers of defenses where each layer is stronger than the one proceeding it. And resources with lower security level requirements are placed closer to the root of the network. Therefore, traffic going from outside the network to an internal resource is faced by multiple access control points ensuring that only legitimate traffic is allowed in. Moreover, lateral traffic from segment to segment is controlled by internal access control points such that authorized users are allowed only to the segment needed to perform their job and denied access to other segments. Therefore,

ensuring the protection of resources from internal and external threats. Finally, the third criterion ensures that the robust network is obtained with minimum cost.

6.3 Network Segmentation Algorithms

In this section, we present the network segmentation algorithms to build robust networks. We start by presenting an exponential algorithm then proceed to present a polynomial version of the algorithm. The purpose of starting with the exponential algorithm is that it is easy to grasp the use of the formalism presented in constructing the network graph.

6.3.1 Exponential Network Segmentation Algorithm (Exp-RNS)

In this section, we present the Exponential Robust Network and Segmentation Algorithm (Exp-RNS) which is a brute force algorithm to build a robust network having an exponential running time. Given a set R of resources and their policies, the Exp-RNS algorithm uses R to build a robust network by computing the GCDs of resource policies and using the refinement relation on these GCDs. For simplicity, we discuss the algorithm using singleton families of policies, and so we refer to them as policy rather than a family of policies.

A resource having $0_{\mathcal{F}}$ is essentially inaccessible, as it cannot be protected by an access control system. The $0_{\mathcal{F}}$ which represents a pseudo policy cannot be enforced at an access control point. Therefore, it cannot be part of robust network architecture. Hence, we assume that the input R does not contain resources with $0_{\mathcal{F}}$ policies. Furthermore, by Lemma 5.2.1, having any internal node/leaf with $1_{\mathcal{F}}$ policy implies that SDD is not achievable in the network graph. By Lemma 6.2.1(a) resources with $1_{\mathcal{F}}$ policy have to be attached to the root with $1_{\mathcal{F}}$ policy. Therefore, in the Exp-RNS algorithm, we remove resources with $1_{\mathcal{F}}$ policy a priori from the set R, and as a final step add these resources to the root.

The Exp-RNS algorithm consists of five main tasks. The first task is calculating the GCDs or common policies. The second task is building a network graph based on the refinement

relation consisting of only firewalls. The third task is adding resources to the network graph, and the fourth task is pruning the network graph to remove superfluous firewall chaining. Finally, the fifth task is to add all the resources with $1_{\mathcal{F}}$ policy (if they exist) to the graph.

Algorithm 1 Exponential Robust Network and Segmentation Algorithm

```
1:  procedure EXP-SEGMENTATION(R)
2:      S ← set of resources having 1_F policy
3:      R ← R − S
4:      GCD ← COMPUTE-GCD(R)
5:      G ← BUILD-NETWORK-GRAPH(GCD, R)
6:      ADD-RESOURCES-TO-NET(G, GCD, R)
7:      OPTIMIZE-NETWORK-GRAPH(G)
8:      if S ≠ ∅ then
9:          ADD-1F-RESOURCES(G, S)
10:     end if
11: end procedure
```

In the subsequent sections, we will explain each of the main steps of the above algorithm.

Calculating GCDs

The procedure COMPUTE-GCD computes the commonalities or the GCD of policies of resources. As seen in Section 6.2, to segment resources we need to place them within a segment such that maximum security (access-protection) is achieved. To achieve this, we use a brute force approach to compute all possible segments by computing the power set of R, $\mathcal{P}(R)$, (where each set in $\mathcal{P}(R)$ represent a possible segment). Then we compute the commonalities in each segment by computing the GCD of the resource policies in that segment.

procedure COMPUTE-GCD(R)

$\mathcal{P}(R) \leftarrow$ power set of R minus all singleton sets and the empty set.

GCD $\leftarrow \emptyset$ ▷ initialize **GCD** set.

for each $s \in \mathcal{P}(R)$ **do**

Create *gcd_s* object

gcd_s.p $\leftarrow$ Calculate the GCD of the policies of resources in s

```
        gcd_s.weight ← weight of gcd_s.p
        flag ← false
        for each gcd ∈ GCD do
            if gcd.p = gcd_s.p then
                flag ← true
                if gcd.size < |s| then
                    gcd.size ← |s|
                    gcd.set ← s
                end if
            else if gcd.p ⊏ gcd_s.p ∧ gcd.weight = gcd_s.weight then
                GCD = GCD − {gcd}
            else if gcd_s.p ⊏ gcd.p ∧ gcd.weight = gcd_s.weight then
                flag ← true
            end if
        end for
        if flag = false then
            gcd_s.set ← s
            gcd_s.size ← |s|
            gcd_s.weight ← weight of gcd_s.p
            GCD ← GCD ∪ {gcd_s}
        end if
    end for
    return GCD
end procedure
```

We now explain the COMPUTE-GCD procedure in detail. The procedure first computes the power set of the set of resources R, $\mathcal{P}(R)$. It then computes the GCD for each set in $\mathcal{P}(R)$. While computing the GCDs, the procedure maintains a set **GCD**, consisting of

objects called the *gcd* objects. Each *gcd* object has the following attributes: p, *set*, *weight*, and *size*. Attribute *set* represents the set of resources, which is a set in $\mathcal{P}(R)$. Attribute p represents the GCD of the policies of the resources in *set*. Attribute *weight* is the weight of the policy p, and attribute *size* represents the cardinality of *set*. To minimize the number of *gcd* objects stored in **GCD**, each object in **GCD** has a unique policy p. To achieve this, if the GCD of two sets in $\mathcal{P}(R)$ is the same, then we store the *gcd* object corresponding to the larger set. Further for any two sets $s_1, s_2 \in \mathcal{P}(R)$, if the GCD of policies of s_1 refines the GCD of policies of s_2, and if both of these policies have the same weight, then we only store the *gcd* object corresponding to s_2. Table 6.1 gives all the *gcd* objects in the **GCD** set, and the set of resources in *set* for the illustrative example after procedure COMPUTE-GCD is executed. In Appendix-D, we present the computation of policies for all the *gcd* objects given in Table 6.1.

gcd objects in **GCD** set	Set of Resources
gcd 1	$\{Web\_server, Email\_server, File\_server, Fin.DB, Fin.1, Fin.2, Eng.1, Eng.2\}$
gcd 2	$\{File\_server, Fin.DB, Fin.1, Fin.2, Eng.1, Eng.2\}$
gcd 3	$\{Eng.1, Eng.2, Fin.DB, Fin.1, Fin.2\}$
gcd 4	$\{Eng.1, Eng.2\}$
gcd 5	$\{Fin.DB, Fin.1, Fin.2\}$

Table 6.1: *gcd* objects in **GCD** set, and their corresponding set of resources

Building a Network Graph

The procedure BUILD-NETWORK-GRAPH builds a network graph G with no resources attached. This graph is a temporary network graph consisting of only firewalls and might contain a superfluous firewall chain. However, the extra firewalls are removed at a later stage (by the OPTIMIZE-NETWORK-GRAPH). The procedure uses the **GCD** set = F ordered in decreasing order by weights, and the refinement relation to produce the network

graph. The vertices of the network graph are partitioned into *gcd nodes* representing a *gcd* object (representing firewalls), and *resource nodes* representing resources in the network. Each node in the graph has all the attributes defined for a *gcd* object, and in addition to these, it has two more attributes: $is\_resource$ and π. The attribute $is\_resource$ identifies whether the node is a resource node or a *gcd* node. It is set to $true$ if the node represents a resource, otherwise, it is set to $false$. The attribute π stores the set of the parent(s) of the node.

The root r is the *gcd* object having the least weight and its set consists of all resources (except the resources with $1_{\mathcal{F}}$ policy). It is first added to the network graph G. After which nodes in F having the same weight as the root r are removed. Then, it computes w_{max} weight - the maximum weight over the weights of all nodes in F, and computes the set T consisting of all nodes with w_{max} weight and removes them from F. Then each node in the set T is added to G using the ADD-NODESET-TO-G procedure. The last two steps are repeated till $F = \emptyset$; that is, the last two steps are repeatedly executed to (possibly) add all the nodes in F to G.

procedure BUILD-NETWORK-GRAPH(F, R)

$G \leftarrow \emptyset$ ▷ $G = (E; V; r)$

$r \leftarrow$ `gcd` object that has all resources R in its *set* attribute

$r.\pi \leftarrow \emptyset; r.resource \leftarrow false$

$V \leftarrow V \cup \{r\}$

$F \leftarrow F-$ minus *gcd* objects with the same weight as the root r

while $F \neq \emptyset$ **do**

$w_{max} \leftarrow$ maximum weight of any $s \in F$.

$T \leftarrow \emptyset$

for each $s \in F$ **do**

if $s.weight = w_{max}$ **then**

$T \leftarrow T \cup s; F \leftarrow F - s$

```
            end if
        end for
        ADD-NODESET-TO-G(G, F, T, w_max)
    end while
    return G
end procedure
```

Adding *gcd* Nodes in T to the Network Graph The *gcd* nodes are added to the network graph based on the weight attribute of the *gcd* object. We add nodes to G in batches (set T) starting with maximum weight to minimum weight. The ADD-NODESET-TO-G procedure adds the set of nodes in T in non-increasing order by their *size* attribute; that is, nodes in T having the largest *size* is processed first, followed by a node with the next (or same) largest size is processed to add it to G. During the execution of the procedure, at all times we maintain the $resT$ set that contains the set of resources for which *gcd* nodes have not been added. Initially, $resT$ contains all resources contained in all the *gcd* nodes in T. Then the procedure loops through each node s in T and if $s.set$ still contains resources unaccounted for in G, it is evaluated. During this evaluation, every child c of the root r that refines s is added to *cset*. Then v is added to G by attaching it to r. If *cset* is not empty, then all nodes in it are attached to v as its children, and their connection to r is removed.

```
procedure ADD-NODESET-TO-G(G, F, T, w_max)          ▷ G = (V; E; r)
    resT ← ∅
    for each s ∈ T do
        resT ← resT ∪ s.set
    end for
    T ← T nodes ordered in non-increasing order by their size attribute.
    for each s ∈ T ∧ (s.set ∩ resT ≠ ∅) do
```

$cset \leftarrow \emptyset$

for each child c of r **do**

if $c.p \sqsubset s.p$ **then**

$cset \leftarrow cset \cup \{c\}$

end if

end for

$V \leftarrow V \cup \{s\}$

$E \leftarrow E \cup \{(r, s)\}$

$s.\pi \leftarrow \{r\}$

if $cset \neq \emptyset$ **then**

for each $c \in cset$ **do**

$E \leftarrow E - \{(r, c)\} \cup \{(s, c)\}$

$c.\pi \leftarrow c.\pi - \{r\} \cup \{s\}$

end for

end if

$resT \leftarrow resT - s.set$

end for

end procedure

Figure 6.4 shows the network graph for the illustrative example produced by the procedure Build-Network-Graph. Observe that in the graph each path represents a refinement chain (e.g., $gcd4 \sqsubset_{\mathcal{F}} gcd3 \sqsubset_{\mathcal{F}} gcd2 \sqsubset_{\mathcal{F}} gcd1$).

Adding resources to the Network Graph

The procedure Add-Resources-to-Net adds resource nodes to the network Graph G. While adding the resource nodes, we first order the *gcd* objects in **GCD** in descending order by weight ($\mathbf{GCD_{max}}$). Then we choose the *gcd* node having the maximum weight and attach all the resources in its *set* attribute, and in R' (initially $R' = R$), to it. After which we remove the attached resources from the set of resources R'. We do this for each

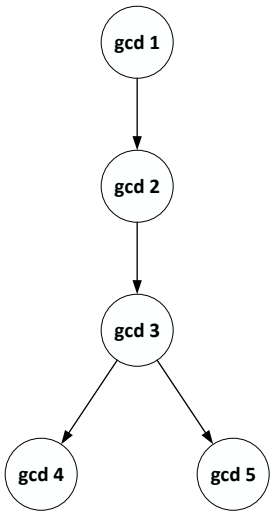


Figure 6.4: The network graph of the illustrative example after BUILD-NETWORK-GRAPH procedure

gcd node in $\mathbf{GCD_{max}}$. In the end, all resources are added to the network graph G. Observe that by adding resources to *gcd* nodes starting with maximum weight to minimum weight, we ensure that resources with higher security requirements are protected with maximum layers of protection.

procedure ADD-RESOURCES-TO-NET($G, \mathbf{GCD}, R$)

$\mathbf{GCD_{max}} \leftarrow$ ordered list of $gcd \in \mathbf{GCD}$ in descending order by weights.

$R' = R$

for each $gcd \in \mathbf{GCD_{max}}$ **do**

for each $s \in gcd.set$ **do**

if $s \in R'$ **then**

$s.is\_resource \leftarrow true$

$s.\pi \leftarrow s.\pi \cup \{gcd\}$

$V \leftarrow V \cup \{s\}$

$E \leftarrow E \cup \{(gcd, s)\}$

$R' \leftarrow R' - \{s\}$

end if

end for

end for

end procedure

Optimize GCD graph

The network graph obtained after executing the procedure ADD-RESOURCES-TO-NET could suffer from superfluous firewall chaining. To eliminate this issue, we use the OPTIMIZE-NETWORK-GRAPH procedure. This procedure performs a post-order traversal of G using two stacks S and T so that all children of a node are evaluated before it gets evaluated. Since our graph G is not necessarily binary, we do not always evaluate the children of a node from left to right (any order is acceptable). The only criteria we follow is that all children of a node are evaluated before evaluating it. S is used to keep track and evaluate all children of a node before itself, and T is used to store the post-order traversal of G. When S is empty (and T is full), we pop nodes from T one at a time and begin evaluating them. When a node u is being evaluated, if u is a *gcd* node and it has no children or if u has only one child that is a *gcd* node, then u and its corresponding edges are deleted from G. If u has only one child that is a *gcd* node, then the child is attached to u's parent.

procedure OPTIMIZE-NETWORK-GRAPH(G) ▷ $G = (V; E; r)$

$S \leftarrow \emptyset;\ T \leftarrow \emptyset$ ▷ S, T are stacks

$Push(S, r)$

while $S \neq \emptyset$ **do**

$u \leftarrow Pop(S)$

$Push(T, u)$

for each c child of u **do**

$Push(S, c)$

end for

end while

while $T \neq \emptyset$ **do**

$u \leftarrow Pop(T)$

$child\_count \leftarrow 0$

$f\_flag \leftarrow false$

for each c child of u **do**

$child\_count \leftarrow child\_count + 1$

if $c.is\_resource = false$ **then**

$f\_flag \leftarrow true$

end if

end for

if $child\_count = 0 \wedge u.is\_resource = false$ **then**

$V \leftarrow V - \{u\}$

for each $x \in u.\pi$ **do**

$E \leftarrow E - \{(x, u)\}$

end for

else if $(child\_count = 1 \wedge f\_flag = true)$ **then**

$V \leftarrow V - \{u\}$

$c.\pi \leftarrow c.\pi - \{u\}$

$c.\pi \leftarrow c.\pi \cup \{u.\pi\}$

$E \leftarrow E - \{(u, c)\}$

for each $x \in u.\pi$ **do**

$E \leftarrow E - \{(x, u)\}$

$E \leftarrow E \cup \{(x, c)\}$

end for

end if

end while

end procedure

The robust network graph with resources attached for the illustrative example is shown in Figure 6.5.

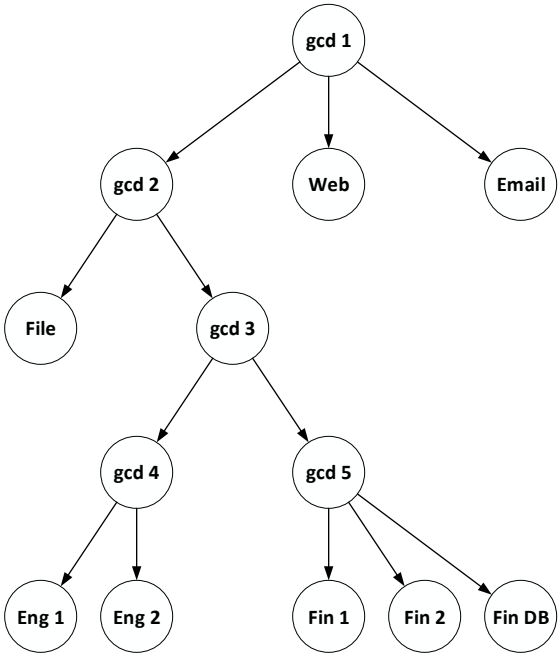


Figure 6.5: The network graph of the illustrative example after adding resources

Adding resources with policy $1_{\mathcal{F}}$ to G

If $S \neq \emptyset$; that is, the set of resources having $1_{\mathcal{F}}$ policy is not empty, we add it to the network graph using the procedure ADD-1F-RESOURCES outlined as follows:

1. If the policy of the root r is equal to $1_{\mathcal{F}}$, then we directly attach all the resources in the set S to r.

2. However, if the policy of the root r is not equal to $1_{\mathcal{F}}$, then we create a new node r', set its attributes appropriately (as shown in Procedure ADD-1F-RESOURCES), and

add it to G. After which we attach r and all the resources in the set S to it. The node r' is the new root for the network graph G.

procedure ADD-1F-RESOURCES(G, S)

if $S = \emptyset$ **then return** *error*

end if

if $r.p = 1_{\mathcal{F}}$ **then**

$r.set \leftarrow r.set \cup S$; $r.size \leftarrow r.size + |S|$

for each $s \in S$ **do**

$s.is\_resource \leftarrow true$

$s.\pi \leftarrow \{r\}$

$V \leftarrow V \cup \{s\}$

$E \leftarrow E \cup \{(r, s)\}$

end for

else

Create node r' ▷ Create new root r' of G

$r'.set \leftarrow r.set \cup S; r'.size \leftarrow r.size + |S|$

$r'.p \leftarrow 1_{\mathcal{F}}; r'.weight \leftarrow$ weight of $1_{\mathcal{F}}$ policy

$r'.\pi \leftarrow \emptyset$

$V \leftarrow V \cup \{r'\}$ ▷ Add new root r' to G

$r.\pi \leftarrow \{r'\}$ ▷ Attach r to r'

$E \leftarrow E \cup \{(r', r)\}$

for each $s \in S$ **do**

$s.is\_resource \leftarrow true$

$s.\pi \leftarrow \{r'\}$

$V \leftarrow V \cup \{s\}$

$E \leftarrow E \cup \{(r', s)\}$

end for

end if

end procedure

Exp-RNS Algorithm: Robustness and complexity

The running time of the Exp-RNS algorithm is exponential in the size of the set of resources R. This is due to the brute force approach used to compute the *gcd* nodes, obtained by computing the power set $\mathcal{P}(R)$, and using it to build the network graph.

Theorem 6.3.1. *The Exp-RNS algorithm constructs a robust network graph.*

Proof. The proof is straightforward and is by contradiction. We assume that at least one of the three criteria required for a network graph to be robust (Section 6.2), is not satisfied. Suppose Condition 1 is not satisfied. Then there exists at least one path in G from the root to the parent of a resource, such that the SDD strategy is not satisfied. Recall that the **GCD** set computed by the COMPUTE-GCD procedure has *gcd* objects with distinct policies. In Exp-RNS, *gcd* nodes are added to G only by the BUILD-NETWORK-GRAPH and ADD-1F-RESOURCES procedures. In the BUILD-NETWORK-GRAPH procedure, a *gcd* node s is first added to G, and any child of r that refines s is disconnected from the root and attached as a child to s. In both these cases, the SDD strategy is satisfied. In the ADD-1F-RESOURCES procedure, the only *gcd* node added to G is the root r' having $1_{\mathcal{F}}$ policy, and is added only if $r.policy \neq 1_{\mathcal{F}}$. Clearly, SDD strategy is satisfied as all policies $\neq 1_{\mathcal{F}}$ strictly refine $1_{\mathcal{F}}$. Therefore, every path from the root to a resource's parent satisfies the SDD strategy, which is a contradiction.

Suppose Condition 2 is not satisfied. Then there exists a segment S, such that a resource $r \in S$ and a resource $r' \notin S$ have commonalities with more weight than the weight of the commonalities between elements of the segment S. If this holds then there will be a *gcd* node in G with the policy $\texttt{gcd}(r, r')$ and its corresponding weight. However, since we add

resources to *gcd* nodes starting with *gcd* nodes having maximum weight to the *gcd* nodes having minimum weight, the resources r and r' would be added to the node having the policy $\texttt{gcd}(r, r')$, which is a contradiction.

Suppose Condition 3 is not satisfied; that is, superfluous firewall chaining exists in G. Then there exists at least one firewall with a single node that is a firewall attached. Recall that in the procedure Optimize-Network-Graph, we count the number of children attached to each firewall. If the node has only one child that is a firewall, we remove this node and attach its child to its parent(s). Therefore, by the end of the Exp-RNS algorithm, G has no superfluous firewalls, which is a contradiction. □

6.3.2 Robust Network and Segmentation Algorithm (RNS)

In this section, we propose RNS algorithm which a polynomial algorithm to build a robust network. This version of the algorithm is obtained in collaboration with other researchers and is published as joint work in [MAK21]. Given a set R of resources with their policies, RNS builds a robust and secure network. Similar to Exp-RNS, for this algorithm also we assume that R does not contain resources with $0_{\mathcal{F}}$ policies.

The idea of the algorithm is simple. We use an approach similar to the one used in the Build-Network-Graph procedure presented in Section 6.3, where the *gcd* nodes are added in non-increasing order of weights (except for the root). However, to achieve a polynomial running time, we use resources and their weights and create temporary nodes to guide us through the network building process.

In RNS algorithm, the root is added first to G, and nodes are added in batches (set T) to G in decreasing order of their weights. Hence at any given time, when a node s is evaluated to see where in G it can be added, the weight of all the nodes in G is greater or equal $s.weight$. While adding s to G, clusters of resources are evaluated to see if they can form a segment containing s based on their weights. Note that, the weight of any segment containing s is always less or equal to $s.weight$, and so the maximum weight of any segment containing s

is *s.weight*. Therefore, when such a cluster of nodes is identified, we create a *gcd* node to protect this segment and add it to the root of G and attach the nodes forming the segment to it. Otherwise, we create a temporary node for the cluster and add it to F, so that it can be evaluated later when the set having the weight of the cluster is being evaluated.

Algorithm 2 Robust Network and Segmentation Algorithm

```
1: procedure SEGMENTATION(R)                    ▷ R = {r_1, r_2, ..., r_n}
2:     G ← NULL                                 ▷ G = (V, E, r)
3:     r ← CREATE-NODE(R)                       ▷ Create root r
4:     ADD-NODE-TO-G(G, r, ∅, false)            ▷ Add root r to G
5:     S_1, S_2, ..., S_m ⊂ R such that ⋃_{i=1}^{m} S_i = R, and no two subsets have resources with
       same policies
6:     F = ∅
7:     for each s ∈ {S_1, S_2, ..., S_m} do
8:         F = F ∪ CREATE-NODE(s)
9:     end for
10:    while F ≠ ∅ do
11:        w_max ← maximum weight of any s ∈ F.
12:        T ← ∅
13:        for each s ∈ F do
14:            if s.weight = w_max then
15:                T ← T ∪ s; F ← F − s
16:            end if
17:        end for
18:        ADD-NODESET-TO-G(G, F, T, w_max)
19:    end while
20: end procedure
```

Note that, temporary nodes are never added to G. While evaluating a temporary node, if the cluster of nodes with s actually forms a segment at that point, a new *gcd* node is created and added to G. This is similar to any other *gcd* node. After which, the temporary node is deleted from F. We now give the outline of the algorithm as follows:

- The algorithm first creates the root r for all resources in R using the CREATE-NODE procedure and adds it to the network graph G.
- Then it creates subsets $S_1, S_2, \ldots, S_m \subset R$, such that no two subsets have resources

with same policies, and creates a node for each subset S_i, $1 \leq i \leq m$, and adds it to F. Therefore, F contains either a single resource node or a *gcd* node having resources in its set. If we assign distinct values from $1-m$ termed as *key values*, to each distinct policy in R, then the subsets $S_1, S_2, \ldots, S_m$ can be easily created using Bucket sort as follows: we create m buckets labeled from $1-m$, for each distinct policy. As we traverse through the policies in R, we simply evaluate its label and place it in the bucket having the same label.

- After which, it computes the w_{max} weight - the maximum weight over the weights of all nodes in F, and computes the set T consisting of all nodes with w_{max} weight and removes all these nodes from F.
- Then the nodes in set T are added to G using the ADD-NODESET-TO-G procedure.
- The above two steps are repeated till $F = \emptyset$; that is, the above two steps are repeatedly executed to (possibly) add all the nodes in F to G.

procedure ADD-NODESET-TO-G(G, F, T, w_{max}) ▷ $G = (V; E; r)$
  if $w_{max} = r.weight$ **then** ▷ Attach resources having same weight as r to r
    ATTACH-RESOURCES-TO-R(G, T)
  else
    for each $s \in T$ **do** ▷ Add permanent nodes in T to G
      if $s.temp = false$ **then**
        ADD-NODE-TO-G($G, s, \emptyset, false$)
      end if
    end for
    for each $s \in T$ **do** ▷ Check child nodes of r to add *gcd* node
      $cset \leftarrow \emptyset$
      $cset \leftarrow$ CURR-MAX-WEIGHT-SET(G, s)

```
        if cset ≠ ∅ then
            gcd = CREATE-NODE(cset)
            if !s.temp then
                if (s.weight = gcd.weight) then
                    ADD-NODE-TO-G(G, gcd, cset, true)
                else                ▷ Mark gcd as a temporary node
                    gcd.temp ← true; gcd.set ← s
                    gcd.size ← 1
                    F ← F ∪ {gcd}
                end if
            else
                if gcd.weight = s.weight then
                    ADD-NODE-TO-G(G, gcd, cset, true)
                end if
            end if
        end for
    end if
end procedure
```

The ADD-NODESET-TO-G procedure adds the set of nodes in T to G. The outline of this procedure is as follows:

- If $w_{max} = r.weight$, the ADD-NODESET-TO-G procedure directly attaches to r all the resources in every non-temporary node $s \in T$ and exits. Otherwise, it implements the below steps.
- The procedure loops through each node in T to (possibly) add it to G. If s is not a temporary node (indicated by the *temp* node attribute), it is added to G as a child of

the root r.

- Then every child of r is evaluated to compute *cset* - the set consisting of all nodes including s (or including $s.set$, if s is a temporary node). *cset* is computed by the CURR-MAX-WEIGHT-SET function.

- A new *gcd* node is computed for the set of resources in *cset*, if $cset \neq \emptyset$. If s is a permanent node, then it checks if $gcd.weight = s.weight$. If it is, *gcd* is added to G; otherwise *gcd* is tagged as "temporary" – to be considered for evaluation later – and added to F. If s is a temporary node and if the node is in $s.set$ is still part of the segment formed by *gcd*, then *gcd* is added to G. Otherwise, it is automatically removed from F, as all the nodes in T are removed from F, before adding them to G in Algorithm 2.

Recall that nodes are added in batches (T) to G in decreasing order of their weights. Therefore, in the ADD-NODESET-TO-G procedure when $gcd.weight < s.weight$, the node *gcd* is not added to G, but instead added to F (if it is not a temporary node), to be evaluated when the set of nodes (T) having weight = $gcd.weight$ is processed. Furthermore, temporary nodes are associated with a node for which they were created. For example, when a permanent node s from T is added to G, we check to see if any child nodes of r can form a segment with s. If the weight of this segment is less than the weight of s, we create a temporary node for the segment and evaluate it later when the nodes with its weight are being evaluated.

During the execution of the ADD-NODESET-TO-G function, while evaluating a set $s \in T$, let the root r have $c_1, c_2, \ldots, c_k$ child nodes. Let the GCD of the policies of $(c_1, s), (c_2, s), \ldots, (c_k, s)$ be $g_1, g_2, \ldots, g_k$. Let *cmax* be the maximum weight over all the weights of the the policies $g_1, g_2, \ldots, g_k$. Then the primary objective of the CURR-MAX-WEIGHT-SET function is to compute the set (not necessarily the largest) of all child nodes $smax = \{c_{i1}, c_{i2}, , \ldots c_{il}\}$, such that $smax \subseteq \{c_1, c_2, \ldots, c_k\}$, and $\texttt{gcd}(c_{i1}, c_{i2}, \ldots c_{il}, s)$ has *cmax* weight.

To compute $smax$, CURR-MAX-WEIGHT-SET function maintains a stack S (this can also be achieved with an array/list). Initially, S is empty and $cmax = 0$. While evaluating the child nodes of r, S contains the nodes having the current/local maximum weight. Therefore, by the end of the **for** loop, S contains all the child nodes, such that the weight of the GCD of their policies is $cmax$. $smax$ contains the largest possible set of nodes if $cmax = s.weight$. However, if $cmax \neq s.weight$, it does not necessary store the largest set. This is not a problem, as a node corresponding to $cmax$ is added as a temporary node (if s is a permanent node) to F, in which case it is evaluated later and the maximum possible set of nodes corresponding to it, is formed at that time.

