Reasoning About Firewall Policies Through Refinement and Composition

Ultan James Neville
B.Sc. (Hons.), M.Sc.

Thesis submitted for the degree of Doctor of Philosophy

NATIONAL UNIVERSITY OF IRELAND, CORK

Faculty of Science
Department of Computer Science

February 2017

Head of Department: Prof. Cormac J. Sreenan

Supervisor: Dr. Simon N. Foley

"If in other sciences we should arrive at certainty without doubt and truth without error, it behooves us to place the foundations of knowledge in mathematics." - **Roger Bacon**.

In memory of my Dad, John, and for my Mam, Bridget. My success is your success.

Contents

List of Figures

List of Tables

List of Theorems

I, Ultan James Neville, certify that this thesis is my own work and has not been submitted for another degree at University College Cork or elsewhere.

Ultan James Neville

Abstract

Network and host-based access controls, for example, firewall systems, are important points of security-demarcation, operating as a front-line defence for networks and networked systems. A firewall policy is conventionally defined as a sequence of order-dependant rules, and when a network packet matches with two or more policy rules, the policy is anomalous. Policies for access-control mechanisms may consist of thousands of access-control rules, and correct management is complex and error-prone. Policies may need to be reconfigured for highly dynamic environments, as threats to, and access requirements for, resources behind a firewall do not usually remain static. Misconfiguration is common, and correct policy management is often reliant on the expert-knowledge of security administrators, and drawing from best practice.

The thesis of this dissertation is that a firewall policy should be anomaly-free by construction, and as such, there is a need for a firewall policy language that allows for constructing, comparing, and composing anomaly-free policies. An algebra is proposed for constructing and reasoning about anomaly-free firewall policies. Based on the notion of refinement as safe replacement, the algebra provides operators for sequential composition, union and intersection of policies. The algebra allows a policy specifier to compose policies in such a way, that the result of the composition upholds the access requirements of each policy involved, and allows one to reason as to whether some policy is a safe (secure) replacement for another policy.

This approach enables a common framework, whereby knowledge related to detailed access control configurations and standards-based firewall policies can be represented and reasoned about. This dissertation explores the effectiveness of firewall policy specification and analysis, that extends the conventional five-tuple rule to include stateful inspection, TCP flags, ICMP Types/Codes, and additional filter condition attributes. The effectiveness of the algebra is demonstrated by its application to anomaly detection, and standards compliance.

The effectiveness of the approach in practice is evaluated through a mapping to/from iptables. The evaluation shows that the approach is practical for large policies. The effectiveness is also evaluated through a mapping to OpenStack network and host-based access controls, and the development of a policy management framework for the Android OS.

Acknowledgements

I would like to begin by sincerely thanking my supervisor, Dr. Simon Foley, for providing me with the opportunity to undertake my doctoral research. Your perpetual motivation, guidance and friendship over the last few years have been invaluable and inspiring. You are an exceptional mentor and a Cyber Security Jedi. I will be forever grateful, I can't imagine having had a better supervisor.

I wish to also thank my viva voce examiners, Prof. Joaquín García-Alfaro and Dr. John Herbert, for a rigorous, but enjoyable three-hour examination.

I am extremely grateful for the genuine support shown to me by my friends and fellow alumni of the University College Cork Security Group, namely Dr. Vivien Rooney, Dr. Olgierd Pieczul, Dr. Jonathan Petit and in particular, Dr. William Fitzgerald.

I would also like to express my gratitude and thanks to my colleagues and friends in the Department of Computer Science, in particular to Cathal Hoare and to past members Dr. Hazzaa Alsharif, Ashruff Mahadi, Andrew Hobbs and JJ Earls. This thanks is also extended to the friends I've made during my BSc and MSc studies in University College Cork.

I am very grateful to Caitríona Walsh, Eleanor O'Riordan, Peter McHale and Mary Noonan for their administrative support throughout the course of my research. I would like to extend this gratitude to Prof. Barry O'Sullivan for having me as a member of Cork Constraint Computation Centre (4C) and the Insight Centre for Data Analytics, and to Prof. Cormac Sreenan for having me as a member of the Mobile and Internet Systems Laboratory (MISL), while reading for my PhD. I wish to also acknowledge that my research would not have been possible without the financial support provided through the Federated, Autonomic Management of End-to-end communication services (FAME) and the Centre for Telecommunications Value-Chain-Driven Research (CTVR) projects.

A special acknowledgement is reserved for the expats - Colum Brosnan and George Boyle. Our friendships have endured both distance and time, thank you both for the moral support.

I must take this opportunity to thank my family for their love and support also. My Dad, John, who passed away during the course of my studies, you are forever in my thoughts. My Mam, Bridget, for everything you have done for me. My sister, Aisling, for the encouragement that I needed. You are the people I have counted on, and you were always there for me.

Last, but not least, a special thank you to my partner Jayenthrie - your love and support in recent years have made dark days bright.

Part I

Background and Review

Chapter 1

Introduction

1.1 Motivation

Network access control mechanisms, for example, firewalls, Virtual Private Networks (VPN) and Intrusion Prevention Systems (IPS), are configured in accordance with high-level security requirements, and play an important role in provisioning the security for networks and networked systems. While VPNs provide end-to-end security for the packets traversing a network, the use of firewalls as a front-line defence against unwanted network traffic is widespread. Firewalls are used, for example, to mitigate network-based threats for enterprise, research and home networks, and may be deployed throughout a network configuration and at the end-points. For end-users, software-based firewalls come as standard with most of the popular operating systems. In this chapter, we develop a simple running example to illustrate challenges and complexities in managing a firewall policy configuration, and present our approach.

1.1.1 Challenges to Effective Firewall Policy Management

A firewall enforces a policy, and a policy may comprise thousands of low-level and order-dependant firewall rules. As a consequence, policy management is complex and error-prone, and misconfiguration is common. A policy, or distributed policy configuration, may be updated on an ad-hoc basis, possibly by multiple administrators. This can be problematic and may introduce anomalies, whereby the intended semantics of the specified access controls become ambiguous. A misconfiguration may be inconsistent with the high-level security requirements, resulting in accesses that were intended to be denied being permitted, and/or vice-versa. There are a number of approaches available to an administrator for managing a firewall policy configuration. For example, the goal of [6, 30, 37, 66, 159] is to provide an administrator with the means to detect/resolve anomalies, and work such as [47, 52, 88, 95] allows for querying a policy configuration with regard to the filtering of specific network traffic. High-level specification languages such as [3, 13, 39, 72, 84] allow an administrator to abstractly specify what would otherwise be low-level rules. However, in general, the literature for policy management is focused on the conventional five-tuple firewall rule with a binary target action of *allow* or *deny*, and few have considered stateful firewall configurations.

1.1.2 Anomaly-free Firewall Policy Composition

A firewall policy may be developed as a collection of independent or related specifications that an administrator will need to replace by a policy that adequately captures the requirements of the individual specifications. A configuration may need to be updated with additional policy specifications when a new threat is identified, or when a new permissible access is required. Therefore, having a consistent means of composing these specifications is desirable. The objective of this dissertation is to develop a theory about composing anomaly-free firewall policies. When a policy is anomaly-free, there is no ambiguity as to whether a given network packet is allowed or denied by the firewall. Having a consistent means of anomaly-free firewall policy composition enables a means of anomaly-free, dynamic firewall policy reconfiguration, and thus, this is our goal.

Example 1 When configuring the rules that define a firewall policy, the specifier must understand the relationship of each rule to every other rule in the policy. Consider, as a running example, a company that employs a team of administrators and a team of developers. There are two network security policy requirements, whereby network traffic destined to the IP range $[1..3]$ on ports $[1..3]$ is to be allowed from the administrators, and traffic destined to the IP range $[2..4]$ on ports $[2..4]$ is to be allowed from the developers. For ease of exposition, we give the IPs as natural numbers, and the IP and port ranges as intervals of $\mathbb{N}$. For simplicity, we do not consider the source IP ranges for the administrators and the developers.

Specifying The Requirements. System Administrator *Bob* manages the network access controls for the administration and development teams. He specifies the network security policy requirement for the administration team as follows.

Index	Dst IP	Dst Port	Action
1	$[1..3]$	$[1..3]$	*allow*

$\mathsf{Pol}_{\mathsf{Admin}}$

Similarly, for the development team, he specifies:

Index	**Dst IP**	**Dst Port**	**Action**
1	$[2 \mathinner{..} 4]$	$[2 \mathinner{..} 4]$	*allow*

$\mathsf{Pol}_{\mathsf{Dev}}$

Bob needs to combine $\mathsf{Pol}_{\mathsf{Admin}}$ and $\mathsf{Pol}_{\mathsf{Dev}}$ into a single firewall policy, and security requirements may change. He requires a consistent means of composing firewall policies, whereby the result is anomaly-free and upholds the enforcements of each policy involved in the composition.

Sequential Composition. To specify the firewall policy for the company, he tries sequentially composing $\mathsf{Pol}_{\mathsf{Admin}}$ and $\mathsf{Pol}_{\mathsf{Dev}}$ as follows.

Index	**Dst IP**	**Dst Port**	**Action**
1	$[1 \mathinner{..} 3]$	$[1 \mathinner{..} 3]$	*allow*
2	$[2 \mathinner{..} 4]$	$[2 \mathinner{..} 4]$	*allow*

$\mathsf{Pol}_{\mathsf{Admin}} \frown \mathsf{Pol}_{\mathsf{Dev}}$

This approach, however, does not does yield the desired result. That is, in the above specification ($\mathsf{Pol}_{\mathsf{Admin}} \frown \mathsf{Pol}_{\mathsf{Dev}}$), the rule at Index 1 allows some of the IP/-port pairs allowed by the rule at Index 2 and vice-versa. Therefore, the policy is anomalous. From this, we have that the naïve sequential composition of rules is not a consistent operation when specifying an anomaly-free policy.

A Flattening Approach. To specify the firewall policy, Bob considers the approach whereby for each IP/port pair allowed by the company's network security policy requirements, there is a rule to allow each IP/port pair. He specifies:

Index	**Dst IP**	**Dst Port**	**Action**
1	1	1	*allow*
2	1	2	*allow*
3	1	3	*allow*
..			
14	4	4	*allow*

This policy does provide the desired result, in that it is both anomaly-free and consistent with the two network security policy requirements involved in the composition. We observe however, that this approach is tedious, error-prone and not practical for large numbers of IPs/ports or large numbers of policy rules.

A Different Approach. Bob is required to specify the firewall policy whereby policy rules allow the IP/port-range pairs from either $\mathsf{Pol}_{\mathsf{Admin}}$ or $\mathsf{Pol}_{\mathsf{Dev}}$. To specify the policy for the company, he decides to compose the two requirements into a single requirement as follows.

Index	Dst IP	Dst Port	Action
1	$[1\mathinner{.\,.}4]$	$[1\mathinner{.\,.}4]$	*allow*

This approach, however, is inconsistent with the two network security policy requirements outlined by the company. That is, the result of composition is an overly-permissive firewall policy, whereby network traffic is permitted to IP $\mathsf{1}$ on port $\mathsf{4}$, and to IP $\mathsf{4}$ on port $\mathsf{1}$. Conversely, this approach would result in an overly-restrictive policy if the rules had a target action of *deny*.

A Better Approach. Bob is required to specify the firewall policy whereby the desired result is the smallest number of anomaly-free rules that allow all the IP/port pairs from either $\mathsf{Pol}_{\mathsf{Admin}}$ or $\mathsf{Pol}_{\mathsf{Dev}}$. He specifies the policy:

Index	Dst IP	Dst Port	Action
1	$[1\mathinner{.\,.}4]$	$[2\mathinner{.\,.}3]$	*allow*
2	$[1\mathinner{.\,.}3]$	$[1\mathinner{.\,.}1]$	*allow*
3	$[2\mathinner{.\,.}4]$	$[4\mathinner{.\,.}4]$	*allow*

In this case, the result is as desired, as it is the smallest number of anomaly-free rules that allow all the IP/port pairs from either $\mathsf{Pol}_{\mathsf{Admin}}$ or $\mathsf{Pol}_{\mathsf{Dev}}$.

Mutual Policy Enforcements. Bob receives a request to configure the policy that allows the IP/port pairs from both $\mathsf{Pol}_{\mathsf{Admin}}$ and $\mathsf{Pol}_{\mathsf{Dev}}$. He specifies:

Index	Dst IP	Dst Port	Action
1	$[2\mathinner{.\,.}3]$	$[2\mathinner{.\,.}3]$	*allow*

In this case, the specification provides the desired result, and defines the smallest number of anomaly-free rules that allow all the IP/port pairs from both $\mathsf{Pol}_{\mathsf{Admin}}$ and $\mathsf{Pol}_{\mathsf{Dev}}$. We observe that the resulting policy upholds the restrictions of both $\mathsf{Pol}_{\mathsf{Admin}}$ and $\mathsf{Pol}_{\mathsf{Dev}}$. △

For an administrator, it may be relatively straightforward to understand policy composition where only a small number of rules are involved, however, this does not scale. We argue that to reason confidently a policy or distributed policy configuration is anomaly-free and adequately mitigates the identified threats; a common framework is required, whereby knowledge related to detailed access control configurations and standards-based firewall policies can be represented and reasoned about.

1.2 Research Approach

The thesis of this dissertation is that *a firewall policy should be anomaly-free by construction, and as such, there is a need for a firewall policy language that allows for constructing, comparing, and composing anomaly-free policies.*

1.2.1 An Algebra for Firewall Policies

In this dissertation, a formal model for network access control policies is proposed. It is important to have a consistent means of comparing firewall policies, where we can give a precise meaning to the notion of one firewall policy being superior to another firewall policy, with respect to the property of restrictiveness. Therefore, it is necessary to define an ordering over firewall policies. An administrator may need to develop a policy as a collection of independent or related specifications, that will need to be replaced by a policy that adequately captures the requirements of the individual specifications. Therefore, we require a consistent means of composing policy fragments. The proposed model is developed as a lattice structure. The lattice consists of a partially ordered set of all firewall policies, where every pair of policies have a unique *lowest upper bound* (lub) and a unique *greatest lower bound* (glb). The glb of two policies defines the most permissive replacement policy that enforces the restrictions of either policy, and the lub of two policies defines the most permissive replacement policy that enforces the restrictions of both policies. Given that firewall policies are used to specify complex access control restrictions, a significant challenge to the approach is determining a suitable lattice. We construct a generic framework for firewall policies that can be used to model policies for different network and host-based access controls. The framework is extensible, and defines an n-tuple model for firewall rule filter condition attributes that extends the conventional five-tuple rule, used for example in [1, 5, 6, 30, 35, 73, 87, 159]. The proposed framework defines a firewall policy algebra for constructing and reasoning over anomaly-free policies, and provides sound and consistent lattice operators for policy composition. Based on the notion of refinement as safe replacement [56, 57, 77], the algebra provides operators for sequential composition, union and intersection of policies. In this dissertation, when one policy is considered a safe (secure) replacement for another, then this means that the former is no less restrictive than the latter. The effectiveness of the algebra is demonstrated by its application to anomaly detection, and standards compliance.