Note that, we do not consider s while computing this set, because $\texttt{gcd}(s, s) \geq \texttt{gcd}(c_i, s)$, where $1 \leq i \leq k$. Hence, if s is included in the set, $cmax = s.weight$. This would be a problem if $smax = \{s\}$, as we cannot form a segment with only one element. Therefore, we compute S without considering s and if $S \neq \emptyset$ we add s to it. Furthermore, if $cmax$ is equal to the weight of the root, it is pointless to create another sub segment with its weight $= r.weight$. Therefore, we disregard this case.

function CURR-MAX-WEIGHT-SET(G, s)

$S \leftarrow \emptyset; snew \leftarrow \emptyset$

$Sgcd \leftarrow \emptyset$ ▷ gcd of policies of the nodes in S

$cmax \leftarrow 0$ ▷ Current Max.weight of gcd of policies

$flag \leftarrow false$

for each child c of r **do**

if $s.temp \wedge c \in s.set$ **then**

$flag \leftarrow true$

end if

if $(!s.temp \wedge c \neq s) \vee (s.temp \wedge c \notin s.set)$ **then**

$tempp \leftarrow gcd(c.p, s.p)$

$tempw \leftarrow$ weight of $tempp$

```
            if cmax < tempw then
                Empty S; Push(S, c)
                Sgcd ← gcd(s, c)
                cmax ← tempw
            else if cmax = tempw then
                Sgcd ← gcd(Sgcd, c)
                Sgcdweight ← weight of Sgcd
                if Sgcdweight = cmax then
                    Push(S, c)
                end if
            end if
        end if
    end for
    if !S.isempty ∧ cmax ≠ r.weight ∧ s.temp ∧ flag then
        Push(S, s.set)
        while S ≠ ∅ do
            snew ← snew ∪ Pop(S)
        end while
    end if
    if !S.isempty ∧ cmax ≠ r.weight ∧!s.temp then
        Push(S, s)
        while S ≠ ∅ do
            snew ← snew ∪ Pop(S)
        end while
    end if
    return snew
end function
```

The ATTACH-RESOURCES-TO-R procedure attaches resources having the same weight as r to r.

procedure ATTACH-RESOURCES-TO-R(G, T) ▷ $G = (V; E; r)$
  for each $s \in T \wedge !s.temp$ **do**
    if $!s.isresource$ **then**
      for each $res \in s.set$ **do**
        $res =$ CREATE-NODE(res)
        ADD-NODE-TO-G$(G, res, \emptyset, false)$
      end for
    else
      ADD-NODE-TO-G$(G, s, \emptyset, false)$
    end if
  end for
end procedure

The ADD-NODE-TO-G procedure encapsulates the steps to add nodes (and any resources not already added to G in its set attribute) to G. The ADD-NODE-TO-G procedure always adds the node s to the root of r (if it exists). If s is a *gcd* node and has children that are resource nodes not added to G yet (indicated by $exist = false$), then s is added to G and all the resources in its $s.set$ are created as new nodes and added to G as children of s. If s is a *gcd* node object, which is created to be a *gcd* node for existing nodes in G (indicated by $exist = true$), then s is added to G and all nodes in *cset* are added as its children (this includes removing their connection with r and attaching them to s). In this case, *cset* can contain a node having the same weight as that of s, in which case, we delete that node from G, and attach its children to s.

procedure ADD-NODE-TO-G$(G, s, cset, exist)$
  if $G \neq NULL$ **then**

$V \leftarrow V \cup s; E = E \cup \{(r, s)\}$

$s.\pi \leftarrow s.\pi \cup \{r\}$

else

if $s.set = R$ **then**

$V \leftarrow V \cup s; r \leftarrow s$

end if

end if

if $s.set \neq empty \wedge s \neq r \wedge !exist$ **then** ▷ Add *gcd* node with only resources not yet added to G as children

for each $res \in s.set$ **do**

$res =$ CREATE-NODE$(res, \emptyset, false)$

$V \leftarrow V \cup res$

$E \leftarrow E \cup \{(s, res)\}$

$res.\pi \leftarrow res.\pi \cup \{s\}$

end for

else if $cset \neq empty \wedge s \neq r \wedge exist$ **then** ▷ Adding *gcd* node with children already added to G and possibly having *gcd* nodes as children.

for each $res \in cset$ **do**

if $res.isresource = false \wedge s.weight = res.weight$ **then**

$V \leftarrow V - \{res\}$

$E \leftarrow E - \{(r, res)\}$

for each child c of res **do**

$E \leftarrow E - \{(res, c)\} \cup \{(s, c)\}$

$res.\pi \leftarrow res.\pi - \{res\} \cup \{s\}$

end for

else

$E \leftarrow E - \{(r, res)\} \cup \{(s, res)\}$

$res.\pi \leftarrow res.\pi - \{r\} \cup \{s\}$

end if

end for

end if

end procedure

The CREATE-NODE function creates nodes so that they can be added to the graph G. While building G, two types of nodes, namely the resource nodes, and the GCD (firewall) nodes are created, and their attributes set according to the code given in the CREATE-NODE function. Since we mark nodes as temporary (nodes that are added to F and evaluated later) we introduce a new attribute –*temp*– which is added to all nodes. For permanent nodes, *temp* is set to false, and for temporary nodes, *temp* is set to true.

function CREATE-NODE(s)

Create node v ▷ Create node and initialize attribute values

if $|s| = 1 \wedge s \neq R$ **then** ▷ Adding a resource

$v.p \leftarrow$ policy of resource s

$v.weight \leftarrow$ weight of $v.p$

$v.set \leftarrow \emptyset; v.size \leftarrow 0$

$v.isresource \leftarrow true; v.temp \leftarrow false$

else

$v.p \leftarrow$ GCD of all resource policies in s

$v.weight \leftarrow$ weight of $v.p$

$v.isresource \leftarrow false; v.temp \leftarrow false$

for each $node \in s$ **do**

if $!node.isresource$ **then**

$v.set \leftarrow v.set \cup node.set$

$v.size \leftarrow v.size + node.size$

```
            else
                v.set ← v.set ∪ node
                v.size ← v.size + 1
            end if
        end for
    end if
    return v
end function
```

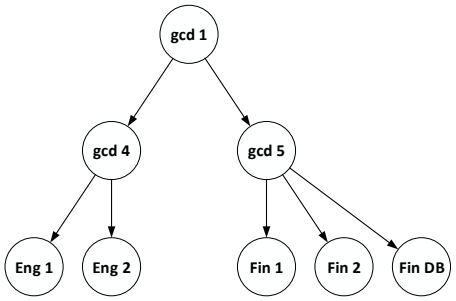
Figure 6.6 shows the progression of executing the RNS algorithm on the illustrative example. The last iteration generates the network shown in Figure 6.5.

RNS Algorithm: Robustness and complexity

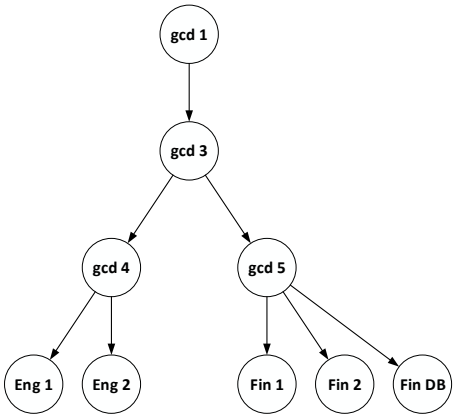
Theorem 6.3.2. *The RNS algorithm constructs a robust network graph.*

Proof. The proof is by contradiction. It is also given in [MAK21]. We assume that at least one of the three criteria required for a network graph to be robust (Section 6.2), is not satisfied.

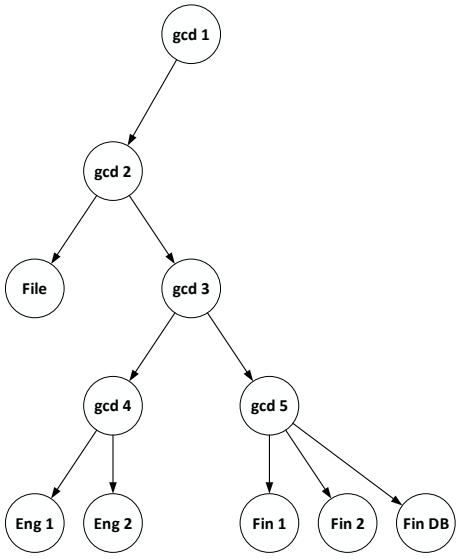
Suppose Condition 1 is not satisfied. Then there exists at least one path in G from the root to the parent of a resource, such that the SDD strategy is not satisfied. In the RNS algorithm, the root is added first to an empty graph G. After which nodes both resource and *gcd* nodes are added in non-increasing order of their weights to G by first attaching them to the root only in the ADD-NODESET-TO-G procedure. In ADD-NODESET-TO-G, when a *gcd* node is added to the root r, *cset* contains all its child nodes. The policy at *gcd* is equal to the GCD of policies of all nodes in *cset*. Furthermore, while adding child nodes from *cset* to *gcd*, we check if any child node's weight is equal to $gcd.weight$, in which case we delete the child node (if it is not a resource) and attach all of its children to *gcd*. Therefore, in all the cases the SDD strategy is satisfied while adding resources and *gcd*



(a) Net. graph after 1st iteration of ADD-NODESET-TO-G



(b) Net. graph after 2nd iteration of ADD-NODESET-TO-G



(c) Net. graph after 3rd iteration of ADD-NODESET-TO-G

Figure 6.6: Progression of RNS for the illustrative example

nodes. Therefore, the SDD strategy is satisfied in all paths from the root to the node of the parent of a resource, which is a contraction.

Suppose Condition 2 is not satisfied. Then, the condition stated under Definition 6.2.1 is not satisfied; that is, there exists a segment S, such that a resource $r \in S$ and a resource

$r' \notin S$ have commonalities with more weight than the weight of the commonalities between elements of the segment S.

Without loss of generality, let us assume that $w(p(r)) \leq w(p(r'))$. Nodes are added to G in non-increasing order of their weights. Therefore, when r is added to G, all the existing nodes in G have a weight greater than or equal to $r.weight$. Since $r.weight \leq r'.weight$, G must include r'. Furthermore, while evaluating clusters of resources (having a weight greater than or equal to $r.weight$) including r, we choose a cluster having the maximum weight (and possibly the maximum size) among all such clusters evaluated, as the segment S. Therefore, it must be the case that a cluster having r and r' is evaluated. Since r and r' have commonalities with higher weight, then the chosen cluster forming a segment must contain r and r'. Therefore, the segment S must contain both r, r', which is a contradiction.

Suppose Condition 3 is not satisfied; that is, superfluous firewall chaining exists in G. Then there exists at least one firewall/*gcd* node with a single node that is a firewall attached. Recall that *gcd* nodes are added to G only when more than one node can be attached to them. Therefore, by the end of the RNS algorithm, G has no superfluous firewalls, which is a contradiction. □

Theorem 6.3.3. *Let $R = \{r_1, r_2, \ldots, r_n\}$ be the input of size n to the RNS Algorithm. Then, the running time of RNS is $O(n^2)$.*

Proof. The proof can also be found in [MAK21]. Using numeral key values between $1-m$ as labels for distinct policies in R, and Bucket sort, the time required to compute the subsets $S_1, S_2, \ldots, S_m$, where $m \leq n$ is $O(n)$.

Since *gcd* nodes (either permanent or temporary) are created for each set $s \in S_1, S_2, \ldots, S_m$ exactly once, the number of *gcd* nodes in the network graph is at most n. Analogously, the number of temporary nodes created during the execution of the RNS algorithm is also at most n. Hence the total time required for the the RNS algorithm is $O(n)$, plus the total time required to execute the ADD-NODESET-TO-G procedure.

Let $t \leq n$ be the number of distinct weights of policies in the resource set R. Then, the ADD-NODESET-TO-G procedure is executed t times, and the total number of nodes processed by it is at most $2n$ (since $|F| \leq 2n$). Hence the time taken by ADD-NODESET-TO-G is $2n$ plus the time taken by the CURR-MAX-WEIGHT-SET for each node s in F where $s.weight > r.weight$.

The total time required by the CURR-MAX-WEIGHT-SET procedure is equal to the number of child nodes of r the procedure evaluates to construct the $smax$ set for all nodes in F. The maximum number of such nodes is at most $2n$ (total number of nodes in G) at each execution of the CURR-MAX-WEIGHT-SET procedure. Further, the CURR-MAX-WEIGHT-SET procedure is executed at most $2n$ times (number of nodes in F). Hence, the total time required by the CURR-MAX-WEIGHT-SET procedure is at most $O(n^2)$.

Therefore, the total running time for the RNS Algorithm is $O(n^2)$. □

6.3.3 Discussion

We proposed two algorithms, Exp-RNS and RNS, for building a robust network topology. The Exp-RNS algorithm has an exponential running time, and RNS has a polynomial running time. Despite its exponential complexity, we presented the Exp-RNS algorithm for its simplicity in demonstrating the usage of the formalism in building a robust network. We later introduced RNS, which can easily meet the needs of most real-time network configuration in dynamic networks.

The restriction in the superfluous firewall chaining concept mentioned in Section 6.2 can be further tightened, where a firewall having no resources attached to it (irrespective of the number of firewalls attached to it) is removed from the network. This would not only simplify the network but also significantly economize it by minimizing the number of firewalls required to produce a secure and robust network. The Exp-RNS and RNS algorithms can be easily modified with minor changes to adapt to the tighter version of firewall chaining. Both these versions have their advantages and drawbacks. The superfluous firewall

chaining mentioned in Section 6.2 would result in a network having more firewalls and a greater number of layers protecting the resources, however building such a network could be expensive. The tighter version results in a network having fewer firewalls and fewer layers protecting the resources; however, it is economical.

Our approach is for designing and building networks and is not focused on improving the security in existing networks. Quantified attack graphs ([WNJ06, ILP06, DPRW07, AJN12, PDR12, MATWYO16]), and similar formalisms (e.g., [RAS13]) have been proposed to harden the security of existing networks. However, our approach can be used to assess the security robustness of an existing network in the following ways. One way is to use the GCD to calculate the best policy for each of the segments of the existing networks. Then, we compare the calculated firewall policy for each segment to the existing policy. This would reveal any missing rules or any flaws in the policy of the existing firewall. The second way is to verify whether the existing network implements DD strategy. A third way is to use RNS algorithm to get the appropriate robust network for the resources that are in the existing network. Then, we compare the topologies (existing and calculated) with regard to their effectiveness in protecting the resources. This comparison might lead the network security architect to better tune the existing network for enhanced security.

Today's networks are very large in terms of the number of their nodes. The abstraction adopted in the family approach (discussed in Chapter 4) allows collapsing a subset of nodes into one node that is protected by a family of policies obtained from the policies of the abstracted nodes. This approach for example allows us to focus on the design of a network in a geographic area while considering the rest of the network in another geographic area as a node running a family of policies. It is direct usage of the principle of divide and conquer.

It is also important to put our results in perspective with Network Address Translation (NAT) which is a translation mechanism executed by a firewall that is in contact with the Internet outside of a private network. It assigns public addresses to a computer inside a network. Its purpose is to limit the number of public addresses that internal nodes within a

network need to know about external resources. In our work, we assume that only the node that we call root will have this translation capability. Our purpose is to design local networks and NAT can be placed on the boundary between the local network to be designed and the external Internet (i.e., on the entry points (roots) of the network). Hence, our policies use only local addresses to the designed network.

We used the RNS algorithm on a network with 100 resources. On an *Apple MacBook Pro with a CPU 2.7 GHz Intel Core i5 and a memory of 8 GB 1867 MHz DDR3*, the algorithm toke 24.40 seconds to calculate the network topology and the policies to be enforced at its firewalls. Then, we inactivated 10 resources among the previous 100 and run the RNS algorithm to recalculate the new network topology and the policies of its firewalls. The machine took 24.36 seconds. While the above performance numbers do not constitute, by any stretch of empirical assessment, a definitive real performance indicator, they give an idea about the range of time needed. We should keep in mind that after changing the network, we simply recalculate everything. One can imagine several strategies that use the characteristics of the resources that are added or removed. For instance, if the change affects only a subnetwork, we can recalculate only the new topology of that subnetwork that involves a small number of resources. In this situation, we will see an enhanced performance that appropriate for real-time network reconfiguration. In many situations, the performance of the RNS algorithm when used to recalculate the whole topology of the network at each network change is acceptable to reconfigure the network in a reaction to several security threats. However, an empirical study to further assess this aspect is needed. Moreover, we also need further research related to the usage of RNS algorithm in many networks such as mesh networks.

The algorithm presented in this chapter is about generating a network structure from scratch. This leaves room for discussing approaches for an already established network. For instance, an approach that validates the segmentation of a given network and points for potential threats, and gives a suggested new secure structure with minimum changes such

that the structure is not disrupted so much.

6.4 Summary

In this chapter, we have presented a formal definition for segmentation based on security policies. Based on the segmentation formalism we have defined a robust network that is concerned with building the topology of the network and the placement of resources and security mechanisms. We have also proposed an approach that goes beyond building segments to place these segments within the topology to build a robust and secure network topological structure.

Chapter 7

Network Segmentation for Multiple Entry Networks

Current computer networks have several entry points. This chapter extends the formalism for network segmentation introduced in the previous chapter to networks with multiple entry points. Section 7.1 introduces networks with multiple entry points. Section 7.2 presents an illustrative example used to explain the algorithms proposed in this chapter. Section 7.3 proposes two algorithms for implementing the segmentation in a network with multiple entry points. Finally, Section 7.4 presents a summary of the chapter.

7.1 Multiple Entry Networks

Computer networks are experiencing rapid growth in recent years and expected further growth in the future. The number of the resources they encompass is reaching considerable levels which could span over multiple geographical locations. The diversity in the access control policies governing these resources has become challenging for security officers to correctly articulate and maintain, especially for dynamic networks. Moreover, these networks usually have several entry points, which increase the network attack surface. Each

entry point is usually intended to give access to a subnetwork. This brings a unique set of challenges, both while designing and maintaining the network. One of these challenges is segmenting networks to achieve a robust and secure structure such that all resources are properly protected independently from which entry point the traffic has originated.

The decision of having a single entry point to a network or several entry points depends on many aspects. For instance, small networks that use Virtual Private Network (VPN)s typically have a single entry point [WWF13]. On the other hand, many large organizations have networks with site-to-site connections, requiring networks with multiple entry points configuration.

In Chapter 2, we have presented the concept of network slicing and its relation to the segmentation of networks with multiple entry points.

The segmentation algorithm proposed in Chapter 6 builds a network with a single entry point. This chapter extends that algorithm to construct an optimal network with multiple entry points. We consider the example given below to explain the algorithms proposed for building networks with multiple entry points.

7.2 Illustrative Example of Multiple Entry Networks

We consider an organization's network resources. The resources belong to two branches: A and B. Branch A consists of a *Web server*, an *Email server*, two *File servers*, two Human Resources (HR) workstations, and two finance workstations. The *Web server* and *Email server* allow access for HTTP and SMTP protocols from the internet. Moreover, they allow access from all internal resources within the branch and block everything else. The *File servers* are intended to be accessed by only internal resources within branch A. The HR workstations allow access to each other only, and the same applies for the finance workstations. Branch B consists of the same resources and requirements except that it does not have a *File server*. Moreover, the organization has two HR servers that are accessed

only by HR workstations in both branches. It has also two finance servers that allow access to finance workstations only. Figures 7.1, 7.2, 7.3, 7.4, 7.5, 7.6, 7.7, 7.8, and 7.9 show the policy of these resources in *iptables* language.

```
1 -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
2 -A INPUT -m state --state INVALID -j DROP
3 -A INPUT -p tcp -m state --state NEW -m tcp --dport 80 -j ACCEPT
4 -A INPUT -p tcp -m state --state NEW -m tcp --dport 25 -j ACCEPT
5 -A INPUT -s 10.0.1.0/24 -j ACCEPT
6 -A INPUT -s 10.0.2.0/24 -j ACCEPT
7 -A INPUT -s 10.0.3.0/24 -j ACCEPT
8 -A INPUT -s 10.0.4.0/24 -j ACCEPT
9 -A INPUT -j DROP
```

Figure 7.1: Branch A Web and Email servers policy

```
1 -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
2 -A INPUT -m state --state INVALID -j DROP
3 -A INPUT -s 10.0.2.0/24 -j ACCEPT
4 -A INPUT -s 10.0.3.0/24 -j ACCEPT
5 -A INPUT -s 10.0.4.0/24 -j ACCEPT
6 -A INPUT -j DROP
```

Figure 7.2: Branch A File server policy

```
1 -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
2 -A INPUT -m state --state INVALID -j DROP
3 -A INPUT -s 10.0.3.0/24 -j ACCEPT
4 -A INPUT -j DROP
```

Figure 7.3: Branch A Finance workstations policy

The requirement is to structure the network of the organization and to have an entry point for each of its branches A and B. The desired structure is shown in Figure 7.10.

```
1 -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
2 -A INPUT -m state --state INVALID -j DROP
3 -A INPUT -s 10.0.4.0/24 -j ACCEPT
4 -A INPUT -j DROP
```

Figure 7.4: Branch A HR workstations policy

```
1 -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
2 -A INPUT -m state --state INVALID -j DROP
3 -A INPUT -p tcp -m state --state NEW -m tcp --dport 80 -j ACCEPT
4 -A INPUT -p tcp -m state --state NEW -m tcp --dport 25 -j ACCEPT
5 -A INPUT -s 10.0.5.0/24 -j ACCEPT
6 -A INPUT -s 10.0.6.0/24 -j ACCEPT
7 -A INPUT -s 10.0.7.0/24 -j ACCEPT
8 -A INPUT -j DROP
```

Figure 7.5: Branch B Web and Email servers policy

```
1 -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
2 -A INPUT -m state --state INVALID -j DROP
3 -A INPUT -s 10.0.6.0/24 -j ACCEPT
4 -A INPUT -j DROP
```

Figure 7.6: Branch B Finance workstations policy

```
1 -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
2 -A INPUT -m state --state INVALID -j DROP
3 -A INPUT -s 10.0.7.0/24 -j ACCEPT
4 -A INPUT -j DROP
```

Figure 7.7: Branch B HR workstations policy

7.3 Network Segmentation Algorithms for Multiple Entry Networks

In this section, we present two solutions which take $n > 1$ sets of resources $Q_1, Q_2, \ldots, Q_n$ as input, and generate a secure and robust network graph with $e_1, e_2, \ldots, e_n$ entry points, such

```
1 -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
2 -A INPUT -m state --state INVALID -j DROP
3 -A INPUT -s 10.0.3.0/24 -j ACCEPT
4 -A INPUT -s 10.0.6.0/24 -j ACCEPT
5 -A INPUT -s 10.0.8.0/24 -j ACCEPT
6 -A INPUT -j DROP
```

Figure 7.8: Finance server policy

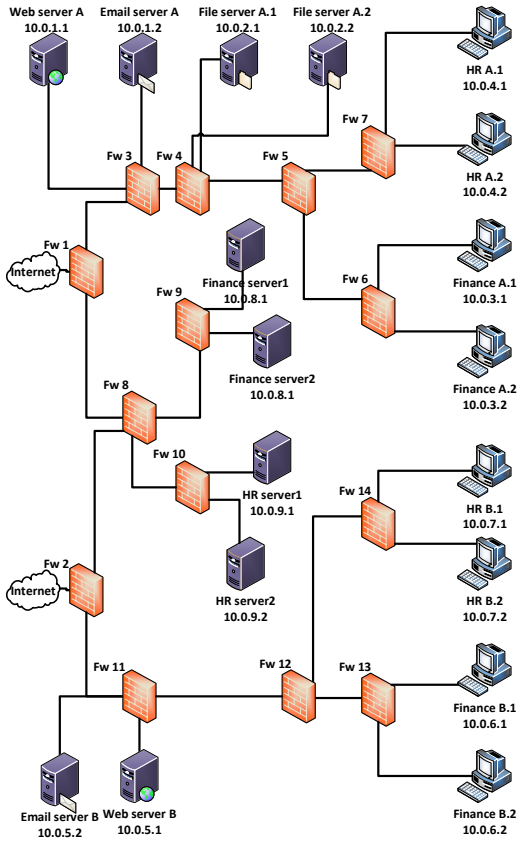
```
1 -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
2 -A INPUT -m state --state INVALID -j DROP
3 -A INPUT -s 10.0.4.0/24 -j ACCEPT
4 -A INPUT -s 10.0.7.0/24 -j ACCEPT
5 -A INPUT -s 10.0.8.0/24 -j ACCEPT
6 -A INPUT -j DROP
```

Figure 7.9: HR server policy

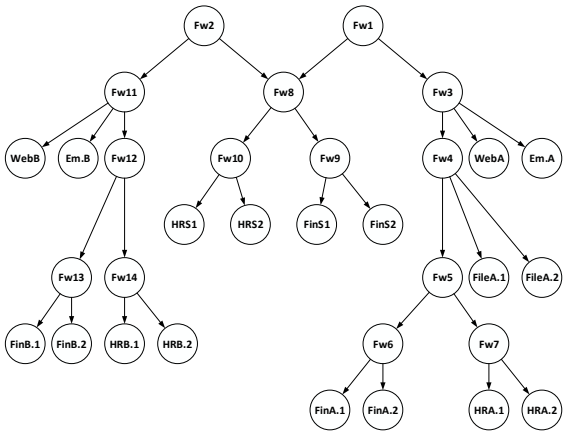
that the i-th entry point e_i, protects the i-set of resources Q_i, $1 \leq i \leq n$. While designing networks with multiple entry points, the intersection of the sets of the resources can be dealt with in two ways as presented in *Solution-1* and *Solution-2*. *Solution-1* integrates the intersections as the network graph G is generated at each step. On the other hand, *Solution-2* separates the intersected sets, such that each of the non-empty set of intersections forms a segment of its own. Hence, *Solution-2* constructs separate slices for the shared resources. For example, if we have a network with three entry points it creates separate slices for common resources accessed from entries points (e_1, e_2), (e_1, e_3), (e_2, e_3) and (e_1, e_2, e_3). Later, in Section 7.3.3, we compare the two solutions and discuss the advantages and drawbacks of each.

7.3.1 Solution-1: Non-slicing of Shared Resources (NSR) Algorithm

The Non-slicing of Shared Resources (NSR) algorithm is a simple algorithm that builds a network with multiple entry points. It takes resource sets $Q_1, Q_2, \ldots, Q_n$, where $n > 1$ as input to build a network graph with entry points $e_1, e_2, \ldots, e_n$, respectively. The outline of



(a) A topological representation of the network



(b) A graph representation of the network

Figure 7.10: The desired network topology and the output of the SSR algorithm

the algorithm is as follows:

- On input $Q_1, Q_2, \ldots, Q_n$, the NSR algorithm first generates a network graph for resources in Q_1 using the RNS algorithm (Algorithm 2).
- Then it loops through resource sets $Q_2, \ldots, Q_n$ one at a time. At the i-th iteration, the algorithm generates the network graph for resources in Q_i using the RNS algorithm and merges (adds V_i, E_i, r_i to G) it with the graph G constructed so far.

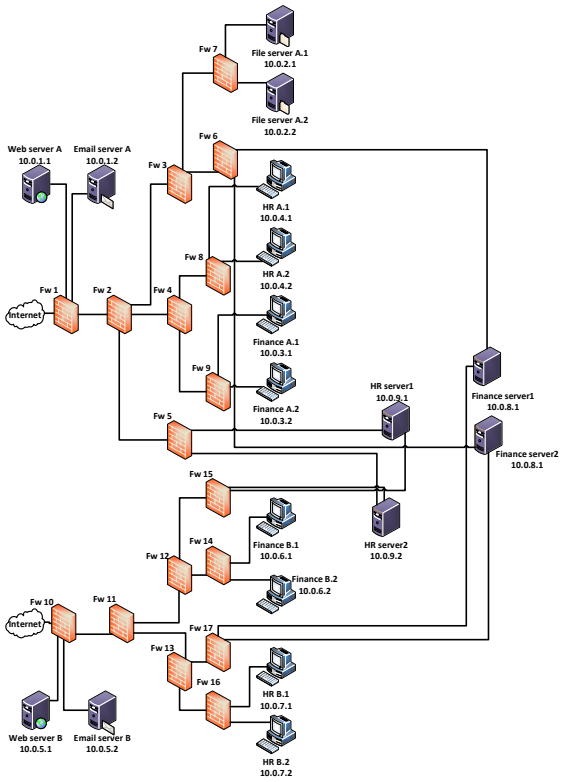
Algorithm 3 Non-slicing of Shared Resources (NSR) Algorithm

function SIMPLE_MULTI_ENTRY_POINTS($Q_1, Q_2, \ldots, Q_n$) ▷ $Q_1, Q_2, \ldots, Q_n \neq \emptyset$
  $G = \text{NULL}$ ▷ $G = (V, E, I)$
  $(G_1, r_1) = \text{RNS}(Q_1)$
  $V = V \cup V_1; E = E \cup E_1; I = I \cup \{r_1\}$
  $Q = Q_1$
  for $i = 2$ to n **do**
    $(G_i, r_i) = \text{RNS}(Q_i)$
    $V = V \cup V_i; E = E \cup E_i; I = I \cup \{r_i\}$
    $Q = Q \cup Q_i$
  end for
  return G
end function

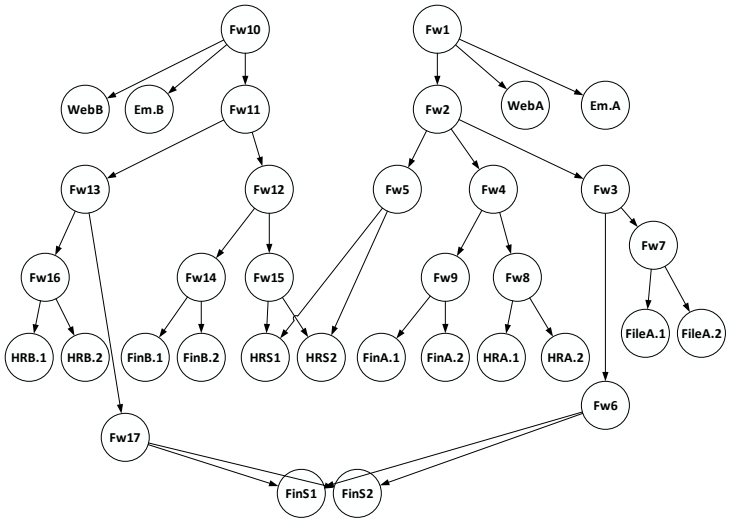
Figure 7.11 presents the topology of the network generated after implementing the NSR algorithm on the resources of the illustrative example.

Theorem 7.3.1. *The NSR algorithm constructs a robust network graph G, with entry points $e_1, e_2, \ldots, e_n$ protecting resources in the sets $Q_1, Q_2, \ldots, Q_n$, respectively.*

Proof. The graphs $G_1, G_2, \ldots, G_n$ are constructed using the RNS algorithm, and by Theorem 6.3.2, they all are robust. Merging the subgraph G_i, $2 \leq i \leq n$ with G, still preserves the robustness of the network as no new vertices are added to G (as $V_i \subset V$). Furthermore, since G_i is robust, the edges in E_i preserve this robustness when added to G. Therefore, the resulting graph G, obtained after execution of the NSR algorithm, is a robust network graph. □



(a) A topological representation of the network



(b) A graph representation of the network

Figure 7.11: The network topology generated after implementing the NSR algorithm

The NSR algorithm is a simple algorithm to build a robust network graph with multiple entry points. The NSR algorithm computes the network graph in time linear to the sum of sizes of the robust networks generated for the input sets $Q_1, Q_2, \ldots, Q_n$.

7.3.2 Solution-2: Slicing of Shared Resources (SSR) Algorithm

The Slicing of Shared Resources (SSR) algorithm builds a robust network graph. Given the sets of resources $Q_1, Q_2, \ldots, Q_n$, the SSR algorithm builds a robust network with entry points $e_1, e_2, \ldots, e_n$, where e_i is the entry point for Q_i. The network graph generated by the SSR algorithm does not have any overlapping subnetworks; that is, the set of resources in all subnetworks are disjoint. The outline of the algorithm is as follows:

- Given an input of sets of resources $Q_1, Q_2, \ldots, Q_n = Set$, the SSR algorithm first adds all the entry points $e_1, e_2, \ldots, e_n$ protecting resources $Q_1, Q_2, \ldots, Q_n$ to G, using the ADD_ENTRY_POINTS function.

- For each set Q_i, $1 \leq i \leq n$, the ADD_ENTRY_POINTS function creates an entry point e_i, and sets its policy equal to the `gcd` of the resource policies in Q_i. The other gcd object parameters for e_i are set accordingly and can be seen in the ADD_ENTRY_POINTS function.

- The SSR algorithm then generates the set $Inter\_Set$, which is a collection of disjoint sets, obtained from the powerset of $Input\_Set$ ($\mathcal{P}(Input\_Set)$) as follows:

  - Let $\mathcal{P}_d$ be the list consisting of all the sets in the powerset $\mathcal{P}(Input\_Set)$, in non-increasing order of their sizes, such that the sets with same cardinalities are placed together.

  - For each set $s \in \mathcal{P}_d$, starting from the largest set, remove each resource r, contained in the sets of resources in s, from all sets in $\mathcal{P}_d - s$; that is, for all $s' \in \mathcal{P}_d - s$, $s' = s' - s$.