Constructing The Policy Framework. We develop a model to describe what it means to specify, compare and compose anomaly-free policies. The core filter condition attributes for firewall rules in the model define a partial mapping of the iptables `filter` table. The specification defines formal constructs for filtering at various OSI network layers. To demonstrate the utility of developing an algebra for firewall policies supporting rules with complex range-based constraints, we firstly specify a simple model of firewall rules; $\mathcal{FW}_0$, whereby policies are defined in terms of constraints on individual IP addresses, ports, protocols and additional filter condition attributes. The proposed algebra $\mathcal{FW}_0$ provides a semantics for firewall policies. While useful for the purposes of reasoning, it is not efficient to naïvely implement the algebra, since a policy is defined in terms of sets of rules constraining *individual* IP addresses and ports. For example, a policy constraining access from a subnet range `172.16.*.*` involves more than 65K individual packet rules, whatever about the impact of combining these with further constraints on destination IPs and ports. Modelling a firewall rule as a predicate that specifies the set of packets for which the rule evaluates to *true* requires the conjunction/disjunction of predicates, or the union/intersection of the sets of packets when composing firewall rules. Intersection and union operations on sets of packets may be computationally-expensive due to the possible number of network packets involved. In practice, firewall rules are defined in terms of range-based attributes, and policy rules may match large numbers of packets.

Subsequently, we develop a model $\mathcal{FW}_1$, that defines a firewall policy in terms of stateful and stateless firewall rules constraining range-based versions of the filter condition attributes used in the simple model. We argue that there is an isomorphic mapping between policies in the $\mathcal{FW}_0$ and $\mathcal{FW}_1$ firewall algebras. A benefit of developing the $\mathcal{FW}_1$ algebra is that we can reason about and compose firewall policies comprising rules constraining range-based attributes, for example ranges of IP addresses and ports, without having to map the rules to individual packets, as this is not practical. We show (in both models), that the set of policies form a lattice under safe replacement, and this enables consistent operators for safe composition to be defined. Policies in each lattice are anomaly-free by construction, and the property of anomaly-freedom is upheld as a result of composition under greatest lower bound and lowest upper bound operators. While policies are anomaly-free by construction, we can however define anomalies algebraically; by considering how a policy changes when composed with other policies. The policy ordering relation allows one to reason whether one policy is a safe replacement for another, and as such, we demonstrate its use as a means to test for policy standards compliance. Firewall rules in $\mathcal{FW}_0$ and $\mathcal{FW}_1$ are defined in terms of a binary target action of *allow* or *deny*. However, an extension to incorporate an additional firewall rule target action of *log* is described as part of future work.

Evaluating The Policy Framework. We consider a number of different areas and demonstrate the effectiveness of the approach in practice. A prototype policy management toolkit that implements $\mathcal{FW}_1$ firewall policies for iptables is developed, and experiments are conducted for policy operators. Overall, the results are promising. The cloud computing paradigm has become widely adopted, however, there are security challenges. The OpenStack [63] cloud operating system avails of multiple access control policies of varying types for a firewall deployment, and the environment is highly dynamic due to platform and service migration. A model is developed for OpenStack network and host-based access control policies. Devices such as smartphones operate in mobile network environments and deploying a fixed security configuration for a global set of threats is not practical. Android [141] is used in a variety of domains, from personal devices, to the Internet of Things, to enterprise, medical and military domains. However, configuration of Android security mechanisms, for example, the firewall, and the allocation of system-level permissions to requesting applications, is typically performed by non-technical end-users. The threat-based model developed as part of earlier work [55] is extended to include knowledge about the Android permission model, and a system policy algebra $Android_{Sys}$ is proposed for the Android OS. We show how the compliance-driven threat-based model can be used in conjunction with the $Android_{Sys}$ framework to dynamically manage the anomaly-free security configuration of Android firewall policies and Android permission polices.

Catalogues of Best Practice. Compliance with best practice standards and recommendations allows one to reason confidently that a policy mitigates the identified threats. For example, RFC 5735 [33] recommends countermeasures that mitigate the threat of IP address spoofing from the IANA-assigned specialized/global-purpose IPv4 address blocks. Best practice standards, including [79, 123, 125, 133, 152] for firewall access control have been encoded as part of this work as collections of iptables rules. Excerpts of these catalogues are illustrated in Appendix B, Tables B.1, B.2, B.3, B.4 and B.5. Compliance policies are defined for use in the $\mathcal{FW}_1$ algebra in Chapter 5 and Chapter 8. In Chapter 8 we also consider policy compliance for permission policies on Android systems. A best practice catalogue extract for permission policies from [112] is illustrated in Appendix B Table B.6. The knowledge-base of defined compliance policies is used to describe the dynamic synthesis of security configurations for Android.

1.3 Contributions

The contributions within this dissertation are summarised as follows.

- The primary contribution of this dissertation is a firewall policy algebra $\mathcal{FW}_1$, in which anomaly-free firewall policies can be specified and reasoned about. The algebra $\mathcal{FW}_1$ is defined over stateful and stateless firewall policies constructed in terms of constraints on source/destination IP/port ranges, the TCP, UDP and ICMP protocols, and additional filter condition attributes. $\mathcal{FW}_1$ is a generic firewall algebra that can be used to model different firewall systems, and an n-tuple firewall rule filter condition specification is supported by the model.

- A partial mapping of the iptables filter table. This mapping is used to define the filter condition constraints for the firewall policy models.

- A policy model $\mathcal{FW}_{OpenStack}$ is defined for OpenStack firewall policies using a derivation of $\mathcal{FW}_1$. $\mathcal{FW}_{OpenStack}$ provides a uniform way to specify and reason about OpenStack host-based and network access controls. In particular, it gives a meaning for OpenStack security group policies and perimeter firewall policies.

- A policy management framework for Android. The framework amalgamates a threat-based model that represents catalogues of best practice standards and a policy algebra $Android_{Sys}$ for the Android OS. The $Android_{Sys}$ algebra incorporates $\mathcal{FW}_1$, and a simple algebra $Android_{Perm}$ for Android permission policies. The framework defines a model of dynamic security control reconfiguration for Android.

Publications. Early versions of the results in this dissertation have been reported in peer-reviewed publications.

- U. Neville and S.N. Foley. Reasoning About Firewall Policies Through Refinement and Composition [1]. In *Data and Applications Security and Privacy XXX: 30th Annual IFIP WG 11.3 Conference, DBSec 2016, Trento, Italy, July 18-20, 2016. Proceedings*, 2016.
- S.N. Foley and U. Neville. A Firewall Algebra for OpenStack. In *2015 IEEE Conference on Communications and Network Security, CNS 2015, Florence, Italy, September 28-30, 2015, pages* 541–549. IEEE, 2015.
- W.M. Fitzgerald, U. Neville, and S.N. Foley. MASON: Mobile Autonomic Security for Network Access Controls. *Journal of Information Security and Applications (JISA)*, 18(1):14–29, 2013.
- W.M. Fitzgerald, U. Neville, and S.N. Foley. Automated Smartphone Security Configuration. In *Data Privacy Management and Autonomous Spontaneous Security, 7th International Workshop, DPM 2012, and 5th International Workshop, SETOP 2012, Pisa, Italy, September 13-14, 2012. Revised Selected Papers*, pages 227-242, 2012.

1.4 Layout of Dissertation

The remainder of this dissertation is structured as follows.

- **Chapter 2, Network Access Control and Policy Management.** In this chapter, the background research is described, followed by a detailed analysis of related work.
- **Chapter 3, Attributes of a Linux-based Firewall.** We develop a formal model for various filter condition attributes of the Linux iptables firewall. The attributes are used to construct packet-rules for the $\mathcal{FW}_0$ firewall policy algebra in Chapter 4, and are extended in Chapter 5 to construct range-based rules for the $\mathcal{FW}_1$ algebra.
- **Chapter 4, The $\mathcal{FW}_0$ Policy Model.** A firewall policy algebra $\mathcal{FW}_0$ is proposed for constructing and reasoning over anomaly-free policies. Firewall

[1] An extended version of this paper is currently being prepared for an invited selected publication in the Journal of Computer Security (JCS IOS Press).

rules in $\mathcal{FW}_0$ are constructed using the filter condition attributes defined in Chapter 3. $\mathcal{FW}_0$ defines a simple model of iptables rules that do not consider range-based filter condition attributes; the purpose of developing $\mathcal{FW}_0$ is to demonstrate the utility of building such an algebra.

- **Chapter 5, The $\mathcal{FW}_1$ Policy Model.** We develop a firewall policy algebra $\mathcal{FW}_1$, for constructing and reasoning over anomaly-free policies. The policy algebra $\mathcal{FW}_1$ defines a firewall policy in terms of rules constraining range-based versions of the filter condition attributes defined in Chapter 3.

- **Chapter 6, Implementing The $\mathcal{FW}_1$ Policy Algebra.** In this chapter, a prototype policy management toolkit that implements the $\mathcal{FW}_1$ $\mathit{Policy}_{\mathcal{I}}$ firewall policies defined in Chapter 5 Section 5.4.1 for iptables is described. Experiments for policy operators are conducted and the results are reported.

- **Chapter 7, A Firewall Policy Algebra For OpenStack.** A firewall policy algebra $\mathcal{FW}_{\mathit{OpenStack}}$ for OpenStack is proposed. $\mathcal{FW}_{\mathit{OpenStack}}$ is a derivation of the $\mathcal{FW}_1$ algebra. We use the algebra $\mathcal{FW}_{\mathit{OpenStack}}$ to provide a uniform way to specify and reason about OpenStack host-based and network access controls. A case study OpenStack deployment illustrates practical use of the algebra.

- **Chapter 8, A Policy Management Framework For Android.** A policy management framework for the Android OS is developed. A case study deployment illustrates practical use of the $\mathcal{FW}_1$ algebra on Android systems. A system policy algebra $\mathit{Android}_{\mathit{Sys}}$ is proposed. $\mathit{Android}_{\mathit{Sys}}$ uses $\mathcal{FW}_1$, in conjunction with an algebra $\mathit{Android}_{\mathit{Perm}}$, to manage Android firewall policies and Android permissions. The policy management framework incorporates a threat-based model with $\mathit{Android}_{\mathit{Sys}}$ to enable a model of dynamic security control reconfiguration for Android.

- **Chapter 9, Conclusion and Future Research.** This chapter concludes the dissertation and considers some future research that may be undertaken.

The Z notation [135] is used to provide a consistent syntax for structuring and presenting the definitions and examples in this dissertation. We use only those parts of Z that can be intuitively understood and Appendix A gives a brief overview of the notation used. Mathematical definitions have been syntax- and type-checked using the $f$uzz tool [134].

Chapter 2

Network Access Control and Policy Management

This chapter examines the background research for the thesis and related work. In Section 2.1 we consider network and host-based access control through the concepts of the firewall and firewall policy. A classification for firewalls, and the various mitigations they provide in terms of a collection of known network threats is considered. Section 2.2 explores conflict resolution in access control policies. In Section 2.3, current firewall policy management practice is reviewed, and we consider how the work in this dissertation relates to several of the firewall policy management approaches currently available. In Section 2.4 we consider some of the challenges associated with firewall policy composition.

2.1 The Firewall and Firewall Policy

A firewall is an important network security mechanism, used to regulate network access control; it operates as a front-line defence for networks and networked systems. Cheswick et al. [27] give one of the earliest definitions of a firewall as: *"a collection of components placed between two networks that collectively have the following properties:*

1. *All traffic from inside to outside, and vice-versa, must pass through the firewall.*

2. *Only authorized traffic, as defined by the local security policy, will be allowed to pass.*

3. *The firewall itself is immune to penetration."*

In later work, Cheswick et al. describe a firewall as *"any software, device, or arrangement or equipment that limits network access"* [28].

Security Policy. A *security policy*, for example, the Payment Card Industry Data Security Standard (PCI-DSS) [34], is a high-level document that defines a *"set of rules and practices that specify or regulate how a system or organization provides security services to protect sensitive and critical system resources"* [132]. Table 2.1 illustrates an overview of the policy requirements from the PCI-DSS.

Build and Maintain a Secure Network	
Requirement ID	**Requirement**
req-1	Install and maintain a firewall configuration to protect cardholder data.
. .	
req-1.1.2	Current network diagram with all connections to cardholder data, including any wireless networks.
. .	
req-1.2	Build firewall and router configurations that restrict connections between untrusted networks and any system components in the cardholder data environment.
. .	
req-1.4	Install personal firewall software on any mobile and/or employee-owned computers with direct connectivity to the Internet (for example, laptops used by employees), which are used to access the organization's network.

Table 2.1: PCI-DSS security policy excerpt

Network Security Policy. To administer network access control, a security policy is refined into a *network security policy* that *"describes an organisation's network security concerns and specifies the way network security should be achieved in that organisation's environment"* [105]. Table 2.2 illustrates an example excerpt of a network security policy for a company.

Policy ID	**Description**
nsp-1	Allow administrators to ping the code revision control system in the development subnet for liveness tests.
nsp-2	Allow developers FTP access to the production Web server.
nsp-3	Mitigate the threat of IP spoofing at the network perimeter, in accordance with RFC 5735.

Table 2.2: Example network security policy excerpt

Firewall Policy. A *firewall policy*, or distributed firewall policy configuration, implements a network security policy. A firewall policy is conventionally defined as a sequence of order-dependent rules. A rule is composed of filter conditions and a target action. Filter conditions usually consist of fields/attributes from IP, TCP/UDP packet headers. Table 2.3 illustrates the most commonly used filter condition attributes at the various layers of the OSI and TCP/IP network model architectures [40, 68].