  - For each $s \in \mathcal{P}_d$, where $s = \{Q_{i1}, Q_{i2}, \ldots, Q_{ik}\}$, $1 \leq k \leq n$, $Q_{i1} \cap Q_{i2} \cap \ldots \cap Q_{ik}$ is added to $Inter\_Sets$ if and only if $Q_{i1} \cap Q_{i2} \cap \ldots \cap Q_{ik} \neq \emptyset$. This step is to identify the resources that are going to form slices of shared resources.

- For each $s \in Inter\_Sets$, the SSR algorithm computes the robust network graph G_s for s using the RNS algorithm and add it to G using the CONCATENATE function.

- The CONCATENATE function takes the graphs G and G_s as inputs. It loops through every entry point e_i, $1 \leq i \leq n$, and adds and edge from e_i to r_s, only if, the policy of r_s refines the policy of e_i and $r_s.set \subset e_i.set$. If the policies of r_s and e_i are equal and $r_s.set \subseteq e_i.set$, then it deletes the root r_s, and all its outgoing edges and attaches its children to e_i.

Algorithm 4 Slicing of Shared Resources (SSR) Algorithm

```
function MULTIPLE_ENTRY_POINTS(Q_1, Q_2, ..., Q_n)        ▷ Q_1, Q_2, ..., Q_n ≠ ∅
    Input_Set = {Q_1, Q_2, ..., Q_n}
    G = NULL                ▷ G = (V, E, I)
    G = ADD_ENTRY_POINTS(G, Input_Set)
    Inter_sets = Disjoint intersection sets without empty sets
    for each s ∈ Inter_set do
        (G_s, r_s) = RNS(s)
        G = CONCATENATE(G, G_s)
    end for
    return G
end function
```

Figure 7.10 presents the topology of the network generated after implementing the SSR algorithm on the set of resources of the illustrative example.

```
function ADD_ENTRY_POINTS(G, Set)
    for each Q_i ∈ Set do
        Create node e_i
        e_i.p ← GCD(Q_i)            ▷ GCD of all resources in Q_i
        e.weight ← weight of e.p
```

$e_i.set \leftarrow Q_i$;

$e.size \leftarrow |e.set|$; $e.is\_resource \leftarrow false$

$V \leftarrow (V \cup e_i)$; $I \leftarrow (I \cup e_i)$

end for

return G

end function

function CONCATENATE(G, G_s)

for each $e_i \in I$ **do**

if $r_s.p \sqsubset e_i.p \wedge r_s.set \subset e_i.set$ **then**

$E \leftarrow E \cup \{(e_i, r_i)\}$

end if

if $r_s.p = e_i.p \wedge r_s.set \subseteq e_i.set$ **then**

for each child c of r_i **do**

$E \leftarrow E - \{(r_i, c)\}$

$E \leftarrow E \cup \{(e_i, c)\}$

$c.\pi = c.\pi - \{r_s\} \cup \{e_i\}$

end for

$V = V - \{r_s\}$

end if

end for

end function

Since the SSR algorithm computes the powerset of the *Input_Set*, it is an exponential algorithm. Therefore, it requires more time to execute than the NSR algorithm. However, it creates a network design with slices of the shared resources, such that the set of resources in the slices are disjoint; thus, resulting in a smaller attack surface that is limited to the concerned segment.

Theorem 7.3.2. *The SSR algorithm constructs a robust network graph G, with entry points $e_1, e_2, \ldots, e_n$, protecting resource sets $Q_1, Q_2, \ldots, Q_n$ respectively.*

Proof. The SSR algorithm, creates all the entry points $e_1, e_2, \ldots, e_n$ and adds it to G. At this point G consists of n disconnected vertices (entry points) and is robust. Then, the algorithm uses the RNS algorithm to build a subnetwork graph for each of the disjoint sets in $Inter\_Set$. By Theorem 6.3.2, all these sub-graphs are robust network graphs. Further, when a subnetwork G_i, where $1 \leq i < 2^n$, is added to G using the CONCATENATE function, an edge is added between r_i and an entry point e_j, where $1 \leq i \leq n$, only if the policy at e_j is refined by the policy at r_i and $r_s.set \subset e_i.set$. If both the policies at e_j and r_i are the same and $r_s.set \subseteq e_i.set$, then the root r_i is removed from G along with its outgoing edges, and all its children are attached to e_j. Thus, in both these cases the policy at each child node of e_i refines the policy at e_i. Therefore, the resulting graph G, obtained after execution of the SSR algorithm, is a robust network graph. □

7.3.3 Discussion

We presented the NSR and SSR algorithms to design a network with multiple entry points. Each of the algorithms presents a design solution. In the following, we review the strengths and the weaknesses of the output of each of the two algorithms. The NSR algorithm gives networks that provide deeper layered protection to resources. We can notice that in Figure 7.11, which gives the topology of the network obtained from the illustrative example using the NSR algorithm, the resource *Finance server1 (10.0.8.1)* is protected by four firewalls on each path from any of the two entry points. It is heavily protected by four layers of access control protection. While in the topology given by the SSR algorithm and illustrated by Figure 7.10, the same resource (i.e., *Finance server1*) is protected by only three firewalls. We notice that the enhanced security offered by the topology given by the NSR algorithm induces a high implementation cost (more firewalls). Moreover, the

network generated by the NSR algorithm, has edges going from a firewall in a subnetwork, to resources in other subnetworks. For instance, in Figure 7.11, firewall *Fw17* which is a part of the subnetwork of Branch B is related to the resource *Finance server1*, which belongs to the subnetwork of Branch A. This might lead to increasing the attack surface for an internal intruder; through *Finance server1* it can reach the two networks. On the other hand, in the topology generated by the SSR algorithm and illustrated by Figure 7.10, the resource *Finance server1* is isolated in a subnetwork under one firewall *Fw9*, which reduces any contamination for the rest of the network, should an internal attack be mounted starting from this resource.

Recall that the run time complexity of the NSR algorithm is linear in the sum of sizes of the robust networks created for the input set of resources $Q_1, Q_2, \ldots, Q_n$. On the other hand, the runtime complexity of the SSR algorithm is exponential (caused by constructing $Inter\_Set$ set).

The weight function captures the implied security requirements embodied in the policies. This weight function can be amended to capture for instance the traffic volume induced by each rule. This would require measuring the traffic and tracing it back to the rules that allowed it. Then, one can perform a slicing based on the criterion of traffic volume by capturing in the weight function the information about the effect of the rules of the policies on the traffic between nodes. Hence, the algorithm that we propose in this chapter can be extended to be used in many forms of network slicing.

Conventional firewalls rely on enforcing traffic filtering at the entry points, and on that each machine within the network is to be trusted [Bel99]. However, due to the latter assumption securing the entire network is hard in practice. Firewalls at the entry point do not protect the network from internal attacks. The RNS algorithm, and consequently the NSR and SSR algorithms presented in Section 7.1, are based on the assumption that all nodes are not to be trusted. Internal traffic has to satisfy the policies to be allowed access to resources. If a node wants to communicate with another node, due to DD strategy, it has to go through

internal firewalls to access the destination.

It has been documented (e.g., [KKE+20]) that it is trivial for anyone to establish a new clandestine unauthorized entry point (or rogue access point) to a network. Let G be a network with two entry points e_1 and e_2 that is built using either the NSR algorithm or the SSR algorithm. Let e_3 be a clandestine entry point connected to an internal node. Only packets arriving from e_3 satisfying the policies will reach their corresponding resources. In other words, clandestine entry points can be created, but their packets can access the resources only if they abide by the policies. To fully prevent clandestine entry points, one needs to have a mechanism to trace the paths of a packet from an entry point to the resources.

7.4 Summary

In this chapter, we have proposed an extension of the segmentation algorithm to networks with multiple entry points. The NSR algorithm integrates shared resources while the SSR algorithm separate shared resources.

Chapter 8

SDN Segmentation

Software Defined Networks (SDN) is a networking paradigm that separates the control plane from the forwarding plane. This chapter implements the RNS algorithm in SDN environment. The implementation is done through the introduction of an additional plane in charge of the configuration and governance of SDN data planes that we call Dynamic Configuration and Governance (DCG) plane. It is intended to give agility to dynamic networks. Section 8.1 gives an overview of the work presented in this chapter. Section 8.2 presents the implementation details of the RNS module located in the DCG plane. Section 8.3 presents the implementation of the three proposed architectures. Section 8.4 presents the testbed and selected topology for the experimentation. Section 8.5 shows and discusses the assessment and extermination results. Finally, Section 8.7 summarizes the chapter.

8.1 Overview

We have introduced the concept of SDN and related research in Chapter 2. There is little research on structuring the SDN data plane for security as discussed in Chapter 2. The RNS algorithm can be used in SDN environment to create the data plane topology and to assign policies to be enforced by the switches. It could also be used to segment large networks into

multiple subnetworks with multiple SDN controllers. The RNS algorithm and its interfaces to the data plane and the control plane form a module for dynamic configuration and governance of the network. We call this module DCG plane as shown in Figure 8.1.

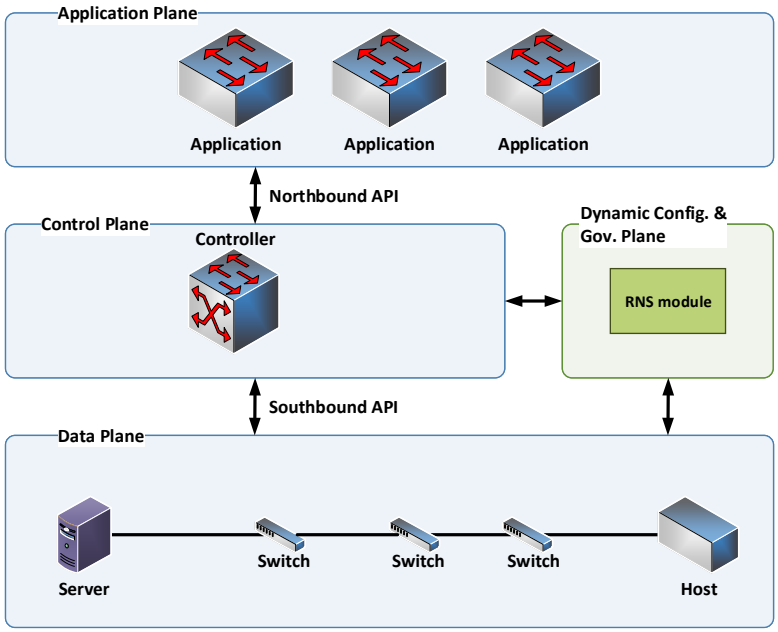


Figure 8.1: SDN architecture and the module running the RNS algorithm

In this chapter, we aim at assessing the efficiency of the use of DCG plane that is supported by the RNS algorithm in SDN. For this purpose, we build three SDN architectures using *mininet* [dOSSP14] that conform to the topology calculated by the RNS algorithm. Since in SDN environment the firewalls can be placed either at the control plane or at the data plan (see the discussion in Chapter 2), we conceive the following three possible implementations for the firewalls calculated by the RNS algorithm:

1. **Architecture 1**: A single and centralized stateful firewall located at the control plane. The firewall governs all the data plane switches.

2. **Architecture 2**: Multiple distributed stateful firewalls located at the control plane.

Each of the firewalls is assigned to a unique switch at the data plane. Hence, we have as many firewalls as switches.

3. **Architecture 3**: Multiple distributed stateful firewalls located at the data plane. Each switch is transformed into a stateful firewall. This is made possible by the usage of the data plane abstraction, *BEBA software switch* [SPB+20], which is an extension of OpenState [BBCC14].

We assess the usage of the RNS algorithm, as an essential component of the DCG plane, within the three above architectures. We aim at determining the most appropriate architecture to use with the RNS algorithm for SDN environment. We look at the performance of the above architectures. We consider the following performance attributes:

1. Setup cost: It is evaluated by the number of exchanging packets between control and data planes during the setup phase. We used *Wireshark* [Com20] to count the number of exchanged packets.

2. Reachability: To test the effectiveness of the enforced policies, we use *ping* [Muu83] utility such that each host tries to reach every other host in the network.

3. Response time or latency: We use *ping* [Muu83] utility to find out the time needed for the communication between two selected nodes.

4. Bandwidth: We assess this performance parameter using *iPerf* [DEF+20] *TCP test* to obtain the bandwidth for the link under consideration.

5. Latency variation(jitter): We use *iPerf* `UDP` test to get the jitter for the link under consideration.

6. Resilience to topology change: It is measured by the number of packets exchanged to fulfill an intended topology change.

The contribution of the chapter is twofold. First, it illustrates the use of the RNS algorithm as an essential component of the DCG plane in SDN. It shows how the algorithm is essential for a dynamic SDN. We give the details on the use of the RNS algorithm in SDN for the implementation of the three architectures under consideration. Second, we assess these architectures to capture their drawbacks and strengths. This helps us identify the most appropriate architecture for using RNS algorithm in SDN. We find out that when the network is very dynamic, Architecture 2 is the most appropriate. For a relatively stable network, Architecture 3 is the most appropriate. Although it has the highest cost in the setup and update phases, in the operation phase it does not exchange any packets between the control and the data planes.

8.2 RNS Implementation

For this study, the RNS algorithm has been implemented as a module located in the DCG plane. In the context of SDN, the RNS module generates the data plane topology and structure. It determines how many switches are needed, where they should be placed, and the links between resources and switches. Moreover, it generates policies to be enforced at each switch. In Architecture 1, a single firewall with a single policy is used, as shown in Figure 8.4 and will be explained later in detail. The RNS application generates this single policy by combing all the generated policies for the switches, it also adds an attribute to each rule to identify the switch this rule enforced at.

The RNS module at the DCG plane consists of six classes as shown in Figure 8.2. A main class, a class for storing and manipulating policies, a class for storing and calculating GCDs of the families of policies, and a class for storing and generating network graph that uses a node class. Moreover, it has a utility class that encompasses useful methods used by the classes of the RNS module. In the assessment work that we carried for this work, we provide to the RNS module a list of resource names along with their policy files. In the

policy file, the first line indicates the IP address of the resource. The rest gives the policy by sequentially stating the rules. The grammar of the rules is given in Figure 8.3.

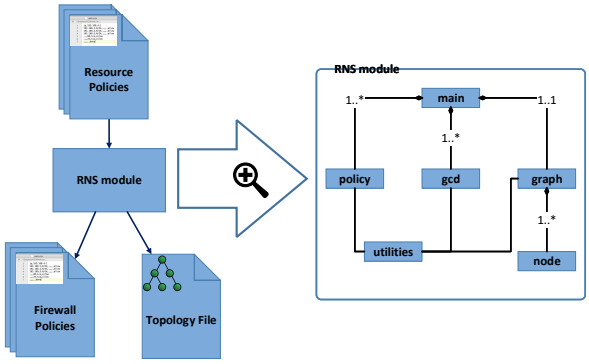


Figure 8.2: Structure of the RNS module and its input and output

⟨rule⟩ ⟶ [⟨source_ip⟩], [⟨source_port⟩], [⟨destination_ip⟩],
[⟨destination_port⟩], [⟨protocol⟩], ⟨action⟩
⟨source_ip⟩ ⟶ ⟨ip_number⟩ | ⟨sub_network⟩
⟨source_port⟩ ⟶ ⟨port_number⟩ | ⟨port_range⟩
⟨destination_ip⟩ ⟶ ⟨ip_number⟩ | ⟨sub_network⟩
⟨destination_port⟩ ⟶ ⟨port_number⟩ | ⟨port_range⟩
⟨protocol⟩ ⟶ `TCP` | `UDP` | `ICMP` | *all*
⟨action⟩ ⟶ *allow* | *deny*
⟨sub_network⟩ ⟶ ⟨ip_number⟩/⟨dgits⟩
⟨ip_number⟩ ⟶ ⟨8bit_digit⟩.⟨8bit_digit⟩.⟨8bit_digit⟩.⟨8bit_digit⟩
⟨8bit_digit⟩ ⟶ 0 – 255
⟨port_range⟩ ⟶ ⟨port_number⟩ : ⟨port_number⟩
⟨port_number⟩ ⟶ ⟨digits⟩
⟨digits⟩ ⟶ ⟨digit⟩ | ⟨digit⟩⟨digits⟩
⟨digit⟩ ⟶ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Figure 8.3: Firewall rule grammar

The non-terminals ⟨source_ip⟩ and ⟨destination_ip⟩ are optional indicating the source IP and destination IP, respectively. They can be a specific resource IP address, a network, or a sub-network. When these non-terminals are not given, it indicates the IP that covers all IP domain (i.e., 0.0.0.0/0). The non-terminals ⟨source_port⟩ and ⟨destination_port⟩ are optional indicating the source and destination ports, respectively. They can be a specific port or a range of ports. When not provided, it indicates the whole range of possible ports

(i.e., 0 to 65535). The non-terminal ⟨protocol⟩ is optional and indicates the communication protocol. The non-terminal can be a specific protocol (i.e., `TCP`, `UDP`, or `ICMP`) or *"all"* which indicates all protocols. When no value is given for this field, it indicates the *"all"* value. The non-terminal ⟨action⟩ is the only mandatory one that indicates the action to be taken by the switch. The possible values in our simulation are *allow* or *deny* which instruct the switch to forward the packet to its destination or drop the packet, respectively.

The RNS module at the DCG plane creates a policy object for each resource. The policy object reads a policy file and stores the policy. It transforms the sequential rules into disjoint rules such that executing them in any order produces a consistent policy. It does the transformation as it reads rules one by one from the policy file. If a new rule has a domain that intersects with that of an already existing rule, the intersected part is removed from the new rule as it already exists in another rule. The policy object uses a weight function to calculate and stores the weight of the policy.

The RNS module proceeds to implements the RNS algorithm as explained in detail in Chapter 6. In the case of a policy or topology change, the DCG plane gets notified and re-executes the RNS module dynamically to generate the updated topology and firewall policies. The updates are then carried to the data plane topology and firewall policies are updated.

8.3 Implementation of the Architectures

In this section, we present the implementation of the three architectures introduced in Section 8.1. They implement stateful firewalls, which keep track of connections state. Therefore, each firewall needs to follow the progress of a session by recording its state attributes and values. The firewall changes the state of the connection upon receiving a packet. The architectures presented in this chapter record sessions' attribute values using state tables. In all architectures, each firewall has its state table. Therefore, Architecture 1 has a single

state table as shown in Figure 8.4, Architecture 2 has a state table for each firewall as shown in Figures 8.5. Architecture 3 has a state table at each firewall for each protocol as shown in Figure 8.6.

In all the architectures that we are considering, the topology that is generated by the DCG plane is used to create the data plane architecture as shown in the Figures 8.4, 8.5, and 8.6. It is the same topology that we will be using in the assessment section. Also, the firewall policies in all architectures under consideration are generated by the DCG plane. In the setup phase, each firewall fetches its policy. In Architecture 1, the single firewall reads its policy, processes it, and stores it in a policy holder as shown in Figure 8.4. The same applies for each firewall in Architecture 2 in Figure 8.5. However, in Architecture 3, as each switch registers with the controller, its policy is read and processed at the control plane then pushed down to the flow tables of the switch at the data plane as shown in Figure 8.6. In the operation phase, switches in Architectures 1 and 2 forward the first packets of every communication to the control plane for checking policy and connection state. Once communication is established or communication is denied by the policy, flow table entries are inserted in the switch to handle future packets. However, Architecture 3 checks policy and track state connection at the switch without the need for any communication with the control plane.

Before we go on to explain the detailed implementation of each architecture, it is important to explain how the `ICMP`, `UDP`, and `TCP` communications are processed in a network. The `ICMP` protocol is a protocol that provides error and information messages for IP-based network. The *ping* application uses `ICMP` messages to test connectivity. It sends an echo request to a host and waits for the echo reply, if no reply is received within a certain period of time, it times out and the remote host is declared unreachable. The `UDP` protocol is a connectionless protocol which means there is no need to establish communication before sending data. *iPerf* sends a `UDP` stream from the sender host. In the end, the receiver host sends an acknowledgment to the sender host. The `TCP` protocol is a reliable connection-oriented

protocol. TCP needs to establish the communication before sending data. The connection is established using the three-way handshake in which three packets are exchanged between the communicating hosts. The first packet in the handshake, sent by the first host, is identified by setting the SYN bit. The second host replies with a second packet in which the SYN and ACK bits are set to indicate the acknowledgment of receiving the first packet and continuing the handshake. To which, the first host sends a third packet that has the ACK bit set to inform the other host of the establishment of the connection.

8.3.1 Implementation of Architecture 1

The first architecture is a single centralized firewall as shown in Figure 8.4. In this architecture, the firewall application creates a single firewall object responsible for enforcing the policies of all the switches in the data plane.

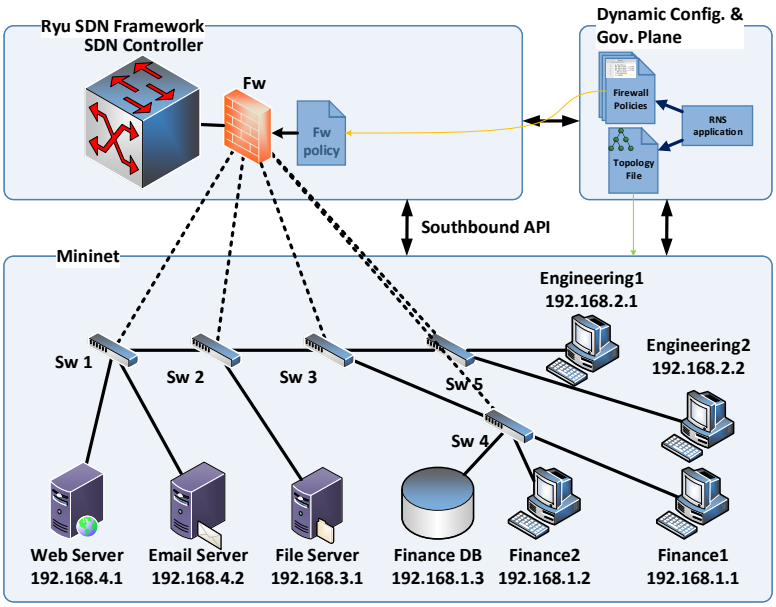


Figure 8.4: Architecture 1 - A single and centralized firewall at the control plane

To implement this architecture, we have created a firewall application attached to the

controller. It has a policy holder, a state table, a package that has protocol handlers, a handler for each protocol, and a module that implements the switch functionality or to communicate *OpenFlow* commands to the switch. In this architecture, the state is handled by the firewall, it has a single state table to keep track of all the connections for all the switches in the network. The same applies to the policy, in this architecture we have a single policy that gets enforced. Hence, the question is then how to obtain the policy of this unique firewall from the several policies to be deployed at the internal control points calculated by the RNS algorithm? The idea is to add to the state-space of the policies an attribute that indicates on what switch a rule is enforced. Hence, generating this global policy is straight forward from the policies of the internal access control points calculated by RNS algorithm.

Once this application is initiated and the firewall object is created at the control plane, it reads and stores its policy that governs the decisions of all the switches. The policy generated by the DCG plane for this architecture is one single policy. When the topology is created on *mininet*, each switch registers with the controller and the firewall instructs it to add a single rule with low priority to forward packets to the controller. When a switch receives a packet, it matches the added rule and forwards the packet to the firewall. The firewall inspects the packet header and assigns it to the appropriate handler. For example, a first `ICMP` packet in communication is checked against the state table. If the state table has no entry for it, the policy is checked. If the policy denies the packet, the firewall instructs the switch to drop the packet and adds an entry to its flow table to drop similar packets for a certain period of time. However, if the packet is allowed by the policy, the firewall adds an entry to the state table to handle the reply packet. It also instructs the switch to forward the packet and adds an entry to its flow table to forward similar packets for a certain period of time. When the reply packet arrives at the switch, the switch forwards the packet to the firewall. The firewall finds an entry in the state table, then updates the state table setting the connection state to established and instructs the switch to forward

the packet and add an entry to its flow table. A `UDP` connection is handled similarly. The firewall handles the `TCP` protocol handshake in a similar way except the first packet has an SYN flag, the second packet has an SYN-ACK flag, and the third packet has an ACK flag. The firewall prevents `DDoS` attack by keeping track of request packets that have no reply, if it passes a certain threshold, the firewall instructs the switch to add an entry to drop such packets and avoid overhead.

In the data plane switches, each time the firewall adds an entry to the flow table, it sets an expiry time for its usage. Once an entry reaches its expiry time, it is removed from the flow table and the firewall gets a notification by an *OpenFlow* message. Once it receives this notification message, the firewall application removes the state table entries.

8.3.2 Implementation of Architecture 2

Architecture 2 consists of multiple firewalls each one is responsible for managing a single switch as shown in Figure 8.5.

To implement this architecture, we have created a firewall application at the controller. It creates a firewall object for each switch. Each firewall object has a policy holder, a state table, a package that has a protocol handler for each protocol, and a module that implements the switch functionality or communicates *OpenFlow* commands to a switch. In this architecture, the state is handled by each firewall separately as it has a state table to keep track of connections for the assigned switch. Compared to Architecture 1, this architecture presents a better design as it applies the principle of separation of concerns: what concerns a switch is delegated to a firewall object.

In the setup phase, once a switch is created at the data plane and sends its features to the controller, the firewall application creates a firewall object designated for this switch. A firewall object reads the policy that governs the decisions of its switch and stores it. When the switch receives a packet that is not matched by any entry table in its flow tables, it forwards it to its corresponding firewall. The firewall checks its state table and, if needed,

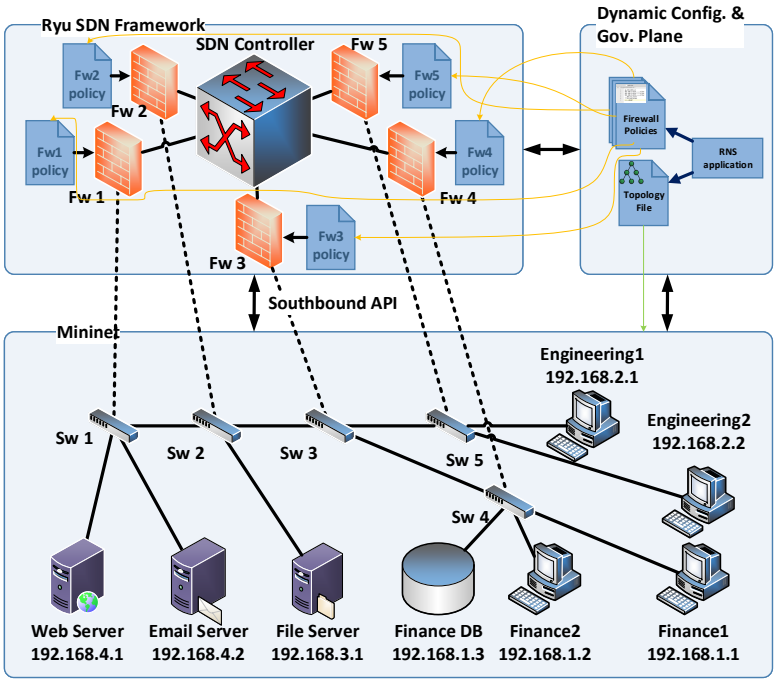


Figure 8.5: Architecture 2 - Multiple distributed firewalls at the control plane

the policy, then it instructs the switch on how to handle the packet. When a communication is established, the firewall instructs the switch to add an entry to its entry table to handle similar future packets.

8.3.3 Implementation of Architecture 3

Architecture 3 is a distributed firewall architecture at the data plane. In this architecture, we transform data plane switches into stateful firewalls using *BEBA software switch* as shown in Figure 8.6. Such that switches handle firewall rules and keep track of the connection state without forwarding traffic to the controller.

In this architecture, we use a state machine to handle each protocol. This is why each protocol is handled by a separate flow table as we explain below. A state machine for a protocol forwards legitimate packets or drops packets according to its state transitions, and

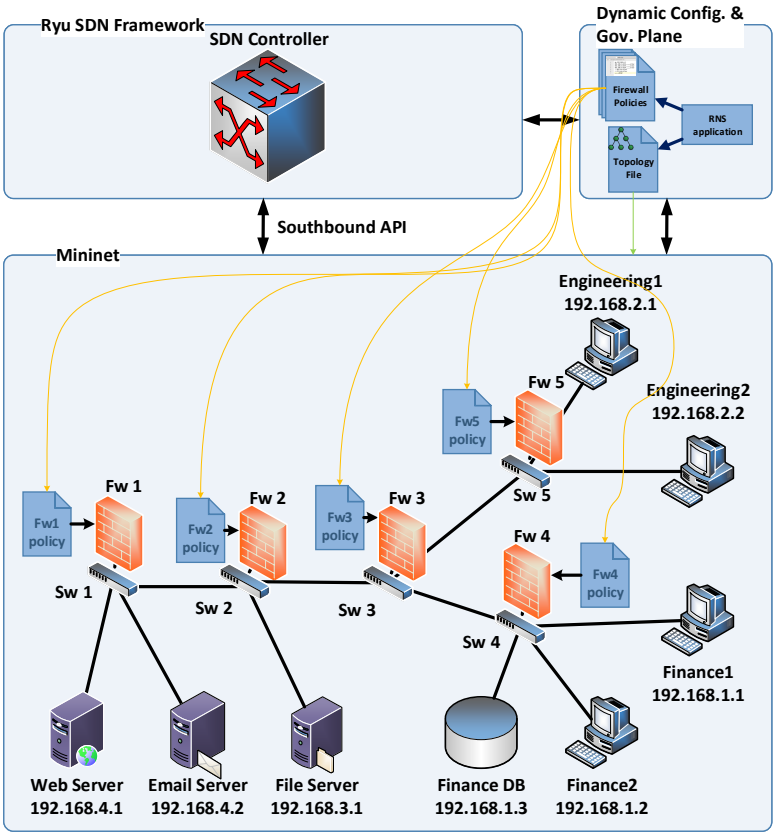


Figure 8.6: Architecture 3 - Multiple distributed firewalls at the data plane

changes state if needed. If a packet is part of an established connection or a connection to be established, then it is forwarded by the firewall. Otherwise, it is dropped.

In Architecture 3, each protocol is handled by a separate flow table which has a state table. Therefore, we have a state table for each of `TCP`, `UDP`, and `ICMP` protocols. Policies might involve rules that are specific to a protocol. Therefore, the policy at each switch can be split into (sub-)policies associated with the flow tables of the protocols. Each flow table enforces the rules that are relevant to its protocol.

To implement Architecture 3, we have created an application attached to the controller. The application consists of a main class, and five more classes each is assigned to handle

a flow table. The first class inserts rules in the first flow table, `T0`, to route packets to the appropriate flow table. The second class inserts flow entries in the flow table, `T1`, to handle `ICMP` packets. The third class inserts `TCP` packets rules in the flow table `T2`. The fourth class inserts rules in the flow table `T3` to handle `UDP` packets. The last class inserts entries in the flow table `T4`, which is intended to keep track of what port is assigned to communicate on each mac address.

In Architecture 3, the firewall application starts with the controller. When a switch sends a feature reply message to the controller (i.e., registers at the controller), the firewall object reads the policy associated with the switch and stores it. It then starts a first table object which inserts six rules in the first flow table, the first two rules are to drop every LLDP and IPV6 packets. The third, fourth, and fifth rules forward `ICMP`, `TCP`, and `UDP` to tables `T1`, `T2`, and `T3`, respectively. The sixth rule forwards packets not matched by the above rules to `T4`.

Afterward, the firewall starts the second object, which is to prepare the flow table `T1` to handle `ICMP` packets. The first *OpenFlow* command sets `T1` into a stateful table. Then it sets the lookup-scope attributes, which are the attributes that the switch extracts from the packet and matches with the state table entries. The lookup-scope in the `ICMP` case is the tuple (`source_ip`, `destination_ip`). The next attributes to be set at the switch are the update-scope attributes, which are the attributes used by the switch to update the state table. In `ICMP`, the update-scope is the tuple (`destination_ip`, `source_ip`). Finally, it is the step of inserting entries in the XFSM flow table. The first entry is to handle an established connection. It checks if the state found in the state table is 1 (i.e., established) then the switch forwards the packet to `T4` to be forwarded to the right port, and update the state for the other direction to 1 (i.e., established). This rule is given the highest priority. Then the object checks the policy rules and inserts flow entries for the rules that are related to `ICMP` or "all" packets. These rules are used to match first packets in a communication (i.e., state 0). An allow rule passes the packet to `T4` to be forwarded and updates the state

for the other direction to 1, otherwise, it drops the packet. An *allow* rule is given a medium priority and a *deny* rule is given a low priority. A first packet is processed by the state table which does not match any entry and is given the default state 0. The XFSM table processes the packet to either forward it and update the state table, or to drop the packet. The third and fourth objects have similar functionality to handle `TCP` and `UDP` packets. The fifth object is intended to update the `T4` entries, which are for keeping track of the ports associated with every mac address.

8.4 Assessment of The Architectures

8.4.1 Testbed Environment

To assess the three architectures presented in Section 8.3, we used the following components:

- *mininet* 2.2.2 [dOSSP14]: It is a tool used to emulate and prototype SDNs, running on Ubuntu 14.04.4 (64 bit) virtual machine on VirtualBox 6.0.
- BEBA controller: It is based on Ryu OpenFlow Controller 3.29.
- *BEBA software switch* supporting *OpenFlow* 1.3.

For generating flows and collecting measurement data we used *ping*, *Wireshark* 1.10.6, and *iPerf* 2.0.5 utilities. The testbed environment is setup on a MacBook Pro with a CPU 2.7 GHz Intel Core i5 and a memory of 8 GB 1867 MHz DDR3.

8.4.2 Experiment Use Case

Selection of the topology to be used in the testbed

The topology used in the testbed is the topology generated by the RNS module in the DCG plane for a network composed of the resources of the illustrative example presented in Chapter 3. The topology can be seen in the Figures 8.4, 8.5, and 8.6.

The low-level policies of these resources fed to the RNS module in the DCG plane. The DCG plane generates the topology as shown in Figures 8.4, 8.5, and 8.6.
To test the dynamic nature of the network, we added two admin resources to the set of resources. The policy of these resources is to allow access for traffic generated from the admin resources only. These resources are to gain access to every resource in the network and this is achieved by updating the policies of all network resources. The DCG plane recalculates the topology and generates updated firewall policies.

Selection of the data flow to be used in the testbed

In the following section, we present some results for tests done on the testbed. First, we measure the setup cost for the different architectures based on the number of exchanged packets between the data plane and the control plane. Second, we assess the reachability by verifying the correctness of the implemented policies. Then, for the traffic originating from *Engineering1* to *Web server*, we measure the response time, bandwidth, and latency variation. Finally, to have an idea about the effect of the dynamic aspects of the network topology, we measure the cost of updating the topology.

8.5 Results and discussion

In this section, we present and discuss the results of the assessment of the three architectures. The tests are done on the SDN topology generated by the DCG plane for the network resources mentioned above. We have tested the three architectures for the topology shown in Figures 8.4, 8.5, and 8.6.

8.5.1 Setup Cost

One of the criteria to compare the architectures is the cost for the setup of the network. The cost is measured by the number of exchanged packets between the control plane and

the data plane in the setup phase. For Architectures 1 and 2, 46 packets are exchanged to complete the setup. Architecture 3 took on average 1508 packets. We performed 10 tests on Architecture 3 and the minimum number of packets exchanged is 1459 and the maximum number is 1742. This is due to the fact that in Architectures 1 and 2 the only messages exchanged are the setup messages, while in Architecture 3 besides setup messages, the messages for setting stateful tables, inserting flow table entries are also exchanged at this stage.

We can notice that Architecture 3 is significantly high in the setup cost compared to architectures 1 and 2. This is due to the fact that in Architectures 1 and 2 the only messages exchanged are the setup messages, while in Architecture 3 besides setup messages, the messages for setting stateful tables, inserting flow table entries are exchanged at this stage.

8.5.2 Reachability

After the setup of the environment, the first test we perform on all architectures is whether the policies are enforced as expected, and for that, we did a reachability test between all the resources in the network. Architectures 1, 2, and 3 all have the same result as shown in Figure 8.7. The reachability test is done by the command *pingall*, where every host tries to ping every other host in the data plane. In the first line in the result, we see a label at each line (i.e., `eng1`) which is the name of the resource initiating the ping request. After the arrow `->` a resource name (i.e., `eng2`) indicates a successful communication with that resource (i.e., reply), and an `x` indicates a failed communication (i.e., blocked). For example, `eng1` is able to access `eng2`, `file`, `web`, and `email` while failing to access `fin1`, `fin2`, and `finDB`.

8.5.3 Response Time

The response time or latency test measures the time it takes to get a reply for a request. We carried this test by sending several `ICMP` packets using *ping*. We measure the response

```
mininet> pingall
*** Ping: testing ping reachability
eng1 -> eng2 X X X file web email
eng2 -> eng1 X X X file web email
fin1 -> X X fin2 finDB file web email
fin2 -> X X fin1 finDB file web email
finDB -> X X fin1 fin2 file web email
file -> X X X X X web email
web -> X X X X X X email
email -> X X X X X X web
*** Results: 51% dropped (27/56 received)
mininet>
```

Figure 8.7: Reachability test

time for the traffic originating from *Engineering1* to *Web server*.

In Figure 8.8, we present the results for only 10 `ICMP` packets as the results are the same for a number above 10 packets. Figure 8.8 shows that the first packet took about the same time for Architectures 1 and 2 and less time for Architecture 3. The reason is that switches in Architectures 1 and 2 do not have entries in their flow tables to handle the packet, and the packet is forwarded to the controller. On the other hand, switches in Architecture 3 have entries in their flow tables to handle the packet, which reduces the time to handle the first packet. Moreover, Architecture 1 and Architecture 2 needed to exchange 32 packets between data plane switches and the controller while Architecture 3 does not exchange any packets. Architectures 1, 2, and 3 show similar response times for the subsequent packets. The reason is that for the subsequent packets switches already have entries in all architectures to handle the packets that follow the first packet.

To see the difference in response time between the three architectures more clearly, we have created three different topologies with the *Engineering1* on one end and *Web server* on the other end. The three topologies consist of $1, 10$, and 20 switches between these resources. These switches enforce the same policy. In all tests, *Engineering1* sends a single `ICMP` packet (i.e., ping request) and we measured the response time. Figure 8.9 shows the average results for 10 tests. The average times for the case of a single switch for Architectures 1, 2, and 3 are 30.31 ms, 29.5 ms, and 1.18 ms, respectively. The average times for the case of 10 switches for Architectures 1, 2, and 3 are 1312.2 ms, 1300.3 ms, and 607.5 ms, respectively. Finally, the average times for the case of 20 switches for Architecture 1, 2, and 3 are 4963.3 ms,

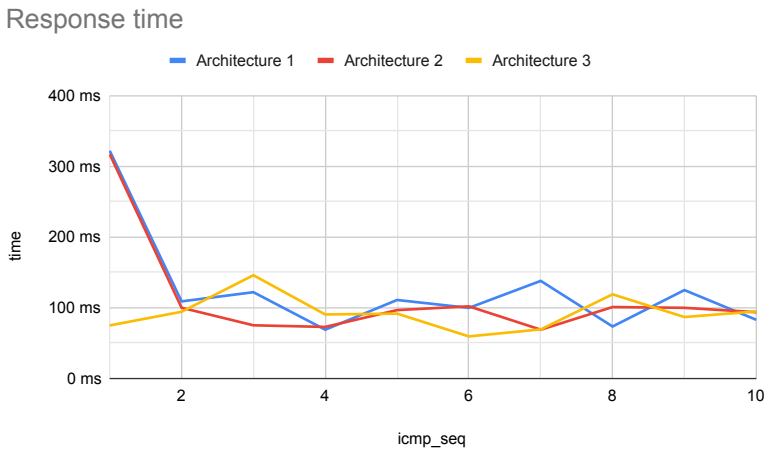


Figure 8.8: Response time for 10 `ICMP` packets

5003 ms, and 2116.5 ms, respectively. It is clear from Figure 8.9 that Architectures 1, and 2 has similar response time and Architecture 3 has a significantly less time.

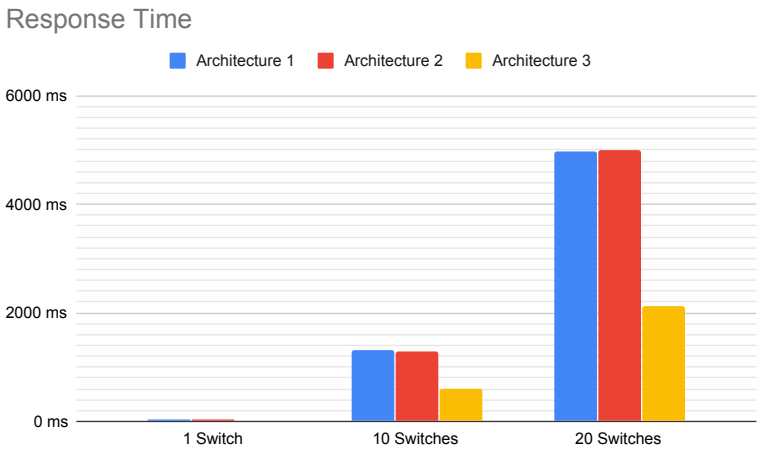


Figure 8.9: Response time for topologies of $1, 10$, and 20 switches

8.5.4 Bandwidth

One of the tests we performed to compare the three architectures is network bandwidth. Network bandwidth is the maximum rate or volume of data that can be transferred on a

link per unit of time. The bandwidth test is done by performing a `TCP` communication using *iPerf* between the selected hosts (i.e., `eng1` and `web`). The result is shown in Table 8.1 and broken down into 0.5 second intervals in Figure 8.10. Note that bandwidth test between virtual hosts depends on the host machine CPU speed. The bandwidth for the virtual host in *mininet* varies because of the host machine running processes and CPU load at a given time [BJD16]. Therefore, the three architectures will not vary in bandwidth as they use the same environment with the same switches and links.

	Interval	Transfer	Bandwidth
Architecture 1	`0.0-20.4 sec`	`512 KBytes`	`206 Kbits/sec`
Architecture 2	`0.0-19.0 sec`	`512 KBytes`	`221 Kbits/sec`
Architecture 3	`0.0-20.4 sec`	`512 KBytes`	`205 Kbits/sec`

Table 8.1: Network bandwidth for Architectures 1,2, and 3

For a `TCP` communication, Architecture 1 and Architecture 2 exchange 154 packets between the data plane and control plane while Architecture 3 does not exchange any packet.

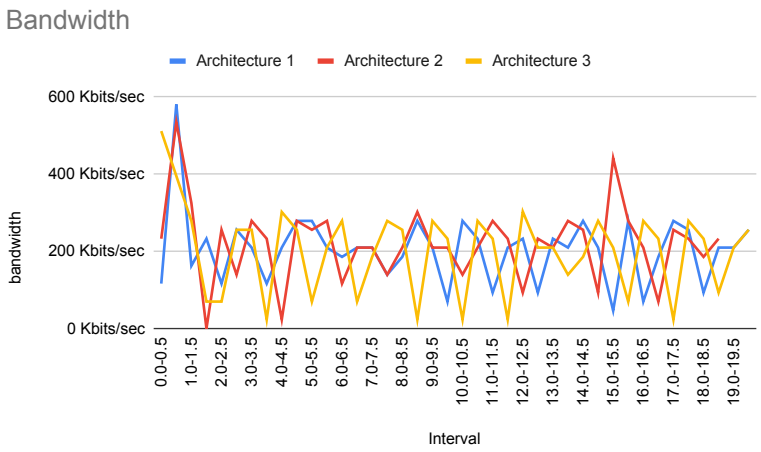


Figure 8.10: Bandwidth

8.5.5 Latency Variation

Latency variation or jitter is the variance in delay time or latency between packets' arrival. The jitter test is done by performing a `UDP` communication using *iPerf* between the selected hosts (i.e., `eng1` and `web`). The result is shown in Table 8.2 which is broken down into 0.5 second intervals in Figure 8.11. We notice that the three architectures have very similar jitter and that it is not affected by the difference in architectures for the same reasons discussed for the bandwidth.

Architecture 1 and Architecture 2 exchange 45 packets in `UDP` connection while 0 packets are exchanged in Architecture 3.

	Interval	Transfer	Bandwidth	Jitter
Architecture 1	`0.0-8.8 sec`	`1.09 MBytes`	`1.05 Mbits/sec`	`17.349 ms`
Architecture 2	`0.0-9.1 sec`	`1.14 MBytes`	`1.05 Mbits/sec`	`16.792 ms`
Architecture 3	`0.0-9.9 sec`	`1.25 MBytes`	`1.06 Mbits/sec`	`15.962 ms`

Table 8.2: Network jitter for Architectures 1,2, and 3

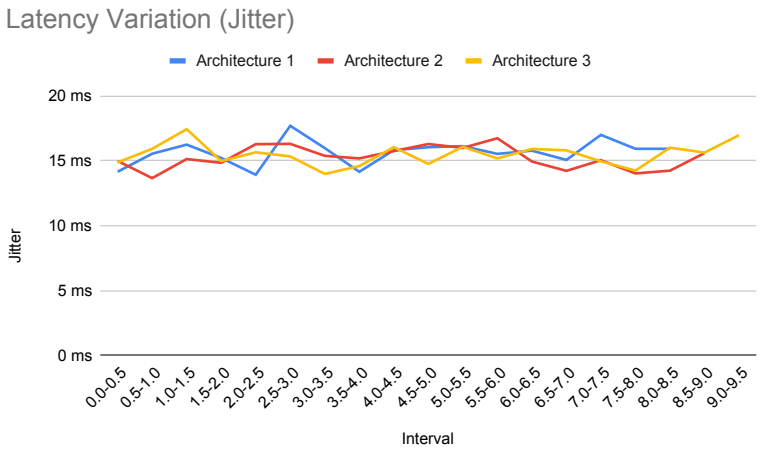


Figure 8.11: Latency variation (jitter)

Looking at the tests done on the operation phase which are the latency, bandwidth, and jitter, we can notice the following. First, results are similar for bandwidth and jitter tests

for the three architectures as they are based on the environment consisting of topology, switches, and links which are the same for our architectures. Second, the result for the response time is high in Architecture 1 and Architecture 2 and low in Architecture 3. The reason for this is the time architectures 1 and 2 take to add entries in the flow table of the switches in response to the first packet. Finally, we also notice that for all these tests, Architecture 1 and Architecture 2 exchange packets between the data plane and the control plane while Architecture 3 does not exchange any packet.

8.5.6 Resilience to Topology Change

In network topology, many changes to the topology can occur. These changes can be intentional and planned for by network administrators such as adding resources, removing resources, or changing resource policies. Moreover, in a dynamic environment, resources can be in and out of the network at will. Hence for efficient security, the governance of the network access needs to shadow the changes. In any of these cases, the RNS module in the DCG plane is re-executed on the fly to generate an updated topology and firewall policies. Then, the data plane topology running on *mininet* needs to be updated at run-time. For this purpose, the updated topology is compared to the old topology and a topology update script is created. The script contains *mininet* commands to delete or add hosts, switches, or links. We run the script on *mininet* CLI. The controller gets notified for such topology changes, and it updates firewall policies that are provided by the DCG plane.

To assess the effect of a change, we added two administration workstations. These workstations can access all resources and they allow access between themselves and deny everything else. we use this case to re-execute the RNS module and generate new firewall policies and an updated topology. The *mininet* topology is updated on the fly as explained above.

To compare the response for the update by the three architectures, we evaluate the update cost of each architecture which is measured by the number of exchanged packets between

the control and data plane to fulfill the update. In our example, Architectures 1 and 2 exchanged 54 packets between control and data planes. Architecture 3 exchanged 1670 packets on average. The difference is huge between Architectures 1 and 2 on one side and Architecture 3 on the other side.

We also did the reachability test of all the architectures after the topology update. The result for the architecture 1, 2, and 3 is shown in Figure 8.12.

```
mininet> pingall
*** Ping: testing ping reachability
eng1 -> eng2 X X X file web email X X
eng2 -> eng1 X X X file web email X X
fin1 -> X X fin2 finDB file web email X X
fin2 -> X X fin1 finDB file web email X X
finDB -> X X fin1 fin2 file web email X X
file -> X X X X X web email X X
web -> X X X X X X email X X
email -> X X X X X X web X X
admin1 -> eng1 eng2 fin1 fin2 finDB file web email admin2
admin2 -> eng1 eng2 fin1 fin2 finDB file web email admin1
*** Results: 50% dropped (45/90 received)
mininet>
```

Figure 8.12: Reachability test after topology update

The use of *mininet* does not allow to test the effect of topology changes perfectly with *BEBA software switch* as it is not supported fully by *mininet*. Instead, *mininet* fully supports *Open vSwitch*. One of the future suggestions is to implement the RNS algorithm using the new version of *Open vSwitch*.

8.6 SDN Implementation for Multiple Entry Networks

We have implemented the NSR and SSR algorithms presented in Chapter 7 in an SDN environment. They have been implemented as modules in the DCG plane shown in Figure 8.1. Each module takes as input sets of policies of resources that need to be accessed from the given entry points. If we have n entry points, we should input n sets of policies of the resources. Each set contains the policies of the resources accessed from its corresponding entry point. Each of the modules generates a topology of the data plane along with policies to be enforced at each switch. They also generate a single policy that combines the policies of the switches to be used by the single firewall.

We adapt the same three architectures presented in Section 8.3 to implement the NSR and SSR algorithms. The above conclusions obtained using the RNS algorithm are also preserved when we use NSR, and SSR algorithms. They are related to network setup procedures and the way packets are exchanged between the switches and the controller. Hence, they cannot be affected by the algorithms used in the DCG plane.

We have used the topologies generated by the NSR and SSR algorithms for the resources of the illustrative example of a network with multiple entry points presented in Section 7.2. The topologies are tested in all three architectures. In all cases, after the setup of the environment, we have performed a reachability test to confirm the enforcement of the corrected policies as shown in Figure 8.13. The three architectures using NSR and SSR algorithms show expected reachability results: the policies are enforced correctly.

```
mininet> pingall
*** Ping: testing ping reachability
filea2 -> filea1 weba X X X X emaila X X X X X X X X X
filea1 -> filea2 weba X X X X emaila X X X X X X X X X
weba -> X X X X X X emaila X X X X X X X X X
hra1 -> filea2 filea1 weba X X hra2 emaila X X X X X X hrs1 hrs2 X X
fina1 -> filea2 filea1 weba X fina2 X emaila X X X X X X X X fins1 fins2
fina2 -> filea2 filea1 weba X fina1 X emaila X X X X X X X X fins1 fins2
hra2 -> filea2 filea1 weba hra1 X X emaila X X X X X X hrs1 hrs2 X X
emaila -> X X weba X X X X X X X X X X X X X
emailb -> X X X X X X X X X X webb X X X X X
hrb2 -> X X X X X X X X emailb X X webb hrb1 hrs1 hrs2 X X
finb2 -> X X X X X X X X X emailb X finb1 webb X X X fins1 fins2
finb1 -> X X X X X X X X X emailb X finb2 webb X X X fins1 fins2
webb -> X X X X X X X X emailb X X X X X X X X
hrb1 -> X X X X X X X X emailb hrb2 X X webb hrs1 hrs2 X X
hrs1 -> X X X X X X X X X X X X X X hrs2 X X
hrs2 -> X X X X X X X X X X X X X X hrs1 X X
fins1 -> X X X X X X X X X X X X X X X X X fins2
fins2 -> X X X X X X X X X X X X X X X X X fins1
*** Results: 79% dropped (62/306 received)
mininet>
```

Figure 8.13: Reachability test of multiple entry points

8.7 Summary

In this chapter, we have presented a practical implementation of the RNS algorithm into an SDN environment. The implementation is achieved by the introduction of the DCG plane, which is responsible for structuring network resources and configuring policies. The new plane provides an adaptive security solution for dynamic changes in the network. We have

proposed three different architectures for incorporating RNS into SDN environment. We assessed each architecture and drew conclusions for which architecture is most suitable for which scenario.

Chapter 9

Conclusion and Future Work

As discussed in Section 1.2, there is a lack of formalism tackling the strategies of layered protection and segmentation for designing secure network architectures. This thesis proposes a mathematical framework for network segmentation from an access control policy perspective. Moreover, the thesis gives a formalism for configuring network resources and structuring network topologies to achieve layered protection and segmentation. Furthermore, the thesis presents an implementation of this formalism into an SDN environment. In this chapter, we highlight the contributions of the thesis and point to future work.

9.1 Highlights of the Contributions

The contributions related to the material presented in this thesis include:

(1) **A mathematical approach for analyzing network policies that is based on product family**: In this thesis, we have proposed a model for network resource access control policies based on guarded commands and PFA. The model uses a family approach where a resource policy is thought of as a set of policies coming from different stakeholder perspectives such as management, laws, and security administration. Another usage of the family approach in the context of a network is the abstraction of

the policies of the resources of a network/subnetwork. The policy enforced at the entry point of the network/subnetwork is the abstraction of the network which is the policy family of its resource policies.

The network policies are abstracted at the entry point of the network as a family of policies at the firewall or node governing the access for these resources. For example, a lab or a building. We have also proposed a weight function that gives weight to policies based on certain security requirements. The function is used in the RNS algorithm to place segments within the network.

(2) **A formalism for layered protection**: In this thesis, we have proposed a formal understanding of layered protection: the DD strategy. This formalism allows us to assess whether a given network with its resource policies satisfies the DD strategy. Moreover, we used a calculation approach to assign policies of network resources such that it satisfies the DD strategy.

We also defined a stronger version of DD strategy which is SDD. We have also presented a set of results indicating when SDD is not achievable in a given network.

(3) **An approach to build secure networks**: In this thesis, we formally defined segmentation and robust network architecture. Then, we have proposed the RNS algorithm to build the robust network architecture which segments resources such that maximum access control protection is provided to the resources. We have also extended the RNS algorithm to networks with multiple entry points.

The advantage of building a robust network using the RNS Algorithm is that public-facing resources are placed close to the edge of the network and have limited access to internal resources. While resources requiring more security are placed farther from the edge of the network and protected by many layers. Furthermore, access from one segment to the other is controlled by internal access control points. This is a common

practice adopted in network segmentation that is driven by intuition. We gave the mathematical rationale for it. Moreover, the advantage of having a formal treatment of the problem allows for the automation of the solution. By automating the solution, networks with a huge number of resources can be handled efficiently.

(4) **An SDN implementation of RNS**: In this thesis, we have shown the validity of the RNS algorithm in a modern network scenario. We have implemented three different architectures for RNS in an SDN environment. We have assessed these architectures and drawn some conclusions and recommendations for implementing RNS in SDN. We have introduced RNS as a separate plane that provides adaptive security. It provides robust and secure topology and configuration for data plane resources. Moreover, it dynamically adapts the topology for any changes in the environment such as policy change or resource availability.

9.2 Future Work

The work presented in the thesis can be extended in many directions. This can include theory, applications, and tools.

9.2.1 Theory: Models and Techniques

(1) In Chapter 2, we have presented an approach introduced in [HPF16] which uses a family approach to model FQAs such as security policies. It uses an aspect-oriented approach to insert the configuration into the application model. The aspect-oriented is also used to insert changes in the security policy. The issue of frequent policy changes is relevant to network access control policies. An aspect-oriented approach would be applicable to quickly handle changes in the network policy without disturbing enforced policy or network topology. Our approach is built on PFA. There is an extension of PFA

which support aspect-oriented which is Aspect-Oriented Product Family Algebra (AO-PFA) [ZKJ12, ZKJ14, ZK16]. Therefore, the formalism presented in the thesis can be extended to handle policy changes using an aspect-oriented approach. This work and its effect on overall network security remain open for further investigation and exploration.

(2) The proposed RNS Algorithm could be scaled to enforce additional requirements or constraints on the segmentation to achieve the desired protection. For example, an organization can limit the number of levels required. Furthermore, additional measures can be added to achieve the desired segmentation. For example, risk assessment measures [WŞW+16] such as infection spread ratio and cleansing time can be added to the requirement of the network. Note that these additional factors can be incorporated into the weight function.

9.2.2 Applications

With respect to the possible applications of the proposed work, the following directions can be investigated further:

(1) A huge body of work focusing on experimental evaluation in several contexts such as evaluating the RNS algorithm in dynamic networks and IoT is needed. Furthermore, based on the evaluations in different settings, one could propose improvements to the RNS algorithm (as it is the base of the other algorithms) to better meet the needs of some settings.

(2) The formalism presented in this thesis is based on the perspective of network access control policies. However, the range of the policies domain can be explored further. This can be applied to other network policies such as routing, NAT, IDS, etc.

9.2.3 Tools and Automation

As part of the work of this thesis, we have presented an implementation to validate the theories presented in the thesis. However, this can be extended in many ways.

(1) A tool that given an established network architecture, finds flaws in that architecture, and proposes changes which would enhance the segmentation and protection of the network resources without disturbing the whole structure of the network.

(2) A tool for the overall management of policies within networks which extends the implementation presented in this thesis. The extensions include validation, consistency check, and monitoring.

9.3 Closing Remarks

Security concerns are some of the limiting factors of modern networks' ongoing growth. Security solutions for these networks to be effective need to be adaptive to cope with the dynamic nature of their environment. In this thesis, we have presented a formalism for the network governance of dynamic networks. The formalism provides an adaptive approach for segmenting and configuring network resources. This work provides a formal approach that takes into consideration the security requirements of local resources and the global overall network. Moreover, the work provides a formal foundation for automated security solutions that are applicable to many domains of connected resources.

Appendix A

Detailed Proofs

A.1 Detailed Proof of Proposition 5.1.1

Proof. a) $(\forall\, i \mid 1 \le i \le m \cdot p(v_m) \sqsubseteq_{\mathcal{F}} p(v_i))$

$\iff$ $\langle\ 1 \le i \le m \iff 1 \le i \le m-1 \vee i = m\ \rangle$

$(\forall\, i \mid 1 \le i \le m-1 \vee i = m \cdot p(v_m) \sqsubseteq_{\mathcal{F}} p(v_i))$

$\iff$ $\langle$ Range Split and One Point Axiom, and Reflexivity of $\sqsubseteq_{\mathcal{F}}$ $\rangle$

$(\forall\, i \mid 1 \le i \le m-1 \cdot p(v_m) \sqsubseteq_{\mathcal{F}} p(v_i)) \wedge \mathsf{true}$

$\iff$ $\langle$ Identity of $\wedge$ $\rangle$

$(\forall\, i \mid 1 \le i \le m-1 \cdot p(v_m) \sqsubseteq_{\mathcal{F}} p(v_i))$

$\iff$ $\langle\ 1 \le i \le m-1 \iff 1 \le i \le m-2 \vee i = m-1\ \rangle$

$(\forall\, i \mid 1 \le i \le m-2 \vee i = m-1 \cdot p(v_m) \sqsubseteq_{\mathcal{F}} p(v_i))$

$\iff$ $\langle$ Range Split and One Point Axiom $\rangle$

$(\forall\, i \mid 1 \le i \le m-2 \cdot p(v_m) \sqsubseteq_{\mathcal{F}} p(v_i)) \wedge p(v_m) \sqsubseteq_{\mathcal{F}} p(v_{m-1})$

$\iff$ $\langle$ Since $(v_{m-1}, v_m) \in E \implies p(v_{m-1}) \sqsubseteq_{\mathcal{F}} p(v_m)$ and transitivity of $\sqsubseteq_{\mathcal{F}}$ $\rangle$

$(\forall\, i \mid 1 \le i \le m-3 \cdot p(v_m) \sqsubseteq_{\mathcal{F}} p(v_i)) \wedge \mathsf{true}$

$\iff$ $\langle$ Identity of $\wedge$ $\rangle$

$(\forall\, i \mid 1 \le i \le m-2 \cdot p(v_m) \sqsubseteq_{\mathcal{F}} p(v_i))$

$\iff$ $\langle\ 1 \le i \le m-2 \iff 1 \le i \le m-3 \lor i = m-2\ \rangle$

$(\forall\, i \mid 1 \le i \le m-3 \lor i = m-2 \cdot p(v_m) \sqsubseteq_{\mathcal{F}} p(v_i)\,)$

$\iff$ $\langle$ Range Split and One Point Axiom $\rangle$

$(\forall\, i \mid 1 \le i \le m-3 \cdot p(v_m) \sqsubseteq_{\mathcal{F}} p(v_i)\,) \land p(v_m) \sqsubseteq_{\mathcal{F}} p(v_{m-2})$

$\iff$ $\langle$ Since $(v_{m-2}, v_{m-1}) \in E \implies p(v_{m-1}) \sqsubseteq_{\mathcal{F}} p(v_{m-2})$ and transitivity of $\sqsubseteq_{\mathcal{F}}$ $\rangle$

$(\forall\, i \mid 1 \le i \le m-3 \cdot p(v_m) \sqsubseteq_{\mathcal{F}} p(v_i)\,) \land \mathsf{true}$

$\iff$ $\langle$ Identity of $\land$ $\rangle$

$(\forall\, i \mid 1 \le i \le m-3 \cdot p(v_m) \sqsubseteq_{\mathcal{F}} p(v_i)\,)$

$\iff$ $\langle$ Range Split several times and transitivity of $\sqsubseteq_{\mathcal{F}}$ $\rangle$

true

b) Let $Q(m) \stackrel{\text{def}}{\Leftrightarrow} (\forall\, i \mid 1 \le i \le m \cdot p(v_i) \sqsubseteq_{\mathcal{F}} p(v_1)\,)$, for some $m \in \mathbb{N}$.

Base Case: $Q(1) \stackrel{\text{def}}{\Leftrightarrow} (\forall\, i \mid 1 \le i \le 1 \cdot p(v_i) \sqsubseteq_{\mathcal{F}} p(v_1)\,)$, which is obviously true due to the *One Point Axiom* and the reflexivity of $\sqsubseteq_{\mathcal{F}}$.

Inductive Step: For arbitrary $m \ge 1$, we prove $Q(m+1)$ using the hypotheses ($Q(m)$ is true) and (G employs a DD strategy).

$(\forall\, i \mid 1 \le i \le m+1 \cdot p(v_i) \sqsubseteq_{\mathcal{F}} p(v_1)\,)$

$\iff$ $\langle\ 1 \le i \le m+1 \iff 1 \le i \le m \lor i = m+1\ \rangle$

$(\forall\, i \mid 1 \le i \le m \lor i = m+1 \cdot p(v_i) \sqsubseteq_{\mathcal{F}} p(v_1)\,)$

$\iff$ $\langle$ Range Split and One Point Axiom $\rangle$

$(\forall\, i \mid 1 \le i \le m \cdot p(v_i) \sqsubseteq_{\mathcal{F}} p(v_1)\,) \land p(v_{m+1}) \sqsubseteq_{\mathcal{F}} p(v_1)$

$\iff$ $\langle$ From the hypothesis $Q(m)$ is true $\rangle$

$\mathsf{true} \land p(v_{m+1}) \sqsubseteq_{\mathcal{F}} p(v_1)$

$\iff$ $\langle$ From (a), and Idompotency of $\land$ $\rangle$

true

□

A.2 Detailed Proof of Lemma 5.2.1

Proof.