OSI Model Layer	**TCP/IP Model Layer**	**Network Packet Attributes**
Application 7	Application 5	Application protocol pattern matching
Presentation 6		
Session 5	Transport 4	TCP/UDP protocols, TCP Flags, Source and destination network ports
Transport 4		
Network 3	Internet 3	Source and destination IP addresses, ICMP (Type, Code)
Data Link 2	Network Access 2	Source MAC address
Physical 1	Physical 1	

Table 2.3: Firewall filtering at different network layers

All packets traversing the firewall are filtered against policy rules. Both inbound and outbound filtering rules should be specified for resources behind the firewall [152]. Rule target actions are usually *allow*, whereby the network packet is permitted traversal of the firewall, or *deny*, whereby the packet is blocked by the firewall. Firewall rules are matched in sequence starting at the first rule, the packet is either allowed or denied when the packet header data is matched against all the filtering fields of a rule within the firewall policy. If there is no rule that matches the packet header information, a default rule is applied. The default rule can be [148]:

- **Default Deny:** *deny everything except that which is explicitly permitted.*
- **Default Allow:** *permit everything except that which is explicitly denied.*

The default deny rule is considered best practice [136, 152]. It is also considered best practice to log relevant packets for auditing purposes [152], however, a rule with a target action of *log* does not enforce the final decision to be taken on a matching packet. That is, if the packet is not allowed or denied by a matching rule at a later index, then the default rule is applied.

In this chapter, for ease of exposition, the table format depicted in Table 2.4 is used to illustrate firewall policy examples. A table row specifies a firewall rule at a given 'Index' in the policy. The wildcard value '*' matches all possible values in a given filter condition attribute, (or in a given octet in an IP address). For example, a destination IP address specifying '*.*.*.*' means all 2^{32} possible IP values.

Index	**Src IP**	**Src Prt**	**Dst IP**	**Dst Prt**	**Protocol**	**Action**
1	*.*.*.*	*	*.*.*.*	*	*	*deny*

Table 2.4: A firewall policy example

Table 2.5 gives an explanation for the table format used for firewall policy examples. Filter condition attributes from Table 2.5 are used in constructing firewall policy examples throughout this chapter. Note, policy examples will include only filter condition attributes that are relevant to the example.

Column Name	**Description**	**OSI Layer Filtered**
Index	Rule position in firewall policy	
Dir	Packet direction: inbound (ingress) or outbound (egress), with respect to the resources protected by the firewall	
Iface	Network interface on which a packet was received	
Src IP	Source IP address	Network
Dst IP	Destination IP address	Network
Type	ICMP Type	Network
Code	ICMP Code	Network
Protocol	Network protocol	Network Transport
Src Prt	Source port	Transport
Dst Prt	Destination port	Transport
L7 Filter	Packet payload pattern match	Application
Action	Target action to apply to network packet	

Table 2.5: Firewall rule filter-field description

In later chapters, we use different firewall syntaxes, where appropriate, for specifying firewall policies. For example, in Chapter 3, we introduce the iptables [146] command-line syntax, and in Chapter 7 we specify policy rules using the OpenStack [15] firewall syntax.

2.1.1 Firewall Classification

In this section, we examine the various classes of firewall, and consider their effectiveness in terms of mitigating different types of known network threats.

2.1.1.1 Packet-filter Firewall

A *packet-filter* is a firewall that enforces a target action on a network packet based only on information found in the packet header. Packet-filters have also been referred to as *stateless inspection firewalls* [123], as each packet attempting to traverse the firewall is filtered independently of all previously filtered network traffic. Filter condition attributes for policy rules enforced by early packet-filter firewalls are derived from the Network and/or Transport OSI layers [27, 105, 136]. More recent packet-filter firewalls permit an administrator to also specify policy rules that filter at the OSI Data Link layer, and filter based on the network interface a packet is arriving at/leaving through [40, 105, 138].

Example 1 A system's *attack surface* is its number of reachable and exploitable vulnerabilities [75]. An *IP-based attack surface* is the number of network-accessible IP addresses reachable through the firewall for exploitation by an attacker. Configuring the firewall to only permit access to the necessary/authorized IPs will reduce the attack surface. The firewall policy example illustrated in Table 2.6 implements the high-level network security policy goal nsp-1 from Table 2.2, whereby the rule at Index 1 allows inbound ICMP Echo Request (ping) packets from the administrator IP range to the code revision control server, while the rule at Index 2 enables outbound responses from the server.

Index	Dir	Iface	Src IP	Dst IP	Protocol	Type	Code	Action
1	ingress	eth0	adminRange	gitIP	ICMP	8	0	*allow*
2	egress	*	*.*.*.*	*.*.*.*	*	*	*	*allow*

Table 2.6: A packet-filter policy specification for nsp-1

External threats to resources are often more easily perceived than the internal threats [148], and as a consequence, the policy may be overly-permissive to outbound network traffic. For example, the rule at Index 2 in Table 2.6 permits all outbound traffic. When configuring the rules that define a firewall policy, the specifier should consider the bi-directional nature of the traffic being filtered [24], and where possible, apply a default deny rule to outbound traffic [152]. Therefore, a revised specification for the packet-filter policy that implements nsp-1 is given in Table 2.7, whereby the revised rule at Index 2 allows only the correct ICMP reply to the authorized ICMP ping request enabled by the rule at Index 1.

Index	Dir	Iface	Src IP	Dst IP	Protocol	Type	Code	Action
1	ingress	eth0	adminRange	gitIP	ICMP	8	0	*allow*
2	egress	eth0	gitIP	adminRange	ICMP	0	0	*allow*

Table 2.7: Revised packet-filter policy specification for nsp-1

Enforcing a default deny rule supports the principal of least privilege [120], and when considering outbound traffic, this can help to mitigate/reduce the flow of Malware communication from infected systems. For example, a Remote Access Trojan (RAT) may be running on the code revision control server at gitIP. The default deny rule helps mitigate the RAT from indiscriminately making outbound connections to an external Command and Control (C&C). RATs can also be explicitly blocked by filtering ports known to be used for C&C communication [26]. Best practice [33, 115, 152] recommends that filtering rules be applied to ingress and egress traffic in order to mitigate the threat of IP spoofing. A packet's source IP address may be spoofed by an attacker in an attempt to get the firewall to process the packet as if it had originated from the system hosting the firewall itself, or from systems protected by the firewall [51]. Spoofing typically forms part of a Denial of Service (DoS) attack [102]. △

2.1.1.2 Stateful Firewall

A *stateful* firewall improves on packet-filter functionality, whereby the decision to allow or deny a packet is also based on previous packets filtered by the firewall [105, 123]. Stateful firewalls have also been classified as *dynamic packet filters* [28]. When a packet is filtered by the stateful firewall, if the packet matches a rule in the firewall policy, then the packet's state information is added to the firewall's *state table* [152]. A firewall's state table will typically include source and destination IP and port, network protocol, and connection state information such as TCP flags [103]. The connection state information may indicate, for example, the successful establishment of a bi-directional communication channel after the completion of the TCP three-way handshake.

Example 2 A *port-based attack surface* is the number of network ports reachable through the firewall for exploitation by an attacker. Configuring the firewall to only permit access to necessary ports will reduce the attack surface. The initiator of a TCP connection is dynamically allocated a port in the unprivileged port range ($1024 \mathrel{..} 2^{16} - 1$) that only has significance for the lifetime of the connection. A packet-filter firewall must allow traffic on all these ports in order to permit TCP traffic, thereby unnecessarily exposing ports to potential attackers. A stateful firewall dynamically manages the unprivileged ports as part of the TCP connection [136]. △

State information maintained by the firewall can be used to help mitigate replay attacks [103]. State information for the UDP protocol can be managed in terms of source and destination IPs and ports [152]. Some firewalls, for example, Linux iptables can manage ICMP state information in terms of source and destination IP addresses, where also an ICMP error message may be considered *related* to an entry in the firewall's state table. Table 2.8 illustrates a state table entry for the Linux firewall Netfilter [68].