$\neg(p(r) \neq 0_{\mathcal{F}} \wedge (\forall\, a, b \mid (a, b) \in E \cdot p(b) \sqsubset_{\mathcal{F}} p(a)))$

$\iff$ ⟨ De Morgan Law ⟩

$p(r) = 0_{\mathcal{F}} \ \vee \ (\exists\, a, b \mid (a, b) \in E \cdot p(b) \not\sqsubset_{\mathcal{F}} p(a))$

$\impliedby$ ⟨ Strengthening ⟩

$(\exists\, a, b \mid (a, b) \in E \cdot p(b) \not\sqsubset_{\mathcal{F}} p(a))$

$\impliedby$ ⟨ $(v, l) \in E$ ⟩

$p(l) \not\sqsubset_{\mathcal{F}} p(v)$

$\impliedby$ ⟨ $p(v) = \texttt{gcd}(p(v), p(l))$ ⟩

$p(l) \not\sqsubset_{\mathcal{F}} \texttt{gcd}(p(v), p(l))$

$\impliedby$ ⟨ From the hypothesis $p(l) = 1_{\mathcal{F}}$ ⟩

$1_{\mathcal{F}} \not\sqsubset_{\mathcal{F}} \texttt{gcd}(p(v), 1_{\mathcal{F}})$

$\impliedby$ ⟨ Because $1_{\mathcal{F}}$ is the annihilator of gcd ⟩

$1_{\mathcal{F}} \not\sqsubset_{\mathcal{F}} 1_{\mathcal{F}}$

$\impliedby$ ⟨ From the definition of $\sqsubset_{\mathcal{F}}$ ⟩

true

□

A.3 Detailed Proof of Lemma 5.2.2

Proof.

$\neg(p(r) \neq 0_{\mathcal{F}} \wedge (\forall\, a, b \mid (a, b) \in E \cdot p(b) \sqsubset_{\mathcal{F}} p(a)))$

$\Longleftrightarrow$ $\langle$ De Morgan Law $\rangle$

$p(r) = 0_{\mathcal{F}} \vee (\exists\, a, b \mid (a, b) \in E \cdot p(b) \not\sqsubset_{\mathcal{F}} p(a))$

$\Longleftarrow$ $\langle$ Strengthening $\rangle$

$(\exists\, a, b \mid (a, b) \in E \cdot p(b) \not\sqsubset_{\mathcal{F}} p(a))$

$\Longleftarrow$ $\langle$ $(r, s) \in E$ $\rangle$

$p(s) \not\sqsubset_{\mathcal{F}} p(r)$

$\Longleftarrow$ $\langle$ $p(s) = \texttt{gcd}(p(s), \texttt{gcd}(p(l_1), p(l_2)))$ and $p(r) = \texttt{gcd}(p(r), p(s))$ $\rangle$

$\texttt{gcd}(p(s), \texttt{gcd}(p(l_1), p(l_2))) \sqsubset_{\mathcal{F}} \texttt{gcd}(p(r), p(s))$

$\Longleftarrow$ $\langle$ From the hypothesis $\texttt{gcd}(l_1, l_2) = 1_{\mathcal{F}}$ $\rangle$

$\texttt{gcd}(p(s), 1_{\mathcal{F}}) \sqsubset_{\mathcal{F}} \texttt{gcd}(p(r), p(s))$

$\Longleftarrow$ $\langle$ Because $1_{\mathcal{F}}$ is the annihilator of $\texttt{gcd}$ $\rangle$

$1_{\mathcal{F}} \not\sqsubset_{\mathcal{F}} \texttt{gcd}(p(r), 1_{\mathcal{F}})$

$\Longleftarrow$ $\langle$ Because $1_{\mathcal{F}}$ is the annihilator of $\texttt{gcd}$ $\rangle$

$1_{\mathcal{F}} \not\sqsubset_{\mathcal{F}} 1_{\mathcal{F}}$

$\Longleftarrow$ $\langle$ From the definition of $\sqsubset_{\mathcal{F}}$ $\rangle$

true

□

A.4 Detailed Proof of Lemma 5.2.3

Proof.

$\neg(p(r) \neq 0_{\mathcal{F}} \wedge (\forall\, a, b \mid (a, b) \in E \cdot p(b) \sqsubset_{\mathcal{F}} p(a)\,))$

$\iff$ $\langle$ De Morgan Law $\rangle$

$p(r) = 0_{\mathcal{F}} \vee (\exists\, a, b \mid (a, b) \in E \cdot p(b) \not\sqsubset_{\mathcal{F}} p(a)\,)$

$\impliedby$ $\langle$ Strengthening $\rangle$

$(\exists\, a, b \mid (a, b) \in E \cdot p(b) \not\sqsubset_{\mathcal{F}} p(a)\,)$

$\impliedby$ $\langle$ $(u, v) \in E$ $\rangle$

$p(v) \not\sqsubset_{\mathcal{F}} p(u)$

$\impliedby$ $\langle$ $p(u) = (\texttt{gcd}\, v_i \mid (u, v_i) \in E \cdot p(v_i)\,)$ $\rangle$

$p(v) \not\sqsubset_{\mathcal{F}} (\texttt{gcd}\, v_i \mid (u, v_i) \in E \cdot p(v_i)\,)$

$\impliedby$ $\langle$ From the hypothesis $p(v) = (\texttt{gcd}\, v_i \mid (v, v_i) \in E \cdot p(v_i)\,)$ $\rangle$

$p(v) \not\sqsubset_{\mathcal{F}} p(v)$

$\impliedby$ $\langle$ From the definition of $\sqsubset_{\mathcal{F}}$ $\rangle$

true

□

A.5 Detailed Proof of Lemma 5.2.4

Proof.

$\neg(p(r) \neq 0_{\mathcal{F}} \wedge (\forall\, a, b \mid (a, b) \in E \cdot p(b) \sqsubset_{\mathcal{F}} p(a)\,))$

$\iff$ $\langle$ De Morgan Law $\rangle$

$p(r) = 0_{\mathcal{F}} \ \vee \ (\exists\, a, b \ \mid \ (a, b) \in E \ \cdot \ p(b) \not\sqsubseteq_{\mathcal{F}} p(a)\,)$

$\Longleftarrow$ $\langle$ Strengthening $\rangle$

$(\exists\, a, b \ \mid \ (a, b) \in E \ \cdot \ p(b) \not\sqsubseteq_{\mathcal{F}} p(a)\,)$

$\Longleftarrow$ $\langle$ $(u, v) \in E$ as a counter example $\rangle$

$p(v) \not\sqsubseteq_{\mathcal{F}} p(u)$

$\Longleftarrow$ $\langle$ $p(u) = (\oplus\, v_i \ \mid \ (u, v_i) \in E \ \cdot \ p(v_i)\,)$ $\rangle$

$p(v) \not\sqsubseteq_{\mathcal{F}} (\oplus\, v_i \ \mid \ (v, v_i) \in E \ \cdot \ p(v_i)\,)$

$\Longleftarrow$ $\langle$ From the property $p \sqsubseteq_{\mathcal{F}} q \oplus r \Leftrightarrow p \sqsubseteq_{\mathcal{F}} q \vee p \sqsubseteq_{\mathcal{F}} r$ $\rangle$

$p(v) \not\sqsubseteq_{\mathcal{F}} p(v_1) \wedge \cdots \wedge p(v) \not\sqsubseteq_{\mathcal{F}} p(v) \wedge \cdots \wedge p(v) \not\sqsubseteq_{\mathcal{F}} p(v_n)$

$\Longleftarrow$ $\langle$ From the definition of $\sqsubset_{\mathcal{F}}$ $\rangle$

$true \wedge \cdots \wedge true \wedge \cdots \wedge true$

$\Longleftarrow$ $\langle$ From the Idompotency of $\wedge$ $\rangle$

true

□

A.6 Detailed Proof of Lemma 6.2.1

Proof of Lemma 6.2.1 (a).

S is a segment

$\Longleftrightarrow$ $\langle$ Definition 3 $\rangle$

$(\forall\, r, r' \ \mid \ r \in S \wedge r' \in (R - S) \ \cdot \ w_{\mathbf{P}}(\mathtt{gcd}(p(r), p(r'))) \leq w_{\mathbf{P}}(\mathtt{gcd}(\, r \ \mid \ r \in S \ \cdot \ p(r)\,))\,)$

$\Longleftrightarrow$ $\langle$ S is a set consisting of resources having co-prime policies, gcd of co-prime policies $= 1_{\mathcal{F}}$ $\rangle$

$(\forall\, r, r' \mid r \in S \wedge r' \in (R - S) \cdot w_{\mathbf{P}}(\texttt{gcd}(p(r), p(r'))) \leq w_{\mathbf{P}}(1_{\mathcal{F}}))$

$\Longleftarrow$ $\langle$ From hypothesis $p(r') = 1_{\mathcal{F}}$ $\rangle$

$(\forall\, r, r' \mid r \in S \wedge r' \in (R - S) \cdot w_{\mathbf{P}}(\texttt{gcd}(p(r), 1_{\mathcal{F}})) \leq w_{\mathbf{P}}(1_{\mathcal{F}}))$

$\Longleftrightarrow$ $\langle$ Because $(\forall\, a \mid \cdot \quad \texttt{gcd}(a, 1_{\mathcal{F}}) = 1_{\mathcal{F}})$ $\rangle$

$(\forall\, r, r' \mid r \in S \wedge r' \in (R - S) \cdot w_{\mathbf{P}}(1_{\mathcal{F}}) \leq w_{\mathbf{P}}(1_{\mathcal{F}}))$

$\Longleftarrow$ $\langle$ reflexivity of $\leq$ $\rangle$

true

□

Proof of Lemma 6.2.1 (b).

S is a segment

$\Longleftrightarrow$ $\langle$ Definition 3 $\rangle$

$(\forall\, r, r' \mid r \in S \wedge r' \in (R - S) \cdot$

$w_{\mathbf{P}}(\texttt{gcd}(p(r), p(r'))) \leq w_{\mathbf{P}}(\texttt{gcd}(\, r \mid r \in S \cdot p(r))))$

$\Longleftarrow$ $\langle$ From hypothesis: $(\forall\, r, r' \mid r \in S \wedge r' \in (R - S) \cdot \texttt{gcd}(p(r), p(r')) = 1_{\mathcal{F}})$ $\rangle$

$(\forall\, r, r' \mid r \in S \wedge r' \in (R - S) \cdot w_{\mathbf{P}}(1_{\mathcal{F}}) \leq w_{\mathbf{P}}(\texttt{gcd}(\, r \mid r \in S \cdot p(r))))$

$\Longleftarrow$ $\langle$ $r \in S$ and $p(r) = 1_{\mathcal{F}}$, which makes $\texttt{gcd}(\, r \mid r \in S \cdot p(r)) = 1_{\mathcal{F}}$ $\rangle$

$(\forall\, r, r' \mid r \in S \wedge r' \in (R - S) \cdot w_{\mathbf{P}}(1_{\mathcal{F}}) \leq w_{\mathbf{P}}(1_{\mathcal{F}}))$

$\Longleftarrow$ $\langle$ reflexivity of $\leq$ $\rangle$

true

□

Appendix B

Prototype Tool

This appendix presents a prototype that is implemented to automate and validate the results and theories presented in Chapters 4 and 5. Precisely, given a network topology with the leaf nodes assigned policies, the prototype aims at calculating the policies of the internal nodes according to Proposition 5.1.3 and maintain the DD implementation. Section B.1 discusses the high-level design of the prototype. Section B.2 presents the detailed design of the prototype. Section B.3 presents a simple use case of using the tool.

B.1 High-Level Design

The main functionality of the prototype can be explained as follows: given a network topology with leaf nodes assigned policies by the administrator, the tool aims at determining the policies of the internal nodes (i.e., firewalls) that provide resources (i.e., leaf nodes) with extra protection and without disturbing their reachability and availability. Moreover, in case of any change in topology or policy update, the tool readjusts the policies of the internal nodes accordingly. The tool runs on a central unit and agents running on every node in the network. The prototype tool includes two major elements: An analysis element (*Analyzer*) and a broker element (*Broker*) as shown in Figure B.1. These two components reside in the

central unit.

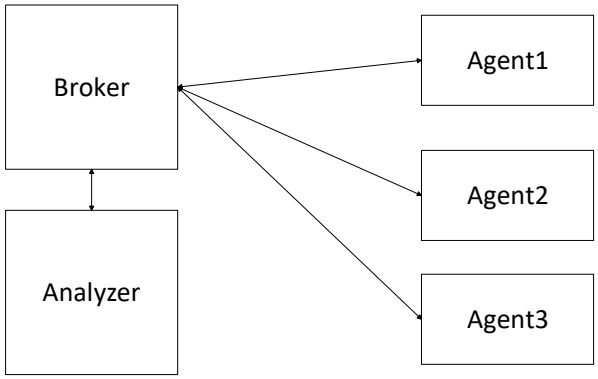


Figure B.1: High level view of the tool

The broker is responsible for managing the different components of the prototype tool. It communicates with agents running on nodes. Through this communication channel, it checks the status of the node, its policy, and the communication. Once an agent is up on a node it registers itself along with its policy to the broker. The agent of a leaf node monitors the policy changes. If the agent observers any change in the policy, it notifies the broker. The broker then updates the registered policy and recalculates the policies of the affected internal nodes and transmits the newly calculated policies through the agents running on those nodes. When an agent of an internal node registers to the broker, the broker calculates its policy from the policies of the nodes attached to it. If a node goes down or the connection gets lost, the broker recalculates and reassigns policies of affected nodes. Moreover, the broker manages the calculation and generation of policies by calling the analyzer. The broker is implemented using Java Remote Method Invocation (RMI) [Co.]. Java RMI is a Java API that performs remote method invocation equivalent to Remote Procedure Call (RPC) for distributed Java applications. It enables a Java program running

on one virtual machine to invoke a method of another Java program on a different virtual machine.

The other component of the prototype, the analyzer, is responsible for checking policy consistency and calculating `gcds` of policies. Policies are fed into the analyzer in the form of tabular expressions presented in Chapter 3.

These tables can be encoded using a markup language. The one used in our prototype is called TabML [KWS03]. Moreover, for the verifications and calculations the analyzer uses Prototype Verification System (PVS) [SRI02]. Essentially, the analyzer is a modified version of the SCENATOR tool [KWS03] which is developed for the verification of requirement scenarios and is built using C language. The formalism SCENATOR uses to verify requirement scenarios similar to the one used in this thesis to analyze policies. The main addition to the prototype to SCENATOR is the development of modules to verify policies consistency and a module to calculate GCDs.

TabML [KWS03] is a markup language used to represent tabular expressions. Figure B.2 shows the TabML representation of rules $3-5$ in the policy shown in Figure 3.3.

A TabML file is divided into two main parts. The first part has the general information of the table such as table name, focus, log, variables, etc. The second part shows the actual table including headers and grids. For more details on TabML, the reader is referred to [Wu01, KWS03].

PVS [OSRSC01] is a theorem prover used by the analyzer. The analyzer generates conjunctures and uses PVS to prove them. It has been shown in [RS93] that PVS is an appropriate theorem prover to perform formalisms similar to the ones used by the analyzer. It has been chosen for several reasons including that theorems can be formalized easily in PVS and saved to text files. Moreover, PVS can run in a patch mode which makes it invisible for the user. All it wants to run in a patch mode is the conjunctures and proof strategies.

```
<MathTable, (1,3)>
<name> POLICY TABLE </name>
<focus>policy table</focus>
<source>Verification and Integration Using Tabular Expressions</source>
<log>Wed Feb 13 10:07:00 2019</log>
<version> 1 </version>
<input>
<tabvar>
IPSEG, TYPE = {x:int | x >= 0 &  x < 255}<varmeaning> IP segment type</varmeaning>
</tabvar>
<tabvar>
IPSN, TYPE = {y:int| y = 8 OR y = 16 OR y = 24 OR y = 32}<varmeaning> IP subnet type</varmeaning>
</tabvar>
<tabvar>IPT, TYPE= [IPSEG,IPSEG,IPSEG, IPSEG, IPSN] <varmeaning> IP Type</varmeaning></tabvar>
<tabvar>s, VAR IPT  <varmeaning> source IP variable</varmeaning> </tabvar>
<tabvar>dport, VAR int <varmeaning> dport variable</varmeaning> </tabvar>
<tabvar>TE, TYPE={tcp, icmp, udp, any} <varmeaning> protocol type </varmeaning> </tabvar>
<tabvar> p, VAR TE <varmeaning> protocol variable </varmeaning> </tabvar>
</input>
<output>
<tabvar>J, TYPE={ACCEPT, REJECT} <varmeaning> action type </varmeaning> </tabvar>
<tabvar>j, VAR J  <varmeaning> action variable </varmeaning> </tabvar>
</output>
<author> Mohammed Alabbad </author>
<slice,8>
<resitrcition> /true </restriction>
<header,(1,3)>
<headername> H1 </headername>
<line>
<cell>s = (192,168,1,0,24) </cell>
<cell>s = (192,168,2,0,24) </cell>
<cell>s = (192,168,3,0,24) </cell>
        </line>
</header>
<header, (1,1)>
<headername> H2 </headername>
<line>
<cell> p = any</cell>
</line>
</header>
<grid,(1,3)>
<line>
<cell>j = REJECT</cell>
<cell>j = ACCEPT</cell>
<cell>j = REJECT</cell>
</line>
</grid>
</slice>
</mathTable>
```

Figure B.2: Part of the Engineering workstation policy in TabML

B.2 Detailed Design

This section shows the detailed design of the main components of the prototype tool. The broker and analyzer modules and functionalities are discussed below.

B.2.1 Broker

The design of the broker is based on the *observer* pattern. In the observer pattern, an object (i.e., broker) maintains a list of observers (e.g., agents running on nodes) and notifies them of any changes (e.g., assigned policies) by calling one of their methods.

The interface of the broker has two taps. Figure B.3 shows the first tap where the user enters the topology of the network or imports it from an existing file. The broker uses this topology to know the graph of the network and the placement of nodes. Figure B.4 shows the main tap of the broker where the user gets the broker activated and monitors the status of the system. It also allows the user to read the policies of the nodes he wishes to check.

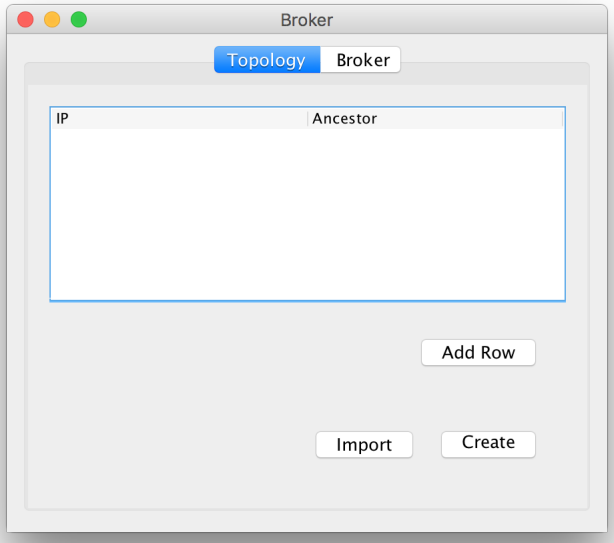


Figure B.3: Broker interface-topology tap

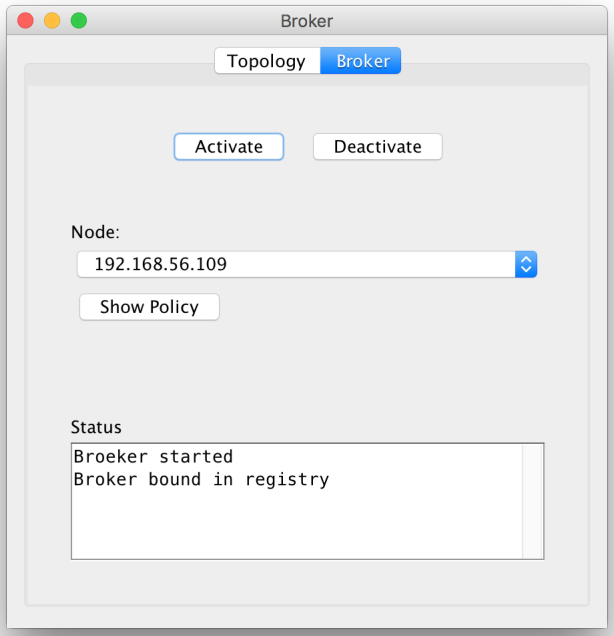


Figure B.4: Broker Interface-main tap

The class diagram of the broker is shown in Figure B.5. The broker is written in Java and contains multiple classes as follows:

- **The broker class (RmiBroker.java)**

  Service: It manages the whole system.

  Secret: It is the main class that initializes the system. It creates topology, status, and policy holder objects to store the topology of the network, the status of the nodes, and the policies of the nodes, respectively. Agents register themselves to the broker by calling the *register* method and update their policies if needed by calling the *update* method. The broker sends a class (session) to each agent to maintain the connection. The broker internally changes the status of the agent and store its policy. It calculates the GCD, if needed, by sending two policies to the analyzer where the GCD of them

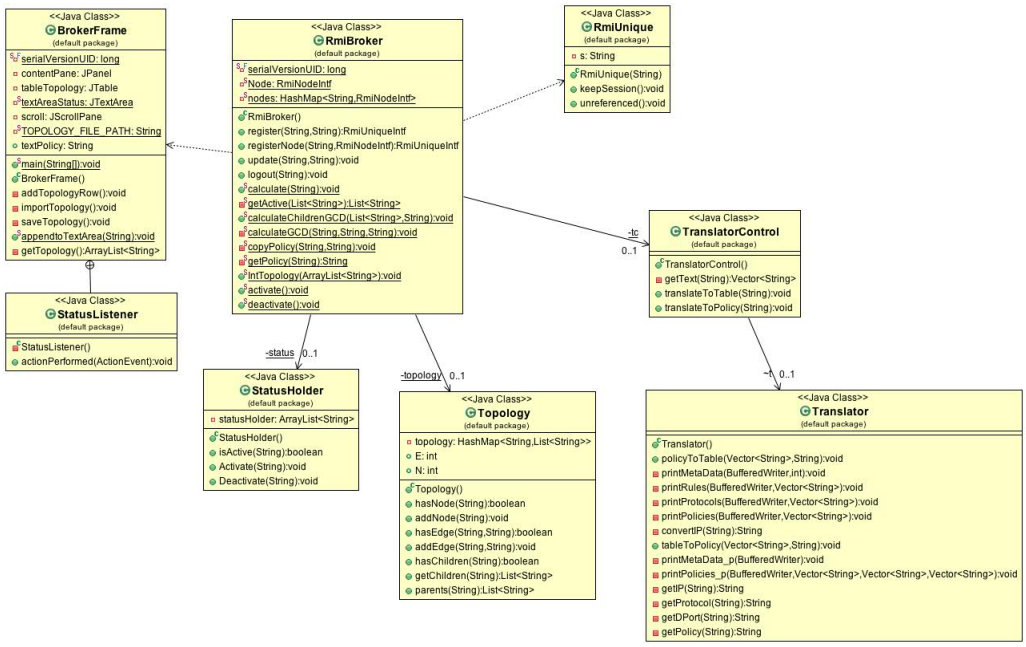


Figure B.5: Class diagram for the broker element

returned to the broker. The broker then updates the policies of the affected nodes.

- **The agent class (RmiAgent.java)**

  Service: It runs on each node and communicates to the broker.

  Secret: It is a class to be located in every node. Once it runs, it registers to the broker by sending its IP and policy. Afterward, it keeps checking the policy of the node periodically, if there is a change in the policy it notifies the broker and sends the new policy.

- **The session class (RmiUnique.java)**

  Service: It is a class that keeps tracks of the status of connection for every agent.

  Secret: It is sent by the broker to the agent to keep track of the connection. Once the connection is lost the session class notifies the broker of the lost connection.

- **Interface classes (RmiBrokerIntf.java, RmiUniqueIntf.java)**

  Service: Classes that makes methods visible to be invoked by the agent.

Secret: They are interface classes that hold the method definitions that can be invoked by the agent class.

- **Translator classes (TranslatorControl.java, Translator.java**

  Service: These are classes that translate from the language of iptables to TabMl and vice versa.

  Secret: It contains the methods to translate from iptables to TabMl and vice versa.

- **Status holder class (StatusHolder.java)**

  Service: The broker uses this class to store the status of the different nodes in the network.

  Secret: It is a class that the broker class creates an object from to store the status of agents.

- **Topology class (Topology.java)**

  Service: It is a class to store the topology of the network.

  Secret: The broker uses an object of this class to store the topology of the networks. Which is entered by the user through the interface.

- **Policy holder class (PolicyHolder.java)**

  Service: This class is to store the policies of the nodes in the network.

  Secret: This class stores the policies of the registered nodes and the calculated ones for easier reach by the broker.

- **Interface class (BrokerFrame.java)**

  Service: The interface class for the broker.

  Secret: It is a class that contains the components that control the interface of the broker.

Figures B.6, B.7, B.8 present the sequence diagrams for the agent register, agent update, and loss of connection respectively.

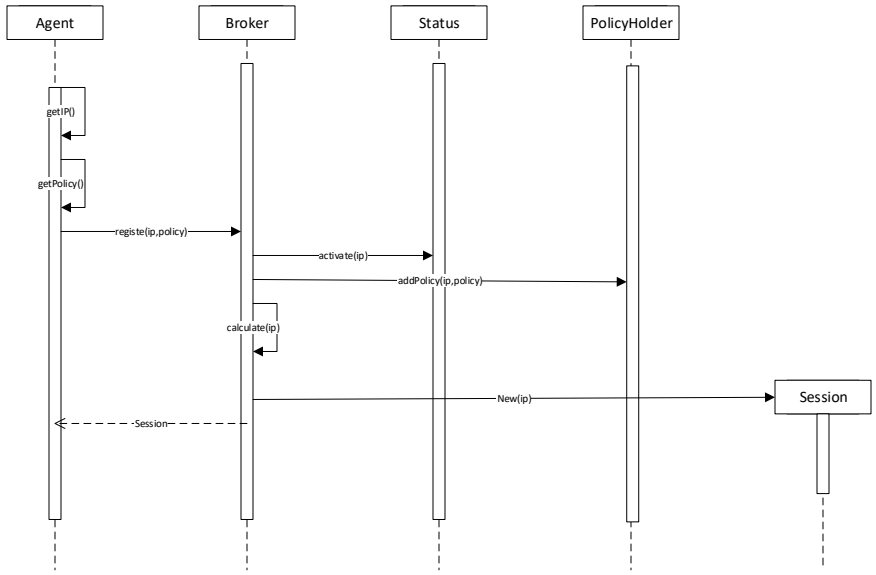


Figure B.6: Sequence diagram for agent registering

B.2.2 Analyzer

The *Analyzer* is responsible for all the calculations needed to generate policies to be assigned to nodes

according to the scheme given in Propositions 5.1.3 based on the assigned policies to leaf resources. Since the analyzer is a modified version of the SCENATOR tool, this section focuses on the added modules to the tool. For the details of the modules that are part of SENATOR, we refer the reader to [Wu01, KWS03].

The parts that have been added or updated to fit the requirement for calculating the `gcd` of two policies are the following:

- **Table GCD module (TableGCD.c)**

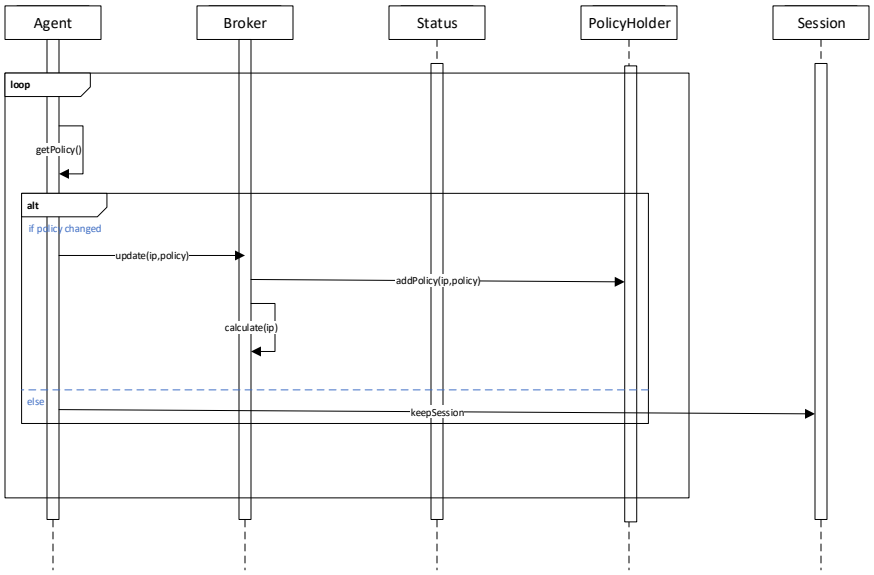


Figure B.7: Sequence diagram for agent update

Service: It calculates the `gcd` of two policy tables and generates a new table.

Secret: Takes two policies written in TabML as arguments. Then it checks if each one of them is consistent. If any of the policies are inconsistent, it notifies the user and generates a new policy with the inconsistent elements marked. Otherwise, it resumes to calculate their `gcd` and generates the output file.

- **Table Consistency check module (cons.c)**

  Service: It checks the consistency of a policy table.

  Secret: Takes a policy table in TabML language and generates PVS conjunctures to check the consistency of the table. If it is inconsistent, it notifies the user and generates a new table with inconsistent cells marked.

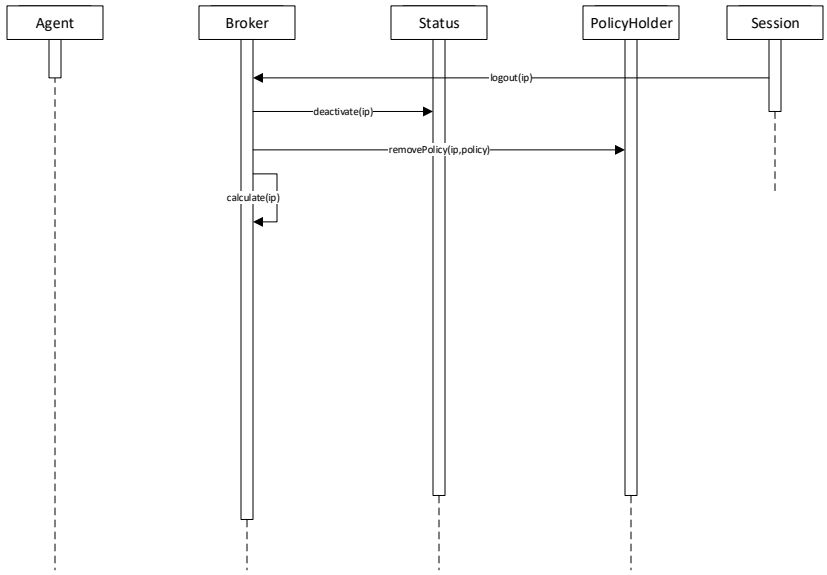


Figure B.8: Sequence diagram for agent logout

In Section 3.2.1, we have presented the procedure ComputeDemonicJoin which is used to calculate the demonic join of tabular expression representations of relations. After the modification of the SCENATOR tool, the integrated analyzer executes the procedure ComputeDemonicJoin. Where the GCD of two tabular expressions is essentially the demonic join of them.