State Table Entry
tcp 6 ESTABLISHED src=192.168.1.5 dst=172.16.1.6 sport=56332 dport=21 src=172.16.1.6 dst=192.168.1.5 sport=21 dport=56332 [ASSURED] [active since 91s]

Table 2.8: Netfilter state table entry

Firewalls also provide an effective way to mitigate against invalid TCP packets used in port scanning techniques. For example, the Nmap XMAS TCP port scan where all TCP flags are simultaneously set [91]. A stateful firewall also provides an effective way to mitigate against valid TCP packets that are forged. For example, TCP packets forged to mimic the expected return packets for outbound TCP traffic requests, such as the Nmap ACK port scan [91]. Packet-filters can also enforce rules based on TCP flags, however, a stateful firewall automatically manages TCP flags.

2.1.1.3 Application-layer Firewall

An *application-layer* firewall may incorporate stateful filtering functionality, however, the decision to allow or deny a packet is also based on the packet's payload at the OSI Application layer. This process is also referred to as Deep Packet Inspection (DPI) [123]. Filtering packet-payload information allows for decisions to be made on network packets containing certain keywords, for example, the FTP "put" command [111], or on packets containing certain regular expressions.

Example 3 Table 2.9 gives a firewall policy specification for the high-level network security policy goal nsp-2 from Table 2.2, whereby filtering by packet-payload is used to allow all inbound traffic to the production Web server using the FTP Application-layer protocol from the developer IP range.

Index	Dir	Src IP	Src Prt	Dest IP	Protocol	L7 Filter	Action
1	ingress	devRange	$\geq$ 1024	webIP	TCP	ftp	*allow*

Table 2.9: A policy specification for nsp-2 using Layer 7 filtering

The following regular expression [85]:

```
^220[\x09-\x0d -~]*ftp
```

is used by the Netfilter firewall to match packets that are part of FTP traffic. △

Application-layer firewalls can filter malicious payloads. For example, Netfilter provides OSI Layer 7 filtering support for executable file types, and some well known worms such as Nimda and Code Red [98]. Application-layer firewalls can also help mitigate the practice of *tunnelling*, that is, the encapsulation of data from one protocol inside another protocol in order to evade the firewall [27]. For example, using DNS tunnels to bypass captive portals for paid WiFi service [50].

2.1.1.4 Proxy Firewalls

A *proxy* is a an intermediary, that operates on behalf of two interacting network principals/systems [103, 105]. An *application-level gateway* is a proxy system with application-layer firewall filtering capabilities, and a *circuit-level gateway* is a proxy that filters at lower layers of the network stack [28]. The application-level proxy deals specifically with a particular application/service, while an advantage of the circuit-level proxy is that it provides a proxy system for a wider variety of services [24]. A proxy can be used to provide an additional level of security to an organisation's internal systems, whereby external clients may only communicate through the proxy to access an organisation's internal IP addresses not publicly accessible from the Internet [103, 152]. Proxies can also make port scan reconnaissance more difficult for an attacker, as the attacker will not receive packets created directly by their target systems [103].

2.2 Access Control Policy Conflict Resolution

An access control policy specifies the authorizations to be enforced regarding system or network resources. In general, the authorizations are positive or negative, and traditionally, an access control policy consisted of either positive or negative authorizations under a Closed or Open policy model [121]:

- **Closed Policy:** *specified authorizations are positive and signify access be granted, all other accesses are denied.*
- **Open Policy:** *specified authorizations are negative and signify access be denied, all other accesses are granted.*

The specification of an access control policy that supports both positive and negative authorisations may result in conflicts in policy decisions. That is, for example, when there is both a positive and negative authorisation specified for a given access in a policy. Whether the specified access should be granted or denied must be decidable in order for an access control system to be useful. There are various approaches to policy conflict resolution, however, there is not a one-size-fits-all solution [78, 90]. Depending on the type of access restrictions to be enforced, then there are different conflict resolution policies/methodologies that can be applied. For example, the *most-specific rule* approach [90] means that the authorization that is more-specific takes precedence, allowing for an *exception* to be made of the less-specific authorization [113]. The *denials-take-precedence* approach [90] supports the principle of least privilege, and means that negative authorizations take precedence over positive authorizations in policy decisions. The *positional* approach [131] is where authorizations are specified in an order-dependant list, and the first occurrence of an authorization in a policy is the authorization that gets enforced.

Policy models can incorporate a combination of conflict resolution policies in order to deal with conflicting authorizations in a policy, for example, by firstly applying the *most-specific rule* approach, followed by *denials-take-precedence* [121]. The rules in a firewall policy are order-dependant, and therefore when making a policy decision, we evaluate the rules in order and the first matching rule provides the decision. Thus, the sequencing of firewall rules provides a semantics for dealing with conflicts. However, the presence of conflicting rules in a policy results in policy anomalies, whereby a policy is anomalous when a network packet matches with two or more policy rules [5, 35]. Even though the sequencing of rules gives a semantics for handling policy conflicts, we argue that a firewall policy should be anomaly-free by construction, and therefore conflict-free. In Section 2.3.1 we explore classifications of firewall policy anomalies and several of the anomaly management approaches available.

2.3 Firewall Policy Management

Firewall policy management is complex and error-prone [25, 156, 157]. In specifying a policy or distributed policy configuration, a high-level security policy is translated into low-level rule syntax via a command line interface (CLI) or by using a Graphical User Interface (GUI) management console. A policy may consist of thousands of firewall rules, and as the number of rules increase, so does difficulty of modifying/updating the firewall policy. Typical errors range from invalid syntax and incorrect rule ordering, to a failure to uphold a security policy due to errors resulting from the poor comprehension of a firewall configuration [94, 152]. The GUI is the most common technique used to configure a firewall [41, 83], however, GUIs often lack configuration granularity and only provide a limited number of filter condition fields, thereby restricting an administrator from, for example, specifying firewall rules that filter TCP Flags or ICMP Codes/Types. An effective firewall policy may be further hampered by the poor understanding and/or management of the overall high-level security requirements. In this section, we consider firewall policy management practice.

2.3.1 Anomaly Management

Recall, a policy is anomalous when a network packet matches with two or more policy rules. Managing anomalies involves determining the relationships between rules in a policy, or between rules across a distributed policy configuration. This structural analysis of policies is used to detect/resolve firewall policy anomalies.

Hari et al. [73] report some of the earliest research on conflict detection and resolution in policies for packet-filters. A rule is referred to as a filter, and a *filter conflict* occurs when *“two or more filters overlap, creating an ambiguity in packet classification”*, whereby the target actions of the two filters are different. The rule relationships are modelled in a directed graph, and the policy is conflict-free if the graph is acyclic. They show that re-ordering conflicting rules does not guarantee anomaly resolution, and a scheme is proposed that results in the addition of resolve rules to the policy. However, the approach requires the manual intervention of an administrator to decide if conflicting rules should be re-ordered, or if a resolve rule is required and the position it should be inserted at in the policy.

Al-Shaer et al. [4–6, 69] provide definitions for firewall policy anomalies. The classifications are *redundancy, shadowing, generalization, correlation, irrelevance* and *spuriousness.* In later chapters of this dissertation, we use a subset of these classifications when reasoning about firewall policy anomalies. Al-Shaer et al. use a form of Binary Decision Diagram (BDD) to represent a firewall policy, and define relationships between pairwise rules. The *Firewall Policy Advisor* [5] tool implements algorithms used to identify firewall rule anomalies using set theory. Anomalies that occur in a single firewall policy are called *intra-anomalies*, and anomalies that occur in a distributed policy configuration are called *inter-anomalies.* Two additional anomaly classifications are considered in [69]. These are the *blocking existing service anomaly* and the *allowing traffic to a non-existing service anomaly.* However, these additional anomalies are detected by applying data mining techniques to firewall logs, as opposed to analysing the rules in the policy configuration.

Intra-Redundancy Anomaly. A policy has intra-redundancy if there are two policy rules with the same target actions filtering some of the same network packets, such that the removal of the redundant rule does not change the filtering semantics of the firewall policy. A redundant rule may be equivalent to, or subsumed by a rule at an earlier index in the policy. For example, in Table 2.10 the rule at Index 2 is redundant to the rule at Index 1.

Index	Src IP	Src Prt	Dst IP	Dst Prt	Protocol	Action	Anomaly
1	192.168.1.5	*	172.16.1.*	*	TCP	*allow*	
2	192.168.1.5	$\geq$ 1024	172.16.1.6	21	TCP	*allow*	Intra-redundant to 1

Table 2.10: An intra-redundancy policy anomaly example

A rule may also be redundant to a rule occurring at a later index in the policy, whereby the preceding rule is subsumed by the later rule, such that there is no intermediary rule with a different target action filtering network packets also filtered by the ‘redundant’ rule. For example, in Table 2.11 the rule at Index 1 is redundant to the ‘superset’ rule at Index 2.

Index	Src IP	Src Prt	Dst IP	Dst Prt	Protocol	Action	Anomaly
1	192.168.1.7	*	*.*.*.*	*	UDP	*deny*	Intra-redundant to 2
2	*.*.*.*	*	*.*.*.*	*	UDP	*deny*	

Table 2.11: An intra-redundancy policy anomaly example

Redundant rules may decrease the performance of the firewall due to unnecessary rule lookup overhead.

Intra-Shadowing Anomaly. A policy has intra-shadowing if there are two policy rules with different target actions filtering the same network packets, such that the shadowed rule is never matched, whereby the target action of the rule at the earlier index in the policy is applied to the network packets. A shadowed rule may be equivalent to, or subsumed by a previous rule. For example, in Table 2.12 the rule at Index 2 is a 'subset' of the rule at Index 1, and is therefore shadowed as the rules have contradictory target actions.

Index	Src IP	Src Prt	Dst IP	Dst Prt	Protocol	Action	Anomaly
1	*.*.*.*	*	172.16.1.7	80	TCP	*deny*	
2	192.168.1.5	$\geq$ 1024	172.16.1.7	80	TCP	*allow*	Intra-shadowed by 1

Table 2.12: An intra-shadowing policy anomaly example

In general, an administrator might re-order the rules or remove the shadowed rule to resolve the anomaly.

Intra-Generalization Anomaly. A policy has intra-generalization if there are two policy rules with different target actions filtering the same network packets, such that the rule at the earlier index in the policy is subsumed by a more general rule at a later index. Al-Shaer et al. consider generalization as an anomaly warning only, as an administrator may specify a rule that makes an exception of a more general rule. For example, in Table 2.13 the rule at Index 1 makes an exception of the rule at Index 2.

Index	Src IP	Src Prt	Dst IP	Dst Prt	Protocol	Action	Anomaly
1	192.168.1.5	$\geq$ 1024	172.16.1.7	80	TCP	*allow*	
2	*.*.*.*	*	172.16.1.7	80	TCP	*deny*	Intra-generalised by 1

Table 2.13: An intra-generalization policy anomaly example

We observe that the intra-generalization defines a *partial shadowing* of the general rule by the exception rule. Note, if an administrator re-ordered the rules in Table 2.12 as an approach to resolving the intra-shadowing anomaly, then an intra-generalization anomaly is the result, as illustrated in Table 2.13.

Intra-Correlation Anomaly. A policy has intra-correlation if there are two policy rules with different target actions, such that both rules can filter some network packets filtered by the other. More formally, the rules are correlated, if some of the filter fields in one of the rules are equivalent to, or subsumed by the corresponding fields in the other rule, and the remaining fields in the former rule are supersets of the corresponding fields in the later rule. For example, in Table 2.14 the rules at Index 1 and Index 2 are correlated.

Index	**Src IP**	**Src Prt**	**Dst IP**	**Dst Prt**	**Protocol**	**Action**	**Anomaly**
1	192.168.1.6	≥ 1024	172.16.*.*	22	TCP	*deny*	Intra-correlated with 2
2	192.168.1.*	≥ 1024	172.16.1.7	22	TCP	*allow*	Intra-correlated with 1

Table 2.14: An intra-correlation policy anomaly example

The intra-correlation anomaly is analogous to the packet filter conflict defined by Hari et al. [73], and in [73] it is considered a serious policy error. In contrast, Al-Shaer et al. consider intra-correlation to be an anomaly warning only. Note, Al-Shaer et al. do not consider intra-correlation for firewall rules with the same target action, we observe that this type of anomaly defines a type of redundant filtering in a policy.

Intra-Irrelevance Anomaly. A policy has intra-irrelevance if there is a policy rule that cannot be matched based on the source and destination IP address of the domains accessible through the firewall. For example, in Table 2.15 the rule at Index 1 is irrelevant in a firewall policy where only the IP range 192.168.2.* is reachable through the firewall.

Index	**Src IP**	**Src Prt**	**Dst IP**	**Dst Prt**	**Protocol**	**Action**	**Anomaly**
1	192.168.1.3	≥ 1024	172.16.1.10	80	TCP	*deny*	Intra-irrelevant

Table 2.15: An intra-irrelevance anomaly example

Intra-irrelevance is viewed as an administrator warning only, as it adds unnecessary overhead to rule filtering by the firewall but does not effect the semantics of the firewall policy.

Inter-Redundancy Anomaly. An inter-redundancy anomaly exists between an *upstream* and a *downstream* firewall policy if the downstream policy denies packets that are also denied by the upstream policy. A redundant rule in the downstream firewall may be equivalent to or subsumed by a rule in the upstream firewall. For example, in Table 2.16b the rule at Index 1 denies Skype traffic on port 33033 from IPs 192.168.1.*, and is inter-redundant to the rule at Index 1 in Table 2.16a. Al-Shaer et al. argue that the redundant rule in the downstream firewall is unnecessary as all packets matching this rule are already denied in the upstream firewall, and propose that the redundant rule can be removed.

Inter-Shadowing Anomaly. An inter-shadowing anomaly exists between an upstream firewall policy and a downstream firewall policy if the upstream policy denies packets that are allowed by the downstream policy. A shadowed rule in the downstream firewall may be equivalent to, subsumed by or a superset of a rule in the upstream firewall. For example, the rule at Index 2 in Table 2.16a inter-shadows the rule at Index 2 in Table 2.16b, whereby intended permissible SSH access from the 192.168.1.* IP addresses is not allowed to the server at 172.16.1.10 by the upstream firewall.

Inter-Spuriousness Anomaly. An inter-spuriousness anomaly exists between an upstream firewall policy and a downstream firewall policy if the upstream policy allows packets that are denied by the downstream policy. A spurious rule in the upstream firewall may be equivalent to, subsumed by or a superset of a rule in the downstream firewall policy. For example, the rule at Index 3 in Table 2.16a of the upstream firewall is allowing FTP traffic to the IP address 172.16.1.13, while the rule at Index 3 in Table 2.16b of the downstream firewall is denying the same network packets.