In this procedure, to calculate the GCD of tow policies P_1 and P_2 represented in TabML, the headers of P_1 and P_2 are first transformed to have identical headers which would be the headers of the of the GCD. Then, the grid values of the GCD are determined as follows: the analyzer loops through the grids in the transformed P_1' and P_2'. If a grid cell value is not $false$ in both tables, then the cell value in the GCD is the union of the corresponding cells in P_1' and P_2', otherwise it is set to $false$. That is; if the identical grid in the two tables

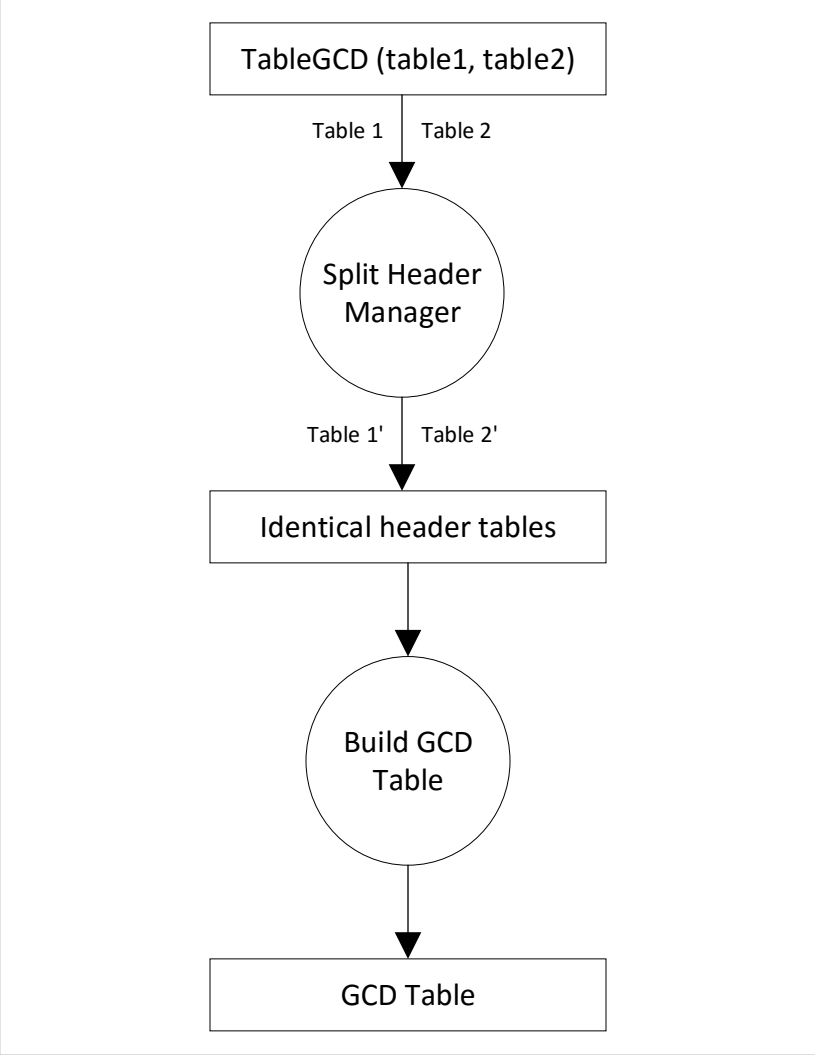


Figure B.9: Data flow diagram for the analyzer element

has the same value then the matching grid in the GCD table will have the same value, if the grids have different values then the grid cell in the GCD table will be the $\vee$ of those values, if any of the grid cells have $false$ then the value will be $false$ because in this case, it is outside the domain of one table therefore not part of the demonic join.

Comparing the headers, splitting them, and creating new tables with modified headers are all part of the SCENATOR tool. SCENATOR is written in C and it reads the policies written in TabML format. From the TabML files, headers are extracted, PVS conjunctures

are generated to compare table headers where PVS run in the patch mode. The results are then read and based on them the decision to split headers. The headers are split again based on running PVS conjunctures. After the tables are transformed and have identical headers, the values for the cells are compared as discussed above.
There are also improvements in the PVS theories to check consistency and calculate GCD of tables.

- **cons.pvs** It contains a function *intersect* that is used by conjunctures created by consistency check to check the consistency of policies. It is also used to check if headers intersect before splitting tables in the step of table transformation before calculating `gcd` of table policies.

- **equalr.pvs** It is a theory that contains a function *equalrule* that used in the conjunctures created by the analyzer to compare the headers of the two tables. if two headers are equal, they are left otherwise we proceed to check the intersection. If they are not intersected, then the header in table T_1 that is not equal or intersect with any other header in table T_2 is added to $T2$.

- **attr.pvs** it contains functions used to determine which attribute to split at for intersecting rules.

- **attrtype.pvs** contains functions used to determine the type of attribute intersection to aid in splitting rules in the deferent tables.

B.3 A Case Study

As a case study, the prototype is executed to calculate the GCD of the Lines $3-5$ of the policies of the engineering workstations and the finance workstations presented in Chapter 3. Figures B.2 and B.10 present the engineering workstation policy and finance workstation policy after translation to TabML, respectively. Figure B.11 represent the GCD in TabML.

```
<MathTable, (1,3)>
<name> POLICY TABLE </name>
<focus>policy table</focus>
<source>Verification and Integration Using Tabular Expressions</source>
<log>Wed Feb 13 10:07:00 2019</log>
<version> 1 </version>
<input>
<tabvar>
IPSEG, TYPE = {x:int | x >= 0 &  x < 255}<varmeaning> IP segment type</varmeaning>
</tabvar>
<tabvar>
IPSN, TYPE = {y:int| y = 8 OR y = 16 OR y = 24 OR y = 32}<varmeaning> IP subnet type</varmeaning>
</tabvar>
<tabvar>IPT, TYPE= [IPSEG,IPSEG,IPSEG, IPSEG, IPSN] <varmeaning> IP Type</varmeaning></tabvar>
<tabvar>s, VAR IPT  <varmeaning> source IP variable</varmeaning> </tabvar>
<tabvar>dport, VAR int <varmeaning> dport variable</varmeaning> </tabvar>
<tabvar>TE, TYPE={tcp, icmp, udp, any} <varmeaning> protocol type </varmeaning> </tabvar>
<tabvar> p, VAR TE <varmeaning> protocol variable </varmeaning> </tabvar>
</input>
<output>
<tabvar>J, TYPE={ACCEPT, REJECT} <varmeaning> action type </varmeaning> </tabvar>
<tabvar>j, VAR J  <varmeaning> action variable </varmeaning> </tabvar>
</output>
<author> Mohammed Alabbad </author>
<slice,8>
<resitrcition> /true </restriction>
<header,(1,3)>
<headername> H1 </headername>
<line>
<cell>s = (192,168,1,0,24) </cell>
<cell>s = (192,168,2,0,24) </cell>
<cell>s = (192,168,3,0,24) </cell>
        </line>
</header>
<header, (1,1)>
<headername> H2 </headername>
<line>
<cell> p = any</cell>
</line>
</header>
<grid,(1,3)>
<line>
<cell>j = REJECT</cell>
<cell>j = REJECT</cell>
<cell>j = ACCEPT</cell>
</line>
</grid>
</slice>
</mathTable>
```

Figure B.10: Part of the Finance workstation policy in TabML

```
<MathTable, (1,3)>
<name> POLICY TABLE </name>
<focus>policy table</focus>
<source>Verification and Integration Using Tabular Expressions</source>
<log>Wed Feb 13 10:07:00 2019</log>
<version> 1 </version>
<input>
<tabvar>
IPSEG, TYPE = {x:int | x >= 0 &  x < 255}<varmeaning> IP segment type</varmeaning>
</tabvar>
<tabvar>
IPSN, TYPE = {y:int| y = 8 OR y = 16 OR y = 24 OR y = 32}<varmeaning> IP subnet type</varmeaning>
</tabvar>
<tabvar>IPT, TYPE= [IPSEG,IPSEG,IPSEG, IPSEG, IPSN] <varmeaning> IP Type</varmeaning></tabvar>
<tabvar>s, VAR IPT  <varmeaning> source IP variable</varmeaning> </tabvar>
<tabvar>dport, VAR int <varmeaning> dport variable</varmeaning> </tabvar>
<tabvar>TE, TYPE={tcp, icmp, udp, any} <varmeaning> protocol type </varmeaning> </tabvar>
<tabvar> p, VAR TE <varmeaning> protocol variable </varmeaning> </tabvar>
</input>
<output>
<tabvar>J, TYPE={ACCEPT, REJECT} <varmeaning> action type </varmeaning> </tabvar>
<tabvar>j, VAR J  <varmeaning> action variable </varmeaning> </tabvar>
</output>
<author> Mohammed Alabbad </author>
<slice,8>
<resitrcition> /true </restriction>
<header,(1,3)>
<headername> H1 </headername>
<line>
<cell>s = (192,168,1,0,24) </cell>
<cell>s = (192,168,2,0,24) </cell>
<cell>s = (192,168,3,0,24) </cell>
        </line>
</header>
<header, (1,1)>
<headername> H2 </headername>
<line>
<cell> p = any</cell>
</line>
</header>
<grid,(1,3)>
<line>
<cell>j = REJECT</cell>
<cell>j = ACCEPT OR REJECT</cell>
<cell>j = ACCEPT OR REJECT</cell>
</line>
</grid>
</slice>
</mathTable>
```

Figure B.11: GCD policy in TabML

Appendix C

Example Policies in Tabular Expressions

		$s = 192.168.1.0/24$		$s = 192.168.2.0/24$		$s = 192.168.3.0/24$		$s = 192.168.4.0/24$		$s \notin \{192.168.1.0/24, 192.168.2.0/24, 192.168.3.0/24, 192.168.4.0/24\}$	
		$p = \texttt{TCP}$	$p \neq \texttt{TCP}$	$p = \texttt{TCP}$	$p \neq \texttt{TCP}$	$p = \texttt{TCP}$	$p \neq \texttt{TCP}$	$p = \texttt{TCP}$	$p \neq \texttt{TCP}$	$p = \texttt{TCP}$	$p \neq \texttt{TCP}$
$st \in \{\texttt{RELATED}, \texttt{ESTABLISHED}\}$	$dport \in \{80, 25\}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$
	$dport \notin \{80, 25\}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$
$st = \texttt{INVALID}$	$dport \in \{80, 25\}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$
	$dport \notin \{80, 25\}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$
$st = \texttt{NEW}$	$dport \in \{80, 25\}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{DROP}$
	$dport \notin \{80, 25\}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$

Table C.1: Web and Email servers policy represented as a tabular expression

		$s = 192.168.1.0/24$		$s = 192.168.2.0/24$		$s = 192.168.3.0/24$		$s = 192.168.4.0/24$		$s \notin \{192.168.1.0/24, 192.168.2.0/24, 192.168.3.0/24, 192.168.4.0/24\}$	
		$p = \texttt{TCP}$	$p \neq \texttt{TCP}$	$p = \texttt{TCP}$	$p \neq \texttt{TCP}$	$p = \texttt{TCP}$	$p \neq \texttt{TCP}$	$p = \texttt{TCP}$	$p \neq \texttt{TCP}$	$p = \texttt{TCP}$	$p \neq \texttt{TCP}$
$st \in \{\texttt{RELATED}, \texttt{ESTABLISHED}\}$	$dport \in \{80, 25\}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$
	$dport \notin \{80, 25\}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$
$st = \texttt{INVALID}$	$dport \in \{80, 25\}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$
	$dport \notin \{80, 25\}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$
$st = \texttt{NEW}$	$dport \in \{80, 25\}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{REJECT}$	$a' = \texttt{REJECT}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$
	$dport \notin \{80, 25\}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{REJECT}$	$a' = \texttt{REJECT}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$

Table C.2: File server policy represented as a tabular expression

		$s = 192.168.1.0/24$		$s = 192.168.2.0/24$		$s = 192.168.3.0/24$		$s = 192.168.4.0/24$		$s \notin \{192.168.1.0/24, 192.168.2.0/24, 192.168.3.0/24, 192.168.4.0/24\}$	
		$p = \texttt{TCP}$	$p \neq \texttt{TCP}$	$p = \texttt{TCP}$	$p \neq \texttt{TCP}$	$p = \texttt{TCP}$	$p \neq \texttt{TCP}$	$p = \texttt{TCP}$	$p \neq \texttt{TCP}$	$p = \texttt{TCP}$	$p \neq \texttt{TCP}$
$st \in \{\texttt{RELATED}, \texttt{ESTABLISHED}\}$	$dport \in \{80, 25\}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$
	$dport \notin \{80, 25\}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$
$st = \texttt{INVALID}$	$dport \in \{80, 25\}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$
	$dport \notin \{80, 25\}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$
$st = \texttt{NEW}$	$dport \in \{80, 25\}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{REJECT}$	$a' = \texttt{REJECT}$	$a' = \texttt{REJECT}$	$a' = \texttt{REJECT}$	$a' = \texttt{REJECT}$	$a' = \texttt{REJECT}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$
	$dport \notin \{80, 25\}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{REJECT}$	$a' = \texttt{REJECT}$	$a' = \texttt{REJECT}$	$a' = \texttt{REJECT}$	$a' = \texttt{REJECT}$	$a' = \texttt{REJECT}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$

Table C.3: Finance workstations and database policy represented as a tabular expression

		$s = 192.168.1.0/24$		$s = 192.168.2.0/24$		$s = 192.168.3.0/24$		$s = 192.168.4.0/24$		$s \notin \{192.168.1.0/24, 192.168.2.0/24, 192.168.3.0/24, 192.168.4.0/24\}$	
		$p = \texttt{TCP}$	$p \neq \texttt{TCP}$	$p = \texttt{TCP}$	$p \neq \texttt{TCP}$	$p = \texttt{TCP}$	$p \neq \texttt{TCP}$	$p = \texttt{TCP}$	$p \neq \texttt{TCP}$	$p = \texttt{TCP}$	$p \neq \texttt{TCP}$
$st \in \{\texttt{RELATED}, \texttt{ESTABLISHED}\}$	$dport \in \{80, 25\}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$
	$dport \notin \{80, 25\}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$
$st = \texttt{INVALID}$	$dport \in \{80, 25\}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$
	$dport \notin \{80, 25\}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$
$st = \texttt{NEW}$	$dport \in \{80, 25\}$	$a' = \texttt{REJECT}$	$a' = \texttt{REJECT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{REJECT}$	$a' = \texttt{REJECT}$	$a' = \texttt{REJECT}$	$a' = \texttt{REJECT}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$
	$dport \notin \{80, 25\}$	$a' = \texttt{REJECT}$	$a' = \texttt{REJECT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{REJECT}$	$a' = \texttt{REJECT}$	$a' = \texttt{REJECT}$	$a' = \texttt{REJECT}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$

Table C.4: Engineering workstations policy represented as a tabular expression

Appendix D

GCD Policies in Tabular Expressions

		$s = 192.168.1.0/24$		$s = 192.168.2.0/24$		$s = 192.168.3.0/24$		$s = 192.168.4.0/24$		$s \notin \{192.168.1.0/24, 192.168.2.0/24, 192.168.3.0/24, 192.168.4.0/24\}$	
		$p = \texttt{TCP}$	$p \neq \texttt{TCP}$	$p = \texttt{TCP}$	$p \neq \texttt{TCP}$	$p = \texttt{TCP}$	$p \neq \texttt{TCP}$	$p = \texttt{TCP}$	$p \neq \texttt{TCP}$	$p = \texttt{TCP}$	$p \neq \texttt{TCP}$
$st \in \{\texttt{RELATED}, \texttt{ESTABLISHED}\}$	$dport \in \{80, 25\}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$
	$dport \notin \{80, 25\}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$
$st = \texttt{INVALID}$	$dport \in \{80, 25\}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$
	$dport \notin \{80, 25\}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$
$st = \texttt{NEW}$	$dport \in \{80, 25\}$	$a' = \texttt{ACCEPT} \vee a' = \texttt{REJECT}$	$a' = \texttt{ACCEPT} \vee a' = \texttt{REJECT}$	$a' = \texttt{ACCEPT} \vee a' = \texttt{REJECT}$	$a' = \texttt{ACCEPT} \vee a' = \texttt{REJECT}$	$a' = \texttt{ACCEPT} \vee a' = \texttt{REJECT}$	$a' = \texttt{ACCEPT} \vee a' = \texttt{REJECT}$	$a' = \texttt{ACCEPT} \vee a' = \texttt{REJECT}$	$a' = \texttt{ACCEPT} \vee a' = \texttt{REJECT}$	$a' = \texttt{ACCEPT} \vee a' = \texttt{DROP}$	$a' = \texttt{DROP}$
	$dport \notin \{80, 25\}$	$a' = \texttt{ACCEPT} \vee a' = \texttt{REJECT}$	$a' = \texttt{ACCEPT} \vee a' = \texttt{REJECT}$	$a' = \texttt{ACCEPT} \vee a' = \texttt{REJECT}$	$a' = \texttt{ACCEPT} \vee a' = \texttt{REJECT}$	$a' = \texttt{ACCEPT} \vee a' = \texttt{REJECT}$	$a' = \texttt{ACCEPT} \vee a' = \texttt{REJECT}$	$a' = \texttt{ACCEPT} \vee a' = \texttt{REJECT}$	$a' = \texttt{ACCEPT} \vee a' = \texttt{REJECT}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$

Table D.5: GCD1 represented as a tabular expression

		$s = 192.168.1.0/24$		$s = 192.168.2.0/24$		$s = 192.168.3.0/24$		$s = 192.168.4.0/24$		$s \notin \{192.168.1.0/24, 192.168.2.0/24, 192.168.3.0/24, 192.168.4.0/24\}$	
		$p = \texttt{TCP}$	$p \neq \texttt{TCP}$	$p = \texttt{TCP}$	$p \neq \texttt{TCP}$	$p = \texttt{TCP}$	$p \neq \texttt{TCP}$	$p = \texttt{TCP}$	$p \neq \texttt{TCP}$	$p = \texttt{TCP}$	$p \neq \texttt{TCP}$
$st \in \{\texttt{RELATED}, \texttt{ESTABLISHED}\}$	$dport \in \{80, 25\}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$
	$dport \notin \{80, 25\}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$	$a' = \texttt{ACCEPT}$
$st = \texttt{INVALID}$	$dport \in \{80, 25\}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$
	$dport \notin \{80, 25\}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$
$st = \texttt{NEW}$	$dport \in \{80, 25\}$	$a' = \texttt{ACCEPT} \vee a' = \texttt{REJECT}$	$a' = \texttt{ACCEPT} \vee a' = \texttt{REJECT}$	$a' = \texttt{ACCEPT} \vee a' = \texttt{REJECT}$	$a' = \texttt{ACCEPT} \vee a' = \texttt{REJECT}$	$a' = \texttt{ACCEPT} \vee a' = \texttt{REJECT}$	$a' = \texttt{ACCEPT} \vee a' = \texttt{REJECT}$	$a' = \texttt{REJECT}$	$a' = \texttt{REJECT}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$
	$dport \notin \{80, 25\}$	$a' = \texttt{ACCEPT} \vee a' = \texttt{REJECT}$	$a' = \texttt{ACCEPT} \vee a' = \texttt{REJECT}$	$a' = \texttt{ACCEPT} \vee a' = \texttt{REJECT}$	$a' = \texttt{ACCEPT} \vee a' = \texttt{REJECT}$	$a' = \texttt{ACCEPT} \vee a' = \texttt{REJECT}$	$a' = \texttt{ACCEPT} \vee a' = \texttt{REJECT}$	$a' = \texttt{REJECT}$	$a' = \texttt{REJECT}$	$a' = \texttt{DROP}$	$a' = \texttt{DROP}$

Table D.6: GCD2 represented as a tabular expression

		$s = 192.168.1.0/24$		$s = 192.168.2.0/24$		$s = 192.168.3.0/24$		$s = 192.168.4.0/24$		$s \notin \{192.168.1.0/24, 192.168.2.0/24, 192.168.3.0/24, 192.168.4.0/24\}$	
		$p =$ TCP	$p \neq$ TCP	$p =$ TCP	$p \neq$ TCP	$p =$ TCP	$p \neq$ TCP	$p =$ TCP	$p \neq$ TCP	$p =$ TCP	$p \neq$ TCP
$st \in \{$RELATED, ESTABLISHED$\}$	$dport \in \{80, 25\}$	$a' =$ ACCEPT	$a' =$ ACCEPT	$a' =$ ACCEPT	$a' =$ ACCEPT	$a' =$ ACCEPT	$a' =$ ACCEPT	$a' =$ ACCEPT	$a' =$ ACCEPT	$a' =$ ACCEPT	$a' =$ ACCEPT
	$dport \notin \{80, 25\}$	$a' =$ ACCEPT	$a' =$ ACCEPT	$a' =$ ACCEPT	$a' =$ ACCEPT	$a' =$ ACCEPT	$a' =$ ACCEPT	$a' =$ ACCEPT	$a' =$ ACCEPT	$a' =$ ACCEPT	$a' =$ ACCEPT
$st =$ INVALID	$dport \in \{80, 25\}$	$a' =$ DROP	$a' =$ DROP	$a' =$ DROP	$a' =$ DROP	$a' =$ DROP	$a' =$ DROP	$a' =$ DROP	$a' =$ DROP	$a' =$ DROP	$a' =$ DROP
	$dport \notin \{80, 25\}$	$a' =$ DROP	$a' =$ DROP	$a' =$ DROP	$a' =$ DROP	$a' =$ DROP	$a' =$ DROP	$a' =$ DROP	$a' =$ DROP	$a' =$ DROP	$a' =$ DROP
$st =$ NEW	$dport \in \{80, 25\}$	$a' =$ ACCEPT $\vee$ $a' =$ REJECT	$a' =$ ACCEPT $\vee$ $a' =$ REJECT	$a' =$ ACCEPT $\vee$ $a' =$ REJECT	$a' =$ ACCEPT $\vee$ $a' =$ REJECT	$a' =$ REJECT	$a' =$ REJECT	$a' =$ REJECT	$a' =$ REJECT	$a' =$ DROP	$a' =$ DROP
	$dport \notin \{80, 25\}$	$a' =$ ACCEPT $\vee$ $a' =$ REJECT	$a' =$ ACCEPT $\vee$ $a' =$ REJECT	$a' =$ ACCEPT $\vee$ $a' =$ REJECT	$a' =$ ACCEPT $\vee$ $a' =$ REJECT	$a' =$ REJECT	$a' =$ REJECT	$a' =$ REJECT	$a' =$ REJECT	$a' =$ DROP	$a' =$ DROP

Table D.7: GCD3 represented as a tabular expression

Appendix E

Example Policies and GCDs in TabML

```
<MathTable, (6,10)>
<name> POLICY TABLE </name>
<focus>policy table</focus>
<source>Verification and Integration Using Tabular Expressions finance</source>
<log>Sun Mar  3 12:12:21 2019
</log>
<version> 1 </version>
<input>
<tabvar>
IPSEG, TYPE = {x:int | x >= 0 &  x < 255}<varmeaning> IP segment type</varmeaning>
</tabvar>
<tabvar>
IPSN, TYPE = {y:int| y = 8 OR y = 16 OR y = 24 OR y = 32}<varmeaning> IP subnet type</varmeaning>
</tabvar>
<tabvar>IPT, TYPE= [IPSEG,IPSEG,IPSEG, IPSEG, IPSN] <varmeaning> IP Type</varmeaning></tabvar>
<tabvar>s, VAR IPT  <varmeaning> source IP variable</varmeaning> </tabvar>
<tabvar>dport, VAR int <varmeaning> dport variable</varmeaning> </tabvar>
<tabvar>TE, TYPE={tcp, icmp, udp, any} <varmeaning> protocol type </varmeaning> </tabvar>
<tabvar> p, VAR TE <varmeaning> protocol variable </varmeaning> </tabvar>
<tabvar>SK, TYPE = {NEW, RELATED, ESTABLISHED, INVALID}<varmeaning> state type </varmeaning> </tabvar>
<tabvar>sk, VAR SK <varmeaning> state variable </varmeaning> </tabvar>
</input>
<output>
<tabvar>J, TYPE={ACCEPT, REJECT, DROP} <varmeaning> action type </varmeaning> </tabvar>
<tabvar>j, VAR J  <varmeaning> action variable </varmeaning> </tabvar>
</output>
<author> Mohammed Alabbad </author>
<slice,8>
<resitrcition> /true </restriction>
<header,(1,10)><headername> H1 </headername>
<line>  <cell>s = (192,168,1,0,24) and p = tcp </cell>
        <cell>s = (192,168,1,0,24) and p /= tcp </cell>
        <cell>s = (192,168,2,0,24) and p = tcp</cell>
        <cell>s = (192,168,2,0,24) and p /= tcp</cell>
        <cell>s = (192,168,3,0,24) and p = tcp</cell>
        <cell>s = (192,168,3,0,24) and p /= tcp</cell>
        <cell>s = (192,168,4,0,24) and p = tcp</cell>
        <cell>s = (192,168,4,0,24) and p /= tcp</cell>
        <cell>s /= (192,168,1,0,24) and s/=(192,168,2,0,24) and s /= (192,168,3,0,24)
            and (s/=192,168,4,0,24) and p = tcp </cell>
        <cell>s /= (192,168,1,0,24) and s/=(192,168,2,0,24) and s /= (192,168,3,0,24)
            and (s/=192,168,4,0,24) and p /= tcp </cell></line></header>
<header, (1,6)><headername> H2 </headername>
<line>  <cell> (sk = RELATED or sk = ESTABLISHED) and (dport = 80 or dport = 25)</cell>
        <cell> (sk = RELATED or sk = ESTABLISHED) and (dport /= 80 and dport /= 25)</cell>
        <cell> (sk = INVALID) and (dport = 80 or dport = 25)</cell>
        <cell> (sk = INVALID) and (dport /= 80 and dport /= 25)</cell>
        <cell> (sk = NEW) and (dport = 80 or dport = 25)</cell>
        <cell> (sk = NEW) and (dport /= 80 and dport /= 25)</cell></line></header>
%Grid cells
</mathTable>
```

Figure E.12: Table information and headers common to all policies in TabML

```
<MathTable, (6,10)>
%Table Information
%headers
<grid,(6,10)>
<line> <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell></line>
<line> <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell></line>
<line> <cell>j = DROP</cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell></line>
<line> <cell>j = DROP</cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell></line>
<line> <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = DROP </cell></line>
<line> <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell></line></grid></slice></mathTable>
```

Figure E.13: Web and Email servers policy in TabML

```
<MathTable, (6,10)>
%Table Information
%headers
<grid,(6,10)>
<line> <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell></line>
<line> <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell></line>
<line> <cell>j = DROP</cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell></line>
<line> <cell>j = DROP</cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell></line>
<line> <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = REJECT </cell>
       <cell>j = REJECT </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell></line>
<line> <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = REJECT </cell>
       <cell>j = REJECT </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell></line></grid></slice></mathTable>
```

Figure E.14: File server policy in TabML

```
<MathTable, (6,10)>
%Table Information
%headers
<grid,(6,10)>
<line> <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell></line>
<line> <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell></line>
<line><cell>j = DROP</cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell></line>
<line> <cell>j = DROP</cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell></line>
<line> <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = REJECT </cell>
       <cell>j = REJECT </cell>
       <cell>j = REJECT </cell>
       <cell>j = REJECT </cell>
       <cell>j = REJECT </cell>
       <cell>j = REJECT </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell></line>
<line> <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = REJECT </cell>
       <cell>j = REJECT </cell>
       <cell>j = REJECT </cell>
       <cell>j = REJECT </cell>
       <cell>j = REJECT </cell>
       <cell>j = REJECT </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell></line></grid></slice></mathTable>
```

Figure E.15: Finance workstation policy in TabML

```
<MathTable, (6,10)>
%Table Information
%headers
<grid,(6,10)>
<line> <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell></line>
<line> <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell></line>
<line> <cell>j = DROP</cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell></line>
<line> <cell>j = DROP</cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell></line>
<line> <cell>j = REJECT </cell>
       <cell>j = REJECT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = REJECT </cell>
       <cell>j = REJECT </cell>
       <cell>j = REJECT </cell>
       <cell>j = REJECT </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell></line>
<line> <cell>j = REJECT </cell>
       <cell>j = REJECT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = ACCEPT </cell>
       <cell>j = REJECT </cell>
       <cell>j = REJECT </cell>
       <cell>j = REJECT </cell>
       <cell>j = REJECT </cell>
       <cell>j = DROP </cell>
       <cell>j = DROP </cell></line></grid></slice></mathTable>
```

Figure E.16: Engineering workstation policy in TabML

```
1  <MathTable, (6,10)>
2  %Table Information
3  %headers
4  <grid,(6,10)>
5  <line> <cell>j = ACCEPT</cell>
6         <cell>j = ACCEPT</cell>
7         <cell>j = ACCEPT</cell>
8         <cell>j = ACCEPT</cell>
9         <cell>j = ACCEPT</cell>
10        <cell>j = ACCEPT</cell>
11        <cell>j = ACCEPT</cell>
12        <cell>j = ACCEPT</cell>
13        <cell>j = ACCEPT</cell>
14        <cell>j = ACCEPT</cell></line>
15 <line> <cell>j = ACCEPT</cell>
16        <cell>j = ACCEPT</cell>
17        <cell>j = ACCEPT</cell>
18        <cell>j = ACCEPT</cell>
19        <cell>j = ACCEPT</cell>
20        <cell>j = ACCEPT</cell>
21        <cell>j = ACCEPT</cell>
22        <cell>j = ACCEPT</cell>
23        <cell>j = ACCEPT</cell>
24        <cell>j = ACCEPT</cell></line>
25 <line> <cell>j = DROP</cell>
26        <cell>j = DROP</cell>
27        <cell>j = DROP</cell>
28        <cell>j = DROP</cell>
29        <cell>j = DROP</cell>
30        <cell>j = DROP</cell>
31        <cell>j = DROP</cell>
32        <cell>j = DROP</cell>
33        <cell>j = DROP</cell>
34        <cell>j = DROP</cell></line>
35 <line> <cell>j = DROP</cell>
36        <cell>j = DROP</cell>
37        <cell>j = DROP</cell>
38        <cell>j = DROP</cell>
39        <cell>j = DROP</cell>
40        <cell>j = DROP</cell>
41        <cell>j = DROP</cell>
42        <cell>j = DROP</cell>
43        <cell>j = DROP</cell>
44        <cell>j = DROP</cell></line>
45 <line> <cell>j = ACCEPT or j = REJECT</cell>
46        <cell>j = ACCEPT or j = REJECT</cell>
47        <cell>j = ACCEPT or j = REJECT</cell>
48        <cell>j = ACCEPT or j = REJECT</cell>
49        <cell>j = ACCEPT or j = REJECT</cell>
50        <cell>j = ACCEPT or j = REJECT</cell>
51    <cell>j = ACCEPT or j = REJECT</cell>
52        <cell>j = ACCEPT or j = REJECT</cell>
53        <cell>j = DROP or j = ACCEPT</cell>
54        <cell>j = DROP</cell></line>
55 <line> <cell>j = ACCEPT or j = REJECT</cell>
56        <cell>j = ACCEPT or j = REJECT</cell>
57        <cell>j = ACCEPT or j = REJECT</cell>
58        <cell>j = ACCEPT or j = REJECT</cell>
59        <cell>j = ACCEPT or j = REJECT</cell>
60        <cell>j = ACCEPT or j = REJECT</cell>
61    <cell>j = ACCEPT or j = REJECT</cell>
62        <cell>j = ACCEPT or j = REJECT</cell>
63        <cell>j = DROP</cell>
64        <cell>j = DROP</cell></line></grid></slice></mathTable>
```

Figure E.17: GCD1 in TabML

```
<MathTable, (6,10)>
%Table Information
%headers
<grid,(6,10)>
<line> <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell></line>
<line> <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell></line>
<line> <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell></line>
<line> <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell></line>
<line> <cell>j = ACCEPT or j = REJECT</cell>
       <cell>j = ACCEPT or j = REJECT</cell>
       <cell>j = ACCEPT or j = REJECT</cell>
       <cell>j = ACCEPT or j = REJECT</cell>
       <cell>j = ACCEPT or j = REJECT</cell>
       <cell>j = ACCEPT or j = REJECT</cell>
    <cell>j = REJECT</cell>
       <cell>j = REJECT</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell></line>
<line> <cell>j = ACCEPT or j = REJECT</cell>
       <cell>j = ACCEPT or j = REJECT</cell>
       <cell>j = ACCEPT or j = REJECT</cell>
       <cell>j = ACCEPT or j = REJECT</cell>
       <cell>j = ACCEPT or j = REJECT</cell>
       <cell>j = ACCEPT or j = REJECT</cell>
    <cell>j = REJECT</cell>
       <cell>j = REJECT</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell></line></grid></slice></mathTable>
```

Figure E.18: GCD2 in TabML

```
<MathTable, (6,10)>
%Table Information
%headers
<grid,(6,10)>
<line> <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell></line>
<line> <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell>
       <cell>j = ACCEPT</cell></line>
<line> <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell></line>
<line> <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell></line>
<line> <cell>j = ACCEPT or j = REJECT</cell>
       <cell>j = ACCEPT or j = REJECT</cell>
       <cell>j = ACCEPT or j = REJECT</cell>
       <cell>j = ACCEPT or j = REJECT</cell>
       <cell>j = REJECT</cell>
       <cell>j = REJECT</cell>
    <cell>j = REJECT</cell>
       <cell>j = REJECT</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell></line>
<line> <cell>j = ACCEPT or j = REJECT</cell>
       <cell>j = ACCEPT or j = REJECT</cell>
       <cell>j = ACCEPT or j = REJECT</cell>
       <cell>j = ACCEPT or j = REJECT</cell>
       <cell>j = REJECT</cell>
       <cell>j = REJECT</cell>
    <cell>j = REJECT</cell>
       <cell>j = REJECT</cell>
       <cell>j = DROP</cell>
       <cell>j = DROP</cell></line></grid></slice></mathTable>
```

Figure E.19: GCD3 in TabML

Appendix F

Relative Atomicity

```
1  -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
2  -A INPUT -m state --state INVALID -j DROP
3  -A INPUT -s 192.168.1.0/24 --sport 0:65535 -d 0.0.0.0/0 --dport 0:65535 -p all -j REJECT
4  -A INPUT -s 192.168.2.0/24 --sport 0:65535 -d 0.0.0.0/0 --dport 0:65535 -p all -j ACCEPT
5  -A INPUT -s 192.168.3.0/24 --sport 0:65535 -d 0.0.0.0/0 --dport 0:65535 -p all -j REJECT
6  -A INPUT -s 192.168.4.0/24 --sport 0:65535 -d 0.0.0.0/0 --dport 0:65535 -p all -j REJECT
7  -A INPUT -s 0.0.0.0/1 --sport 0:65535 -d 0.0.0.0/0 --dport 0:79 -p udp -j DROP
8  -A INPUT -s 0.0.0.0/1 --sport 0:65535 -d 0.0.0.0/0 --dport 0:79 -p icmp -j DROP
9  -A INPUT -s 0.0.0.0/1 --sport 0:65535 -d 0.0.0.0/0 --dport 80:80 -p tcp -j DROP
10 -A INPUT -s 0.0.0.0/1 --sport 0:65535 -d 0.0.0.0/0 --dport 81:65535 -p tcp -j DROP
11 -A INPUT -s 0.0.0.0/1 --sport 0:65535 -d 0.0.0.0/0 --dport 80:80 -p udp -j DROP
12 -A INPUT -s 0.0.0.0/1 --sport 0:65535 -d 0.0.0.0/0 --dport 81:65535 -p udp -j DROP
13 -A INPUT -s 0.0.0.0/1 --sport 0:65535 -d 0.0.0.0/0 --dport 80:80 -p icmp -j DROP
14 -A INPUT -s 0.0.0.0/1 --sport 0:65535 -d 0.0.0.0/0 --dport 81:65535 -p icmp -j DROP
15 -A INPUT -s 128.0.0.0/2 --sport 0:65535 -d 0.0.0.0/0 --dport 0:79 -p udp -j DROP
16 -A INPUT -s 128.0.0.0/2 --sport 0:65535 -d 0.0.0.0/0 --dport 0:79 -p icmp -j DROP
17 -A INPUT -s 128.0.0.0/2 --sport 0:65535 -d 0.0.0.0/0 --dport 80:80 -p tcp -j DROP
18 -A INPUT -s 128.0.0.0/2 --sport 0:65535 -d 0.0.0.0/0 --dport 81:65535 -p tcp -j DROP
19 -A INPUT -s 128.0.0.0/2 --sport 0:65535 -d 0.0.0.0/0 --dport 80:80 -p udp -j DROP
20 -A INPUT -s 128.0.0.0/2 --sport 0:65535 -d 0.0.0.0/0 --dport 81:65535 -p udp -j DROP
21 -A INPUT -s 128.0.0.0/2 --sport 0:65535 -d 0.0.0.0/0 --dport 80:80 -p icmp -j DROP
22 -A INPUT -s 128.0.0.0/2 --sport 0:65535 -d 0.0.0.0/0 --dport 81:65535 -p icmp -j DROP
23 -A INPUT -s 192.0.0.0/9 --sport 0:65535 -d 0.0.0.0/0 --dport 0:79 -p udp -j DROP
24 -A INPUT -s 192.0.0.0/9 --sport 0:65535 -d 0.0.0.0/0 --dport 0:79 -p icmp -j DROP
25 -A INPUT -s 192.0.0.0/9 --sport 0:65535 -d 0.0.0.0/0 --dport 80:80 -p tcp -j DROP
26 ....
```

Figure F.20: Snippet of Engineering workstations relative atomic policy

Bibliography

[ABLP93] Martín Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. *ACM Trans. Program. Lang. Syst.*, 15(4):706–734, September 1993.

[AFGL16] Pedro Adão, Riccardo Focardi, Joshua D. Guttman, and Flaminia L. Luccio. Localizing firewall security policies. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, pages 194–209, June 2016.

[AJN12] Massimiliano Albanese, Sushil Jajodia, and Steven Noel. Time-efficient and cost-effective network hardening using attack graphs. In *42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 1–12, 2012.

[AK21] Mohammed Alabbad and Ridha Khedri. Configuration and governance of dynamic secure SDN. In *The 12th International Conference on Ambient Systems, Networks and Technologies (ANT 2021)*, Procedia Computer Science series, pages 1–8, Warsaw, Poland, March 23 – 26 2021. Elsevier Science.

[ANYG15] Ijaz Ahmad, Suneth Namal, Mika Ylianttila, and Andrei Gurtov. Security in software defined networks: A survey. *IEEE Communications Surveys & Tutorials*, 17(4):2317–2346, 2015.

[AS14] Ehab Al-Shaer. Specification and refinement of a conflict-free distributed firewall configuration language. In *Automated Firewall Analytics*. Springer International Publishing, 2014.

[ASH03] Ehab S. Al-Shaer and Hazem H. Hamed. *Firewall Policy Advisor for Anomaly Discovery and Rule Editing*, pages 17–30. Springer US, Boston, MA, 2003.

[ASH04] Ehab Al-Shaer and Hazem Hamed. Discovery of policy anomalies in distributed firewalls. In *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 4, pages 2605–2616, March 2004.

[ASHBH05] Ehab Al-Shaer, Hazem Hamed, Raouf Boutaba, and Masum Hasan. Conflict classification and analysis of distributed firewall policies. *IEEE Journal on Selected Areas in Communications*, 23(10):2069–2084, Oct 2005.

[ASM] Ali Al-Shabibi and Murphy McCauley. POX Controller. Available: `https://noxrepo.github.io/pox-doc/html` (Accessed: Mar 23, 2020).

[BBCC14] Giuseppe Bianchi, Marco Bonola, Antonio Capone, and Carmelo Cascone. OpenState: Programming platform-independent stateful openflow applications inside the switch. *SIGCOMM Comput. Commun. Rev.*, 44(2):44–51, April 2014.

[BBF01] Elisa Bertino, Piero Andrea Bonatti, and Elena Ferrari. TRBAC: A temporal role-based access control model. *ACM Trans. Inf. Syst. Secur.*, 4(3):191–233, 2001.

[BBKW10] Achim D. Brucker, Lukas Brügger, Paul Kearney, and Burkhart Wolff. Verified firewall policy transformations for test case generation. In *Third*

International Conference on Software Testing, Verification and Validation (ICST), pages 345–354, April 2010.

[BBKW11] Achim D. Brucker, Lukas Brügger, Paul Kearney, and Burkhart Wolff. An approach to modular and testable security models of real-world health-care applications. In *Proceedings of the 16th ACM Symposium on Access Control Models and Technologies*, number 10, pages 133–142, New York, NY, USA, 2011. ACM.

[BBW15] Achim D. Brucker, Lukas Brügger, and Burkhart Wolff. Formal firewall conformance testing: an application of test and proof techniques. *Softw. Test., Verif. Reliab.*, 25(1):34–71, 2015.

[BCG+01] J. Burns, A. Cheng, P. Gurung, S. Rajagopalan, P. Rao, D. Rosenbluth, A. V. Surendran, and D. M. Martin. Automatic management of network security policy. In *DARPA Information Survivability Conference & Exposition II (DISCEX '01), Volume 2*, pages 12 – 26, Anaheim, CA, 12 June – 14 June 2001. DARPA in cooperation with the IEEE Computer Society's Technical Committee on Security and Privacy, IEEE.

[BCL11] Christopher Bailey, David W. Chadwick, and Rogerio de Lemos. Self-adaptive authorization framework for policy based RBAC/ABAC models. In *Proceedings of the 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, DASC '11, pages 37–44, Washington, DC, USA, 2011. IEEE Computer Society.

[BCL12] Cataldo Basile, Alberto Cappadonia, and Antonio Lioy. Network-level access control policy analysis and transformation. *IEEE/ACM Transactions on Networking*, 20(4):985–998, Aug 2012.

[BCV04] Mandis S. Beigi, Seraphin Calo, and Dinesh Verma. Policy transformation techniques in policy-based systems management. In *Proceedings. Fifth IEEE International Workshop on Policies for Distributed Systems and Networks, 2004. POLICY 2004.*, pages 13–22, June 2004.

[BDGPM20] Paolo Bellavista, Alessandro Dolci, Carlo Giannelli, and Dmitrij David Padalino Montenero. SDN-based traffic management middleware for spontaneous WMNs. *Journal of Network and Systems Management*, 28(4):1575–1609, 2020.

[BdS00] Piero Bonatti, Sabrina de Capitani di Vimercati, and Pierangela Samarati. A modular approach to composing access control policies. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*, CCS '00, pages 164–173, New York, NY, USA, 2000. ACM.

[BDS02] Piero Bonatti, Sabrina De Capitani di Vimercati, and Pierangela Samarati. An algebra for composing access control policies. *ACM Trans. Inf. Syst. Secur.*, 5(1):1–35, February 2002.

[BDS04] Michael Backes, Markus Dürmuth, and Rainer Steinwandt. An algebra for composing enterprise privacy policies. In Pierangela Samarati, Peter Ryan, Dieter Gollmann, and Refik Molva, editors, *Computer Security – ESORICS 2004*, pages 33–52, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[Bel99] Steven M Bellovin. Distributed firewalls, 1999.

[Ben06] Messaoud Benantar. *Access Control Systems: Security, Identity Management and Trust Models.* Access Control Systems: Security, Identity Management and Trust Models. Springer US, 2006.

[BGH+14] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi

Kobayashi, Toshio Koide, Bob Lantz, Brian P. O'Connor, Pavlin Radoslavov, William Snow, and Guru M. Parulkar. ONOS: Towards an open, distributed SDN OS. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 1–6, New York, NY, USA, 2014. Association for Computing Machinery.

[Bib75] Kenneth J. Biba. Integrity considerations for secure computer systems. Technical report, Mitre Corporation, 1975.

[Bis02] Matt Bishop. *Computer Security: Art and Science*. Addison-Wesley Professional, 2002.

[BJD16] Idris Zoher Bholebawa, Rakesh Kumar Jha, and Upena D. Dalal. Performance analysis of proposed openflow-based network architecture using mininet. *Wireless Personal Communications*, 86(2):943–958, 2016.

[BK17] Venkatraman Balasubramanian and Ahmed Karmouch. Managing the mobile ad-hoc cloud ecosystem using software defined networking principles. In *2017 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–6, 2017.

[BKS97] Chris Brink, Wolfram Kahl, and Gunther Schmidt. *Relational Methods in Computer Science*. Advances in Computing. Springer-Verlag, Wien, New York, 1997. ISBN 3-211-82971-7.

[BLMR04] Arosha K. Bandara, Emil C. Lupu, Jonathan Moffett, and Alessandra Russo. A goal-based approach to policy refinement. In *Proceedings. Fifth IEEE International Workshop on Policies for Distributed Systems and Networks, 2004. POLICY 2004.*, pages 229–239, June 2004.

[BLP76] D. Elliott Bell and Leonard J. La Padula. Secure computer system: Unified exposition and multics interpretation. Technical report, The MITRE Corporation, March 1976.

[BLR+05] Arosha Bandara, Emil Lupu, Alessandra Russo, Naranker Dulay, Morris Sloman, Paris Flegkas, Marinos Charalambides, and George Pavlou. Policy refinement for diffserv quality of service management. In *2005 9th IFIP/IEEE International Symposium on Integrated Network Management, 2005. IM 2005.*, pages 469–482, May 2005.

[BN89] David F. C. Brewer and Michael J. Nash. The chinese wall security policy. In *Proceedings. 1989 IEEE Symposium on Security and Privacy*, pages 206–214, May 1989.

[Brü12] Lukas Brügger. *A Framework for Modelling and Testing of Security Policies.* PhD thesis, ETH ZURICH, 2012.

[CB94] William R. Cheswick and Steven M. Bellovin. *Firewalls and Internet Security: Repelling the Wily Hacker.* Addison-Wesley, 1994.

[Cen15] Center for Internet Security (CIS). Critical security controls for effective cyber defense version 6.0. Technical report, CIS, 2015.

[Cha13] Chi-Shih Chao. *A Novel and Feasible System for Rule Anomaly and Behavior Mismatching Diagnosis Among Firewalls*, pages 49–57. Springer New York, New York, NY, 2013.

[CHA+18] Ankur Chowdhary, Dijiang Huang, Adel Alshamrani, Abdulhakim Sabur, Myong Kang, Anya Kim, and Alexander Velazquez. SDFW: SDN-based stateful distributed firewall. *arXiv preprint arXiv:1811.00634*, 2018.

[CLL+09] Robert Craven, Jorge Lobo, Emil Lupu, Alessandra Russo, and Morris Sloman. Security policy refinement using data integration: A position paper. In *Proceedings of the 2Nd ACM Workshop on Assurable and Usable Security Configuration*, SafeConfig '09, pages 25–28, New York, NY, USA, 2009. ACM.

[CLL+10] Robert Craven, Jorge Lobo, Emil Lupu, Alessandra Russo, and Morris Sloman. Decomposition techniques for policy refinement. In *2010 International Conference on Network and Service Management*, pages 72–79, Oct 2010.

[CMV00] Paul C. Clark, Marion C. Meissner, and Karen O. Vance. Secure compartmented data access over an untrusted network using a cots-based architecture. In *Proceedings 16th Annual Computer Security Applications Conference (ACSAC'00)*, pages 217–223, Dec 2000.

[CN02] Paul Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley Professional, 2002.

[Co.] Oracle Co. Java remote method invocation - distributed computing for java. Available: `https://www.oracle.com/technetwork/java/javase/tech/index-jsp-138781.html` (Accessed: Feb 17, 2019).

[Col09] Eric Cole. *Network Security Bible*. Wiley Publishing, 2nd edition, 2009.

[Com20] Gerald Combs. Wireshark. Available: `https://www.wireshark.org/` (Accessed: May 29, 2020), 2020.

[CRDP19] Maurantonio Caprolu, Simone Raponi, and Roberto Di Pietro. FORTRESS: An efficient and distributed firewall for stateful data plane SDN. *Security and Communication Networks*, 2019, 2019.

[CRK+07] Pau–Chen Cheng, Pankaj Rohatgi, Claudia Keser, Paul A. Karger, Grant M. Wagner, and Angela Schuett Reninger. Fuzzy multi-level security: An experiment on quantified risk-adaptive access control. In *IEEE Symposium on Security and Privacy*, pages 222–230, May 2007.

[CSFM07] Venanzio Capretta, Bernard Stepien, Amy Felty, and Stan Matwin. Formal correctness of conflict detection for firewalls. In *Proceedings of the 2007 ACM Workshop on Formal Methods in Security Engineering*, FMSE '07, pages 22–30, New York, NY, USA, 2007. Association for Computing Machinery.

[CSL+20] Ioannis P. Chochliouros, Anastasia S. Spiliopoulou, Pavlos Lazaridis, Athanassios Dardamanis, Zaharias Zaharis, and Alexandros Kostopoulos. Dynamic network slicing: Challenges and opportunities. In Ilias Maglogiannis, Lazaros Iliadis, and Elias Pimenidis, editors, *Artificial Intelligence Applications and Innovations. AIAI 2020 IFIP WG 12.5 International Workshops*, pages 47–60, Cham, 2020. Springer International Publishing.

[cSYA09] Cheng chun Shu, Erica Y. Yang, and Alvaro E. Arenas. Detecting conflicts in ABAC policies with rule-reduction and binary-search techniques. In *2009 IEEE International Symposium on Policies for Distributed Systems and Networks*, pages 182–185, July 2009.

[CT08] Gavin A. Campbell and Kenneth J. Turner. Goals and policies for sensor network management. In *2008 Second International Conference on Sensor Technologies and Applications (sensorcomm 2008)*, pages 354–359, Aug 2008.

[CYY16] Laizhong Cui, F. Richard Yu, and Qiao Yan. When big data meets software-defined networking: Sdn for big data and big data for sdn. *IEEE Network*, 30(1):58–65, 2016.

[DBS+95] Jules Desharnais, Nadir Belkhiter, Salah Ben Mohamed Sghaier, Fairouz Tchier, Ali Jaoua, Ali Mili, and Nejib Zaguia. Embedding a demonic semilattice in a relation algebra. *Theoretical Computer Science*, 149(2):333 – 360, 1995.

[DCA+17] Tooska Dargahi, Alberto Caponi, Moreno Ambrosin, Giuseppe Bianchi, and Mauro Conti. A survey on the security of stateful SDN data planes. *IEEE Communications Surveys & Tutorials*, 19(3):1701–1725, 2017.

[DCE08] M.A.C. Dekker, Jason Crampton, and Sandro Etalle. RBAC administration in distributed systems. In *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies*, SACMAT '08, pages 93–102, New York, NY, USA, 2008. ACM.

[DEF+20] Jon Dugan, John Estabrook, Jim Ferbuson, Andrew Gallatin, Mark Gates, Kevin Gibbs, Stephen Hemminger, Nathan Jones, Feng Qin, Gerrit Renker, Ajay Tirumala, and Alex Warshavsky. iPerf. Available: `https://iperf.fr/` (Accessed: May 29, 2020), 2020.

[Der15] Mahdi Derakhshanmanesh. *Model-Integrating Software Components: Engineering Flexible Software Systems*. Springer Fachmedien Wiesbaden, 2015.

[DFKM98] Jules Desharnais, Marc Frappier, Ridha Khedri, and Ali Mili. Integration of sequential scenarios. *IEEE Transactions on Software Engineering*, 24(9):695 – 708, September 1998.

[DFLPW07] Nelly Delessy, Eduardo B. Fernandez, Maria M. Larrondo-Petrie, and Jie

Wu. Patterns for access control in distributed systems. In *Proceedings of the 14th Conference on Pattern Languages of Programs*, pages 3:1–3:11, New York, NY, USA, 2007. ACM.

[DFSJ07] Sabrina De Capitan di Vimercati, Sara Foresti, Pierangela Samarati, and Sushil Jajodia. Access control policies and languages. *Int. J. Comput. Sci. Eng.*, 3(2):94–102, 2007.

[DHC15] Cornelius Diekmann, Lars Hupel, and Georg Carle. Semantics-preserving simplification of real-world firewall rule sets. In N. Bjørner and F. de Boer, editors, *FM 2015: Formal Methods*, pages 195–212, Cham, 2015. Springer International Publishing.

[dMB09] Leonardo de Moura and Nikolaj Bjørner. Satisfiability modulo theories: An appetizer. In Marcel Vinícius Medeiros Oliveira and Jim Woodcock, editors, *Formal Methods: Foundations and Applications*, pages 23–36, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[DMHC16] Cornelius Diekmann, Julius Michaelis, Maximilian Haslbeck, and Georg Carle. Verified iptables firewall analysis. In *IFIP Networking Conference (IFIP Networking) and Workshops*, pages 252–260, May 2016.

[dOSSP14] Rogério Leão Santos de Oliveira, Christiane Marie Schweitzer, Ailton Akira Shinoda, and Ligia Rodrigues Prete. Using mininet for emulation and prototyping software-defined networks. In *2014 IEEE Colombian Conference on Communications and Computing (COLCOM)*, pages 1–6, June 2014.

[DPRW07] Rinku Dewri, Nayot Poolsappasit, Indrajit Ray, and Darrell Whitley. Optimal security hardening using multi-objective optimization on attack tree models of networks. In *Proc. 14th ACM Conf. Comput. Commun. Security*, pages 204–213, 2007.

[DSE12] Mahdi Derakhshanmanesh, Mazeiar Salehie, and Jürgen Ebert. Towards model-centric engineering of a dynamic access control product line. In *Proceedings of the 16th International Software Product Line Conference*, volume 2, New York, NY, USA, 2012. ACM.

[dSSdLPDF19] Lucas Soares da Silva, Carlos Renato Storck, and Fãtima de L. P. Duarte-Figueiredo. A dynamic load balancing algorithm for data plane traffic. In *LANOMS*, 2019.

[DZLL14] Rong Du, Chenglin Zhao, Shenghong Li, and Jian Li. Efficient weakly secure network coding scheme against node conspiracy attack based on network segmentation radar and sonar networks. *Eurasip Journal on Wireless Communications and Networking*, 2014, 2014.

[FH06] Tor Erlend Fægri and Svein O. Hallsteinsen. A software product line reference architecture for security. In *Software Product Lines - Research Issues in Engineering and Management*, pages 275–326. Springer Berlin Heidelberg, 2006.

[FK92] David Ferraiolo and Richard Kuhn. Role-based access control. In *15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[flo] Floddlight controller. Available: `https://floodlight.atlassian.net` (Accessed: May 06, 2020).

[FSG+01] David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4(3):224–274, August 2001.

[GJD11] Deepak Garg, Limin Jia, and Anupam Datta. Policy auditing over incomplete logs: Theory, implementation and applications. In *Proceedings of the*

18th ACM Conference on Computer and Communications Security, CCS '11, pages 151–162, New York, NY, USA, 2011. ACM.

[GKK16] Vipin Gupta, Sukhveer Kaur, and Karamjeet Kaur. Implementation of stateful firewall using POX controller. In *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 1093–1096. IEEE, 2016.

[GMKA06] Korosh Golnabi, Richard K. Min, Latifur Khan, and Ehab Al-Shaer. Analysis of firewall policy rules using data mining techniques. In *2006 IEEE/IFIP Network Operations and Management Symposium NOMS 2006*, pages 305–315, April 2006.

[Goo12] Google Inc. Google's approach to it security. Technical report, Google, 2012.

[GR95] Barbara Guttman and Edward A. Roback. An introduction to computer security: The NIST handbook. Technical report, National Institute of Standards & Technology, 1995.

[GS93] David Gries and Fred B. Schenider. *A Logical Approach to Discrete Math.* Springer Texts And Monographs In Computer Science. Springer-Verlag, New York, 1993.

[HAK12] Hongxin Hu, Gail-Joon Ahn, and Ketan Kulkarni. Detecting and resolving firewall policy anomalies. *IEEE Transactions on Dependable and Secure Computing*, 9(3):318–331, May 2012.

[HCC+13] Safaà Hachana, Frédéric Cuppens, Nora Cuppens-Boulahia, Vijay Atluri, and Stephane Morucci. Policy mining: A bottom-up approach toward a model based firewall management. In Aditya Bagchi and Indrakshi Ray,

editors, *Information Systems Security*, pages 133–147, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[HFK06] Vincent C. Hu, David F. Ferraiolo, and D. Rick Kuhn. Assessment of access control systems. Technical report, NIST Interagency Report 7316, 2006.

[HFK+14] Vincent C. Hu, David Ferraiolo, D. Rick Kuhn, Adam Schnitzer, Kenneth Sandlin, Robert Miller, and Karen Scarfone. Guide to attribute based access control (abac) definition and considerations, January 2014.

[HHPS08] Svein Hallsteinsen, Mike Hinchey, Sooyong Park, and Klaus Schmid. Dynamic software product lines. *Computer*, 41(4):93 – 95, April 2008.

[Hin99] Susan Hinrichs. Policy-based management: bridging the gap. In *Proceedings 15th Annual Computer Security Applications Conference (ACSAC'99)*, pages 209–218, Dec 1999.

[HKFV15] Vincent C. Hu, D. Richard Kuhn, David F. Ferraiolo, and Jeffrey Voas. Attribute-based access control. *Computer*, 48(2):85–88, 2015.

[HKH+05] Nen-Fu Huang, Chia-Nan Kao, Hsien-Wei Hun, Gin-Yuan Jai, and Chia-Lin Lin. Apply data mining to defense-in-depth network security system. In *19th International Conference on Advanced Information Networking and Applications (AINA'05) Volume 1 (AINA papers)*, volume 2, pages 159–162 vol.2, March 2005.

[HKM06] Peter Höfner, Ridha Khedri, and Bernhard Möller. Feature algebra. In Jayadev Misra, Tobias Nipkow, and Emil Sekerinski, editors, *FM 2006: Formal Methods*, volume 4085 of *Lecture Notes in Computer Science series*, pages 300 – 315, 14th International Symposium on Formal Methods,

McMaster University, Hamilton, Ontario, Canada, August 21 – 27 2006. Springer.

[HKM08] Peter Höfner, Ridha Khedri, and Bernhard Möller. Algebraic view reconciliation. In *6th IEEE International Conferences on Software Engineering and Formal Methods*, pages 85 – 94. Cape Town, South Africa, November 10 – 14, 2008.

[HKM11a] Peter Höfner, Ridha Khedri, and Bernhard Möller. An algebra of product families. *Software & Systems Modeling*, 10(2):161–182, 2011.

[HKM11b] Peter Höfner, Ridha Khedri, and Bernhard Möller. Supplementing product families with behaviour. *International Journal of Software and Informatics*, pages 245–266, 2011.

[HNVC13] Sylvain Hallé, Éric Lunaud Ngoupé, Roger Villemaire, and Omar Cherkaoui. Distributed firewall anomaly detection through LTL model checking. In *IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pages 194–201, May 2013.

[HOKY07] Manabu Hirano, Takeshi Okuda, Eiji Kawai, and Suguru Yamaguchi. Design and implementation of a portable id management framework for a secure virtual machine monitor. In *Journal of Information Assurance and Security (JIAS)*. Dynamic Publishers, 2007.

[HPF16] Jose-Miguel Horcas, Mónica Pinto, and Lidia Fuentes. An automatic process for weaving functional quality attributes using a software product line approach. *Journal of Systems and Software*, 112:78 – 95, 2016.

[HRU76] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. *Commun. ACM*, 19(8):461–471, August 1976.

[HSE+08] Manabu Hirano, Takahiro Shinagawa, Hideki Eiraku, Shoichi Hasegawa, Kazumasa Omote, Koichi Tanimoto, Takashi Horie, Kazuhiko Kato, Takeshi Okuda, Eiji Kawai, and Suguru Yamaguchi. Introducing role-based access control to a secure virtual machine monitor: Security policy enforcement mechanism for distributed computers. In *IEEE Asia-Pacific Services Computing Conference APSCC*, pages 1225–1230, 2008 2008.

[HSM12] Brandon Heller, Rob Sherwood, and Nick McKeown. The controller placement problem. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 7–12, New York, NY, USA, 2012. Association for Computing Machinery.

[HSP00] Adiseshu Hari, Subhash Suri, and Guru Parulkar. Detecting and resolving packet filter conflicts. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*, volume 3, pages 1203–1212 vol.3, March 2000.

[Hun80] Thomas W Hungerford. Algebra, volume 73 of graduate texts in mathematics, 1980.

[HW98] Udo Hebisch and Hanns Joachim Weinert. *Semirings: algebraic theory and applications in computer science*, volume 5. World Scientific, 1998.

[HXCL12] JeeHyun Hwang, Tao Xie, Fei Chen, and Alex X. Liu. Systematic structural testing of firewall policies. *IEEE Transactions on Network and Service Management*, 9(1):1–11, March 2012.

[ILP06] Kyle Ingols, Richard Lippmann, and Keith Piwowarski. Practical attack graph generation for network defense. In *Proc. Comput. Security Appl. Conf.*, pages 121—30, 2006.

[JGT+11] Karthick Jayaraman, Vijay Ganesh, Mahesh Tripunitara, Martin Rinard, and Steve Chapin. Automatic error finding in access-control policies. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 163–174, New York, NY, USA, 2011. ACM.

[JK01] Ryszard Janicki and Ridha Khedri. On a formal semantics of tabular expressions. *Science of Computer Programming*, 39(1-2):189–213, March 2001.

[JP10] Ying Jin and David Lorge Parnas. Defining the meaning of tabular mathematical expressions. *Science of Computer Programming*, 75(11):980 – 1000, 2010. Special Section on the Programming Languages Track at the 23rd ACM Symposium on Applied Computing.

[JPZ97] Ryszard Janicki, David Lorge Parnas, and Jeffery Zucker. *Tabular Representations in Relational Documents*, pages 184–196. Springer Vienna, Vienna, 1997.

[JRR14] Tariq Javid, Tehseen Riaz, and Asad Rasheed. A layer2 firewall for software defined network. In *2014 Conference on Information Assurance and Cyber Security (CIACS)*, pages 39–42. IEEE, 2014.

[JS09] Alan Jeffrey and Taghrid Samak. Model checking firewall policy configurations. In *Policies for Distributed Systems and Networks, 2009. POLICY 2009. IEEE International Symposium on*, pages 60–67, July 2009.

[JSBZ15] Sebastian Jeuk, Gonzalo Salgueiro, Fred Baker, and Shi Zhou. Network segmentation in the cloud a novel architecture based on ucc and iid. In *2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*, pages 58–63, Oct 2015.

[JSSS01] Sushil Jajodia, Pierangela Samarati, Maria Luisa Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. *ACM Trans. Database Syst.*, 26(2):214–260, June 2001.

[KBB+03] Anas Abou El Kalam, Rania El Baida, Philippe Balbiani, Salem Benferhat, Frédéric Cuppens, Yves Deswarte, Alexandre Miège, Claire Saurel, and Gilles Trouessin. Organization based access control. In *IEEE 4th International Workshop on Policies for Distributed Systems and Networks POLICY*, pages 120–131, June 2003.

[KCH+90] Kyo Kang, Sholom Cohen, James Hess, William Novak, and A. Spencer Peterson. Feature oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Nov 1990.

[KD06] Timo Käkölä and Juan Carlos Dueñas. *Software Product Lines*. Springer-Verlag Berlin Heidelberg, 2006.

[KFAS14] Rui Kubo, Tomonori Fujita, Yuji Agawa, and Hikaru Suzuki. Ryu SDN framework—open-source SDN platform software. Available: `https://osrg.github.io/ryu/index.html` (Accessed: May 06, 2020), 08 2014.

[KG19] Rakesh Kumar and Rinkaj Goyal. On cloud security requirements, threats, vulnerabilities and countermeasures: A survey. *Computer Science Review*, 33:1 – 48, 2019.

[KH13] Shobhanjana Kalita and Shyamanta M. Hazarika. A rule relation calculus for verification and validation of firewalls. In *National Conference on Communications (NCC)*, pages 1–5, Feb 2013.

[Khe08] Ridha Khedri. Formal model driven approach to deal with requirements

volatility. Technical report, Department of Computing and SoftwareFaculty of Engineering — McMaster University, 2008.

[KJA17] Ridha Khedri, Owain Jones, and Mohammed Alabbad. Defense in depth formulation and usage in dynamic access control. In Matteo Maffei and Mark Ryan, editors, *Principles of Security and Trust: 6th International Conference, POST 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, pages 253–274, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg.

[KKE+20] Myoungsu Kim, Soonhong Kwon, Donald Elmazi, Jong-Hyouk Lee, Leonard Barolli, and Kangbin Yim. A technical survey on methods for detecting rogue access points. In Leonard Barolli, Fatos Xhafa, and Omar K. Hussain, editors, *Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 215–226, Cham, 2020. Springer International Publishing.

[KKLS14] Sangsig Kim, Dae-Kyoo Kim, Lunjin Lu, and Eunjee Song. Building hybrid access control by configuring rbac and mac features. *Information and Software Technology*, 56(7):763 – 792, 2014.

[KKSG15] Karamjeet Kaur, Krishan Kumar, Japinder Singh, and Navtej Singh Ghumman. Programmable firewall using software defined networking. In *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 2125–2129. IEEE, 2015.

[KLG14] Ian Ku, You Lu, and Mario Gerla. Software-defined mobile cloud: Architecture, services and use cases. In *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 1–6, 2014.

[KMA19] Ridha Khedri, Neerja Mhaskar, and Mohammed Alabbad. On the segmentation of networks. Technical report, McMaster University, 2019. Available: `https://www.cas.mcmaster.ca/tech_reports/0reports/CAS-19-01-RK.pdf`.

[KMNH19] Enio Kaljic, Almir Maric, Pamela Njemcevic, and Mesud Hadzialic. A survey on data plane flexibility and programmability in software-defined networking. *IEEE Access*, 7:47804–47840, 2019.

[KS16] Avinash Kumar and NK Srinath. Implementing a firewall functionality for mesh networks using SDN controller. In *2016 International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS)*, pages 168–173. IEEE, 2016.

[KWS03] Ridha Khedri, Rong Wu, and Bahati Sanga. SCENATOR: A prototype tool for requirements inconsistency detection. In F. Wang and I. Lee, editors, *Proceedings of the 1st International Workshop on Automated Technology for Verification and Analysis*, pages 75 – 86, Taiwan, Republic of China, December 10 – 13 2003. National Taiwan University.

[Lam74] Butler W. Lampson. Protection. *SIGOPS Oper. Syst. Rev.*, 8(1):18–24, January 1974.

[LFP11] Ramón Lence, Lidia Fuentes, and Mónica Pinto. Quality attributes and variability in AO-ADL software architectures. In *Proceedings of the 5th European Conference on Software Architecture: Companion Volume*, ECSA '11, New York, NY, USA, 2011. Association for Computing Machinery.

[LIS+06] Richard Lippmann, Kyle Ingols, Chris Scott, Keith Piwowarski, Kendra Kratkiewicz, Mike Artz, and Robert Cunningham. Validating and restoring

defense in depth using attack graphs. In *MILCOM 2006 - 2006 IEEE Military Communications conference*, pages 1 – 10, Oct 2006.

[LLT+13] Yang Luo, Yazhuo Li, Qing Tang, Zhao Wei, and Chunhe Xia. VRBAC: An extended RBAC model for virtualized environment and its conflict checking approach. In J. Su, B. Zhao, Z. Sun, X. Wang, F. Wang, and K. Xu, editors, *Frontiers in Internet Technologies*, volume 401 of *Communications in Computer and Information Science*, pages 194–205. Springer Berlin Heidelberg, 2013.

[LS97] Emil Lupu and Morris Sloman. *Conflict Analysis for Management Policies*, pages 430–443. Springer US, Boston, MA, 1997.

[LS99] Emil C. Lupu and Morris Sloman. Conflicts in policy-based distributed systems management. *IEEE Transactions on Software Engineering*, 25(6):852–869, Nov 1999.

[LS15] Claas Lorenz and Bettina Schnor. Policy anomaly detection for distributed IPv6 firewalls. In *12th International Joint Conference on e-Business and Telecommunications (ICETE)*, volume 04, pages 210–219, July 2015.

[MAK21] Neerja Mhaskar, Mohammed Alabbad, and Ridha Khedri. A formal approach to network segmentation. *Computers & Security*, page 102162, 2021.

[MAN16] Sergey Morzhov, Igor Alekseev, and Mikhail Nikitinskiy. Firewall application for floodlight SDN controller. In *2016 International Siberian Conference on Control and Communications (SIBCON)*, pages 1–5. IEEE, 2016.

[MATWYO16] Hussain M.J. Almohri, Layne T. Watson, Danfeng Yao, and Xinming Ou.

Security optimization of dynamic networks with probabilistic graph modeling and linear programming. In *IEEE Transactions on Dependable and Secure Computing*, volume 13, pages 474–487, 2016.

[MBG+14] Masoud Moshref, Apoorv Bhargava, Adhip Gupta, Minlan Yu, and Ramesh Govindan. Flow-level state transition as a new switch primitive for SDN. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 61–66, New York, NY, USA, 2014. Association for Computing Machinery.

[MCD15] Ferney A. Maldonado-Lopez, Eusebi Calle, and Yezid Donoso. Detection and prevention of firewall-rule conflicts on software-defined networking. In *2015 7th International Workshop on Reliable Networks Design and Modeling (RNDM)*, pages 259–265, Oct 2015.

[MCL+13] Mirko Montanari, Ellick Chan, Kevin Larson, Wucherl Yoo, and Roy H Campbell. Distributed security policy conformance. *Computers & Security*, 33:28–40, 2013.

[MDS+17] Diomidis S. Michalopoulos, Mark Doll, Vincenzo Sciancalepore, Dario Bega, Peter. Schneider, and Peter Rost. Network slicing via function decomposition and flexible network design. In *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 1–6, 2017.

[Mei15] JiaBo Mei. An optimized method of firewall policy exception handling in cloud environment. In *2015 8th International Symposium on Computational Intelligence and Design (ISCID)*, volume 1, pages 149–152, Dec 2015.

[MHMR06] Christopher J. May, Josh Hammerstein, Jeff Mattson, and Kristopher Rush.

Defense in depth: Foundations for secure and resilient it enterprises. Technical report, Carnegie Mellon University, 2006.

[MHS+16] Olli Mämmelä, Jouni Hiltunen, Jani Suomalainen, Kimmo Ahola, Petteri Mannersalo, and Janne Vehkaperä. Towards micro-segmentation in 5G network security. In *European Conference on Networks and Communications (EUCNC)*, 2016. Project code: 101719 ; European Conference on Networks and Communications : Workshop on Network Management, Quality of Service and Security for 5G Networks, 2016 EuCNC2016, EUCNC 2016 ; Conference date: 27-06-2016 Through 30-06-2016.

[MMV16] Theodoros Mavroeidakos, Angelos Michalas, and Dimitrios D. Vergados. Security architecture based on defense in depth for cloud computing environment. In *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 334–339, April 2016.

[MS06] Bernhard Möller and Georg Struth. wp is wlp. In W. MacCaull, M. Winter, and I. Düntsch, editors, *Relational Methods in Computer Science*, volume 3929 of *Lecture Notes in Computer Science*, pages 200–211. Springer Berlin Heidelberg, 2006.

[Muu83] Mike Muuss. The story of the PING program. Available: `https://ftp.arl.army.mil/~mike/ping.html` (Accessed: May 29, 2020), 1983.

[Nat13] National Security Agency (NSA). Top 10 information assurance mitigation strategies. Technical report, NSA, 2013.

[NF18] Ultana Neville and Simon N Foley. Reasoning about firewall policies through refinement and composition. *Journal of Computer Security*, 26(2):207–254, January 2018.

[NGM16] NGMN Alliance. Description of network slicing concept. `https://www.ngmn.org/wp-content/uploads/Publications/2016/161010_NGMN_Network_Slicing_framework_v1.0.8.pdf`, September 2016.

[np18] The netfilter.org project. netfilter/iptables project. `https://netfilter.org/`, 2018.

[OSRSC01] Sam Owre, Natarajan Shankar, John M. Rushby, and David W. J. Stringer-Calvert. PVS system guide. Computer Science Laboratory, 333 Ravenswood Avenue, Menlo Park CA 94025, 2001.

[Par83] David Lorge Parnas. A generalized control structure and its formal definition. *Commun. ACM*, 26(8):572–581, Aug 1983.

[Par97] David Lorge Parnas. Precise description and specification of software. In *Software Fundamentals*. Addison-Wesley, 1997.

[PBvdL05] Klaus Pohl, Günter Böckle, and Frank van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., 2005.

[PDR12] Nayot Poolsappasit, Rinku Dewri, and Indrajit Ray. Dynamic security risk management using bayesian attack graphs. In *IEEE Transactions on Dependable and Secure Computing*, volume 9, pages 61–74, 2012.

[Pet01] Razvan Peteanu. *Best Practices for Secure Development*, 2001.

[PGF+15] Mónica Pinto, Nadia Gámez, Lidia Fuentes, Mercedes Amor, JoséMiguel Horcas, and Inmaculada Ayala. Dynamic reconfiguration of security policies in wireless sensor networks. *Sensors (Basel, Switzerland)*, 15(3):5251–5280, 03 2015.

[PPK+] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Jonathan Stringer, Keith Amidon Pravin Shelar, and Martín Casado. Open vSwitch. Available: `https://www.openvswitch.org/` (Accessed: April 27, 2020).

[PW05] Jon Pincus and Jeannette M. Wing. Towards an algebra for security policies. In Gianfranco Ciardo and Philippe Darondeau, editors, *Applications and Theory of Petri Nets 2005*, pages 17–25, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[PY14] Justin Gregory V Pena and William Emmanuel Yu. Development of a distributed firewall using software defined networking technology. In *2014 4th IEEE International Conference on Information Science and Technology*, pages 449–452. IEEE, 2014.

[RAAS14] A. Rehman, S. Alqahtani, A. Altameem, and T. Saba. Virtual machine security challenges: case studies. *International Journal of Machine Learning and Cybernetics*, 5(5):729–742, 2014.

[Ram02] Jay Ramachandran. *Designing Security Architecture Solutions*. Wiley, 2002.

[RAS13] Mohammad Ashiqur Rahman and Ehab Al-Shaer. A formal framework for network security design synthesis. In *2013 IEEE 33rd International Conference on Distributed Computing Systems (ICDCS)*, pages 560 – 70, 2013.

[RIHP05] Paul Rubel, Michael Ihde, Steven Harp, and Charles Payne. Generating policies for defense in depth. In *21st Annual Computer Security Applications Conference (ACSAC'05)*, pages 10 pp.–514, Dec 2005.

[Ris13] Erik Rissanen. OASIS. eXtesible access control markup language

(XACML) version 3.0. Available: `http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html` (Accessed: Dec 11, 2015), 2013.

[RLB+09] Prathima Rao, Dan Lin, Elisa Bertino, Ninghui Li, and Jorge Lobo. An algebra for fine-grained integration of xacml policies. In *Proceedings of the 14th ACM Symposium on Access Control Models and Technologies*, SACMAT '09, pages 63–72, New York, NY, USA, 2009. ACM.

[Ros02] Kenneth H. Rosen. *Discrete Mathematics and Its Applications*. McGraw-Hill Higher Education, 5th edition, 2002.

[RS93] John Rushby and Mandayam Srivas. Using PVS to prove some theorems of David Parnas. In J. J. Joyce and C.-J. H. Seger, editors, *Higher Order Logic Theorem Proving and its Applications (6th International Workshop, HUG '93)*, volume 780 of *Lecture Notes in Computer Science*, pages 163–173, Vancouver, Canada, aug 1993. Springer-Verlag.

[RSC+06] Javier Rubio-Loyola, Joan Serrat, Marinos Charalambides, Paris Flegkas, and George Pavlou. A functional solution for goal-oriented policy refinement. In *Seventh IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'06)*, pages 133–144, June 2006.

[Rus02] Rusty Russell. Linux 2.4 packet filtering HOWTO. Available: `http://www.netfilter.org/documentation/HOWTO//packet-filtering-HOWTO.html` (Accessed: Dec 11, 2015), 2002.

[Sab18] Khair Eddin Sabri. An algebraic model to analyze role-based access control policies. *Modern Applied Science*, 12(10):50–57, 2018.

[SB97] Robert N. Smith and Sourav Bhattacharya. Firewall placement in a large network topology. In *Proceedings of the 6th IEEE Workshop on Future*

Trends of Distributed Computing Systems, FTDCS '97, pages 40–, Washington, DC, USA, 1997. IEEE Computer Society.

[SBC+17] Chen Sun, Jun Bi, Haoxian Chen, Hongxin Hu, Zhilong Zheng, Shuyong Zhu, and Chenghui Wu. SDPA: Toward a stateful data plane in software-defined networking. *IEEE/ACM Transactions on Networking*, 25(6):3294–3308, 2017.

[SBM99] Ravi Sandhu, Venkata Bhamidipati, and Qamar Munawer. The ARBAC97 model for role-based administration of roles. *ACM Trans. Inf. Syst. Secur.*, 2(1):105–135, February 1999.

[SCFY96] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.

[SD01] Pierangela Samarati and Sabrina De Capitan di Vimercati. Access control: Policies, models, and mechanisms. In *Revised Versions of Lectures Given During the IFIP WG 1.7 International School on Foundations of Security Analysis and Design on Foundations of Security Analysis and Design: Tutorial Lectures*, FOSAD '00, 2001.

[SFK00] Ravi Sandhu, David Ferraiolo, and Richard Kuhn. The NIST model for role-based access control: Towards a unified standard. In *Proceedings of the Fifth ACM Workshop on Role-based Access Control*, RBAC '00, pages 47–63, New York, NY, USA, 2000. ACM.

[SH09] Karen Scarfone and Paul Hoffman. Guidelines on firewalls and firewall policy. Technical report, National Institute of Standards and Technology (NIST), 2009.

[SH16] Khair Eddin Sabri and Hazem Hiary. Algebraic model for handling access

control policies. *Procedia Computer Science*, 83:653 – 657, 2016. The 7th International Conference on Ambient Systems, Networks and Technologies (ANT 2016) / The 6th International Conference on Sustainable Energy Information Technology (SEIT-2016) / Affiliated Workshops.

[SH17] Sibylle Schaller and Dave Hood. Software defined networking architecture standardization. *Computer Standards & Interfaces*, 54:197 – 202, 2017. SI: Standardization SDN&NFV.

[SHBS19] Hadar Sufiev, Yoram Haddad, Leonid Barenboim, and José Soler. Dynamic SDN controller load balancing. *Future Internet*, 11:75, 03 2019.

[SKBJ20] Arash Shaghaghi, Mohamed Ali Kaafar, Rajkumar Buyya, and Sanjay Jha. *Software-Defined Network (SDN) Data Plane Security: Issues, Solutions, and Future Directions*, pages 341–387. Springer International Publishing, Cham, 2020.

[SPB+20] Davide Sanvito, Luca Pollini, Nicola Bonelli, Eder Leão Fernandes, and Carmelo Cascone. BEBA software switch. Available: `http://www.beba-project.eu/` (Accessed: April 27, 2020), 2020.

[SPLY14] Michelle Suh, Sae Hyong Park, Byungjoon Lee, and Sunhee Yang. Building firewall over the software-defined network controller. In *16th International Conference on Advanced Communication Technology*, pages 744–748. IEEE, 2014.

[SRI02] SRI International. Prototype verification system (PVS). `http://www.csl.sri.com/projects/pvs/` (Accessed: May 11, 2016), 2002.

[SS93] Gunther Schmidt and Thomas Ströhlein. *Relations and Graphs: Discrete*

Mathematics for Computer Scientists. Springer-Verlag, Berlin, Heidelberg, 1993.

[SS94] Ravi S. Sandhu and Pierangela Samarati. Access control: Principle and practice. *IEEE Communications Magazine*, 32(9):40–48, Sept 1994.

[SSB14] Amina Saâdaoui, Nihel Ben Youssef Ben Souayeh, and Adel Bouhoula. Formal approach for managing firewall misconfigurations. In *2014 IEEE Eighth International Conference on Research Challenges in Information Science (RCIS)*, 2014.

[SSJP17] Pradip Kumar Sharma, Saurabh Singh, Young-Sik Jeong, and Jong Hyuk Park. DistBlockNet: A distributed blockchains-based secure SDN architecture for IoT networks. *IEEE Communications Magazine*, 55(9):78–85, 2017.

[ST09] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, 4(2):14:1–14:42, May 2009.

[Sta07] Mariusz Stawowski. The principles of network security design. *ISSA*, 2007.

[Sta09] Mariusz Stawowski. Network security architecture. *ISSA*, 2009.

[TA15] Thuy Vinh Tran and Heejune Ahn. A network topology-aware selectively distributed firewall control in SDN. In *2015 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 89–94. IEEE, 2015.

[TA16] Thuy Vinh Tran and Heejune Ahn. FlowTracker: A SDN stateful firewall

solution with adaptive connection tracking and minimized controller processing. In *2016 International Conference on Software Networking (ICSN)*, pages 1–5. IEEE, 2016.

[TAK08] J. Lane Thames, Randal Abler, and David Keeling. A distributed firewall and active response architecture providing preemptive protection. In *Proceedings of the 46th Annual Southeast Regional Conference on XX*, ACM-SE 46, pages 220–225, New York, NY, USA, 2008. ACM.

[Tho97] Roshan K. Thomas. Team-based access control (TMAC): A primitive for applying role-based access controls in collaborative environments. In *Proceedings of the Second ACM Workshop on Role-based Access Control*, RBAC '97, pages 13–19, 1997.

[TIS07] Alok Tongaonkar, Niranjan Inamdar, and R. Sekar. Inferring higher level policies from firewall rules. In *Proceedings of the 21st Conference on Large Installation System Administration Conference*, 2007.

[TS98] Roshan Thomas and Ravi Sandhu. Task-based authorization controls (TBAC): A family of models for active and enterprise-oriented authorization management. In T. Y. Lin and Shelly Qian, editors, *Database Security XI: Status and Prospects*, pages 166–181, Boston, MA, 1998. Springer US.

[TS06] Andrew S. Tanenbaum and Maarten Van Steen. *Distributed Systems: Principles and Paradigms*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[TW10] Andrew S. Tanenbaum and David J. Wetherall. *Computer Networks*. Pearson, 5th edition edition, 2010.

[U.S16] U.S. Department of Homeland Security. *Recommended Practice: Improving*

Industrial Control Systems Cybersecurity with Defense-in-Depth Strategies, September 2016.

[VE05] John R. Vacca and Scott R. Ellis. *Firewalls Jumpstart for Network and Systems Administrators*. Elsevier, 2005.

[VSB+15] Fulvio Valenza, Serena Spinoso, Cataldo Basile, Riccardo Sisto, and Antonio Lioy. A formal model of network policy analysis. In *Research and TechnolIEEE 1st International Forum on Research and Technologies Society and Industry Leveraging a better tomorrow (RTSI)*, pages 516–522, Sep 2015.

[WCLC06] Weiping Wang, Wenhui Chen, Zhepeng Li, and Huaping Chen. *Comparison Model and Algorithm for Distributed Firewall Policy*, pages 545–556. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[WJ03] Duminda Wijesekera and Sushil Jajodia. A propositional policy algebra for access control. *ACM Trans. Inf. Syst. Secur.*, 6(2):286–325, May 2003.

[WLGX16] Tao Wang, Fangming Liu, Jian Guo, and Hong Xu. Dynamic SDN controller assignment in data center networks: Stable matching with transfers. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9, 2016.

[WLWF15] Zhenfang Wang, ZhiHui Lu, Jie Wu, and Kang Fan. CPFirewall: A novel parallel firewall scheme for FWaaS in the cloud environment. In Lina Yao, Xia Xie, Qingchen Zhang, Laurence T. Yang, Albert Y. Zomaya, and Hai Jin, editors, *Advances in Services Computing*, pages 121–136, Cham, 2015. Springer International Publishing.

[WNJ06] Lingyu Wang, Steven Noel, and Sushil Jajodia. Minimum-cost network hardening using attack graphs. *Comput. Commun.*, 29:3812–3824, 2006.

[WŞPS19] Neal Wagner, Cem Ş. Şahin, Jaime Pena, and William W. Streilein. Automatic generation of cyber architectures optimized for security, cost, and mission performance: A nature-inspired approach. In Nagar A. Shandilya S., Shandilya S., editor, *Advances in Nature-Inspired Computing and Applications*. Springer, Cham, 2019.

[WŞŞP+17] Neal Wagner, Cem Ş. Şahin, Jamie Pena, James Riordan, and Sebastian Neumayer. Capturing the security effects of network segmentation via a continuous-time markov chain model. In *Proceedings of the 50th Annual Simulation Symposium*, ANSS '17, pages 1–12, San Diego, CA, USA, 2017. Society for Computer Simulation International.

[WŞW+16] Neal Wagner, Cem Ş. Şahin, Michael Winterrose, James Riordan, Jaime Pena, Diana Hanson, and William W. Streilein. Towards automated cyber decision support: A case study on network segmentation for security. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–10, Dec 2016.

[WŞW+17] Neal Wagner, Cem Ş Şahin, Michael Winterrose, James Riordan, Diana Hanson, Jaime Peña, and William W Streilein. Quantifying the mission impact of network-level cyber defensive mitigations. *The Journal of Defense Modeling and Simulation*, 14(3):201–216, 2017.

[Wu01] Rong Wu. A tool for consistency verification and for integration of formal relational requirements scenarios. Master's thesis, McMaster University, 2001.

[WWF13] Randy Weaver, Dawn Weaver, and Dean Farwood. *Guide to network defense and countermeasures*. Cengage Learning, 2013.

[YB10] Nihel Ben Souayeh Ben Youssef and Adel Bouhoula. Automatic conformance verification of distributed firewalls to security requirements. In *IEEE Second International Conference on Social Computing (SocialCom), 2010*, pages 834–841, Aug 2010.

[YBÖE17] Barış Yamansavaşçılar, Ahmet Cihat Baktır, Atay Özgövde, and Cem Ersoy. Fault tolerant data plane using SDN. In *2017 25th Signal Processing and Communications Applications Conference (SIU)*, pages 1–4, 2017.

[YCM+06] Lihua Yuan, Hao Chen, Jianning Mai, Chen-Nee Chuah, Zhendong Su, and Prasant Mohapatra. FIREMAN: a toolkit for firewall modeling and analysis. In *IEEE Symposium on Security and Privacy*, pages 15 pp.–213, May 2006.

[YQR17] Hans C. Yu, Giorgio Quer, and Ramesh R. Rao. Wireless SDN mobile ad hoc network: From theory to practice. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–7, 2017.

[Zal10] Vadim Zaliva. Firewall policy modeling, analysis and simulation: a survey, 2010.

[ZASJ+07] Bin Zhang, Ehab Al-Shaer, Radha Jagadeesan, James Riely, and Corin Pitcher. Specifications of a high-level conflict-free firewall policy language for multi-domain networks. In *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies*, SACMAT '07, pages 185–194, New York, NY, USA, 2007. ACM.

[ZB07] Hang Zhao and Steven M. Bellovin. Policy algebras for hybrid firewalls.

Technical report, Department of Computer Science, Columbia University, March 2007.

[ZE18] Ali Zeineddine and Wassim El-Hajj. Stateful distributed firewall as a service in SDN. In *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pages 212–216, 2018.

[ZJXsLx09] Lin Zhi, Wang Jing, Chen Xiao-su, and Jia Lian-xing. Research on policy-based access control model. In *International Conference on Networks Security, Wireless Communications and Trusted Computing*, volume 2, 2009.

[ZK16] Qinglei Zhang and Ridha Khedri. On the weaving process of aspect-oriented product family algebra. *Journal of Logical and Algebraic Methods in Programming*, 85(1, Part 2):146 – 172, January 2016. Special Issue on Formal Methods for Software Product Line Engineering.

[ZKJ12] Qinglei Zhang, Ridha Khedri, and Jason Jaskolka. Verification of aspectual composition in feature-modeling. In G. Eleftherakis, M. Hinchey, and M. Holcombe, editors, *Software Engineering and Formal Methods*, volume 7504 of *Lecture Notes in Computer Science*, pages 109 – 125. Springer Berlin / Heidelberg, 2012.

[ZKJ14] Qinglei Zhang, Ridha Khedri, and Jason Jaskolka. An aspect-oriented language for feature-modeling. *Journal of Ambient Intelligence and Humanized Computing*, 5:343 – 356, 2014.

[ZKS+19] Liehuang Zhu, Md Monjurul Karim, Kashif Sharif, Fan Li, Xiaojiang Du, and Mohsen Guizani. Sdn controllers: Benchmarking & performance evaluation. *arXiv preprint arXiv:1902.04491*, 2019.

[ZLB08] Hang Zhao, Jorge Lobo, and Steven M. Bellovin. An algebra for integration

and analysis of ponder2 policies. In *2008 IEEE Workshop on Policies for Distributed Systems and Networks*, pages 74–77, June 2008.

[ZLKM11] Vladimir Zaborovsky, Alexey. Lukashin, Sergey Kupreenko, and Vladimir Mulukha. Dynamic access control in cloud services. In *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 1400–1404, Oct 2011.

Glossary

A

ABAC Attribute Based Access Control. 21, 29, 33, 35, 41, 69

AC attribute certificate. 30

ACL Access Control List. 20, 22, 29

AO-PFA Aspect-Oriented Product Family Algebra. 182

API application programming interface. 42, 43, 192

ARBAC Administrative RBAC. 22, 33

C

CLI Command-line interface. 46, 175

D

DAC Discretionary Access Control. 19, 20

DCG Dynamic Configuration and Governance. 155, 156, 157, 158, 160, 161, 163, 168, 169, 175, 176, 177

DD Defense in Depth. 13, 14, 17, 24, 25, 32, 86, 88, 89, 91, 92, 93, 94, 95, 100, 104, 106, 109, 138, 153, 180, 185, 191

DDL Defense in Depth Law. 88

DMZ Demilitarized Zone. 11

DNS Domain Name System. 6

DSPL dynamic software product lines. 27

E

Exp-RNS Exponential Robust Network and Segmentation Algorithm. 111, 123, 124, 137

F

FAST Flow-level State Transitions. 44

FQA functional quality attributes. 27, 181

FTP File Transfer Program. 6

G

GCD Greatest Common Divisor. 63, 69, 71, 77, 78, 98, 108, 111, 112, 113, 114, 128, 129, 133, 134, 138, 158, 193, 196, 201, 202, 203

H

HR Human Resources. 142, 143

HTTP HyperText Transfer Protocol. 6, 8, 142

I

IDS Intrusion Detection Systems. 24, 182

IoT Internet of Things. 10, 182

IP Internet Protocol. 6, 8, 40, 56, 60

IPS Intrusion Prevention Systems. 24

IPsec IP Security. 7, 8

M

MAC Mandatory Access Control. 19, 20, 27, 30

MANET Mobile Adhoc Network. 46

N

NFV Network Function Virtualization. 10

NOS network operating system. 43

NSR Non-slicing of Shared Resources. 145, 147, 149, 151, 152, 153, 154, 176, 177

O

ORBAC Organization Access Control. 21

OSI Open Systems Interconnection. 6

P

PBAC Policy-Based Access Control. 21

PFA Product Family Algebra. 11, 12, 13, 17, 49, 64, 68, 69, 70, 71, 76, 77, 79, 92, 106, 179, 181

PKC public key certificate. 30

PRC Policy Requirement Constraint. 77, 94

PVS Prototype Verification System. 193, 200, 202, 203

R

RAdAC Risk-Adaptive Access Control. 21

RBAC Role-Based Access Control. 21, 27, 28, 29, 30, 32, 35, 36, 40, 41

RMI Remote Method Invocation. 192

RNS Robust Network and Segmentation Algorithm. xv, 124, 134, 135, 136, 137, 138, 139, 147, 150, 152, 153, 155, 156, 157, 158, 159, 160, 163, 168, 169, 175, 176, 177, 178, 180, 181, 182

RPC Remote Procedure Call. 192

S

SDD Strict Defense in Depth. 14, 15, 17, 95, 96, 97, 98, 99, 100, 101, 109, 110, 111, 123, 134, 135, 180

SDDL Strict Defense in Depth Law. 95

SDN Software Defined Networking. 10, 11, 12, 14, 15, 16, 17, 18, 31, 33, 41, 42, 43, 45, 46, 48, 155, 156, 157, 158, 168, 169, 176, 177, 178, 179, 181

SDPA Stateful Data Plane Architecture. 44

SMTP Simple Mail Transfer Protocol. 6, 8, 142

SPL software product line. 27

SSR Slicing of Shared Resources. 149, 150, 151, 152, 153, 154, 176, 177

T

TBAC Task-Based Authorization Controls. 22

TCP Transmission Control Protocol. 6, 8, 9

TMAC Team-Based Access Control. 22

TRBAC Temporal RBAC. 22

U

UDP User Datagram Protocol. 6

UPF Unified Policy Framework. 31, 32

V

VM Virtual Machines. 30

VMM Virtual Machine Monitor. 30, 31

VPN Virtual Private Network. 142

VRBAC Virtualized RBAC. 22

W

WMN Wireless Mesh Network. 46

WSN wireless sensor networks. 27

X

XFSM eXtensible Finite State Machine. 44, 167, 168