Inter-Correlation Anomaly. An inter-correlation anomaly exists between an upstream and downstream firewall policy if there exists a rule from each policy, such that both rules can filter some network packets filtered by the other, irrespective of the rule target actions. Inter-correlation between upstream and downstream firewalls can also create inter-shadowing and inter-spuriousness anomalies. For example, the rule at Index 4 in Table 2.16a permits Telnet traffic from 192.168.1.3 to 172.16.1.*, however the same traffic is denied by the rule at Index 4 in Table 2.16b. The rules are inter-correlated and this also results in an inter-spuriousness anomaly.

Index	Src IP	Src Prt	Dst IP	Dst Prt	Protocol	Action	Anomaly
1	192.168.*.*	$\geq$ 1024	*.*.*.*	33033	TCP	*deny*	
2	*.*.*.*	*	172.16.*.*	22	TCP	*deny*	
3	192.168.1.*	$\geq$ 1024	172.16.1.13	21	TCP	*allow*	Inter-spurious to Downstream 3
4	192.168.1.3	$\geq$ 1024	172.16.1.*	23	TCP	*allow*	Inter-correlated with Downstream 4

(a) Upstream

Index	Src IP	Src Prt	Dst IP	Dst Prt	Protocol	Action	Anomaly
1	192.168.1.*	*	*.*.*.*	33033	TCP	*deny*	Inter-redundant to Upstream 1
2	192.168.1.*	$\geq$ 1024	172.16.1.10	22	TCP	*allow*	Inter-shadowed by Upstream 2
3	192.168.1.*	$\geq$ 1024	172.16.1.13	21	TCP	*deny*	
4	192.168.1.*	$\geq$ 1024	172.16.*.*	23	TCP	*deny*	Inter-correlated with Upstream 4

(b) Downstream

Table 2.16: Inter-policy anomaly examples

Cuppens et al. [35–37] and García-Alfaro et al. [67], present MIRAGE (MIsconfiguRAtion manaGEr), and provide alternative definitions for intra- and inter-anomalies. For example, a policy has (intra-)shadowing [35] if there is a policy rule that cannot be matched due to a previous rule or a combination of previous rules filtering the same network packets, regardless of rule target actions. For example, in contrast to the view in [5, 6], the rule at Index 2 in Table 2.10 is not intra-redundant to the rule at Index 1, rather it is (intra-)shadowed [35]. A policy has (intra-)redundancy [35] if there is a policy rule that is not shadowed, and removal of this rule does not change the filtering semantics of the firewall policy. In contrast to [5, 6], rules with overlapping filter-fields are considered, and given a firewall configuration, MIRAGE will automatically detect and remove intra-redundant and intra-shadowed rules, and generate a semantically-equivalent order-independent set of disjoint rules that are anomaly-free. In contrast to [73], the approach incorporates the automatic re-writing of anomalous rules. The inter-anomalies discovered by MIRAGE in a distributed policy configuration are presented to the security administrator to determine a suitable resolution.

Yuan et al. [159] present the FIREMAN (FIREwall Modelling ANalysis) tool. Policy configurations are analysed for inconsistencies that consider intra-shadowing, intra-generalisation and inter-shadowing anomalies [5, 6]. The definition given for intra-correlation allows for considering overlapping rules. Firewall inefficiency in packet classification and memory consumption is also considered as a result of intra-redundant rules, and *'verbosities'*, whereby a set of rules may be summarized into a smaller number of rules without changing the filtering semantics of the policy. The authors observe that in a distributed firewall configuration, redundant rules with a target action of *allow* are required for a packet to be permitted traversal of all firewalls on its path from source to destination. In contrast to the opinion in [6] that inter-redundant rules with a target action of *deny* can be removed, the view in [159] is that such rules are considered good practice, as they enhance security by providing a defence in depth approach to a distributed policy configuration. Yuan et al. also consider *end-to-end security behaviour* and Cross-Path inconsistency [159] in a distributed firewall configuration, where there may exist multiple data-paths from one host/network to another host/network, and packets denied on one path may be accepted on another. The authors observe that data-paths are determined by the underlying routing protocol and a given path may not always be available, and as such, the firewall configuration should consider all possible paths.

Chomsiri and Pornavalai [30] propose a method of firewall policy analysis using relational algebra. The definitions provided for intra-redundant and intra-shadowed rules are analogous to [35, 36], and upon detection, such rules are removed in order to reduce the size of the policy. Similar to the notion of verbosities in [159], Chomsiri and Pornavalai propose combining rules that may be 'summarized' without changing the filtering semantics of the policy. The definition given for intra-generalization is analogous to [5, 6], however in contrast to [5, 6], rules with overlapping filter-fields are considered through the definition provided for intra-correlation. Intra-correlated and intra-generalized rules are reported to the administrator as a result of policy analysis.

Abedin et al. [1] propose an algorithmic approach to detect and resolve anomalies between pairwise rules in a firewall policy. The definitions provided for intra-redundancy and intra-shadowing are analogous to [5, 6], and intra-generalization is considered as an anomaly warning only. However, similar to [30, 35, 36, 159], Abedin et al. consider rules with overlapping filter fields, and a definition of intra-correlation is provided, whereby two policy rules are correlated if they are *"not disjoint, but neither is the subset of the other"* [1].

Fitzgerald et al. [53] propose an ontology engineering approach to model and reason about firewall policy configurations. An iptables policy model ontology is constructed, and policies are reasoned over for intra-redundancy, intra-shadowing, intra-generalisation and intra-correlation anomalies [5, 6].

Regardless of the divergence in anomaly classification, the authors of [30, 35, 53, 159] all observe shortcomings in the work of Al-Shaer et al. [5, 6]. For example, in [5, 6] rule anomaly analysis is performed only on a pairwise basis, and therefore anomalies involving more than two rules are not considered. For example, the rules at Index 1 and Index 2 of Table 2.17 deny SSH traffic from the IPs 192.168.1.* to the servers at 172.16.1.7 and 172.16.1.8, respectively, however, the rule at Index 3 is intended to permit SSH access to both servers from 192.168.1.5. Performing pairwise analysis on the rules in Table 2.17 will not detect an anomaly, however, considering the union of the rules at Index 1 and Index 2; we have that all of the packets that match the rule at Index 3 have already been matched [35, 36].

Index	Src IP	Src Prt	Dst IP	Dst Prt	Protocol	Action	Anomaly
1	192.168.1.*	$\geq$ 1024	172.16.1.7	22	TCP	*deny*	
2	192.168.1.*	$\geq$ 1024	172.16.1.8	22	TCP	*deny*	
3	192.168.1.5	$\geq$ 1024	172.16.1.7 172.16.1.8	22	TCP	*allow*	Undetected Intra-shadowing by $(1 \cup 2)$

Table 2.17: A shortcoming of pairwise rule analysis

Buttyán et al. [23] propose a tool based on FIREMAN [159] for managing anomalies in stateful firewall policies. The authors argue that verifying a stateful firewall for inconsistencies can be reduced to the problem of verifying a stateless firewall for inconsistencies. Policies are analysed for intra-redundancy, intra-shadowing and intra-generalization anomalies that are analogous to the definitions given in [5, 6], and intra-correlation anomalies where rule overlap is considered. A limitation of the approach is that their model does not distinguish rules with different state information, that is, for example, there is no differentiation between the establishment and termination phase of a given stateful protocol, and as a consequence, they do not consider more complex anomalies that may occur specifically in the stateful case.

Cuppens et al. [38] and García-Alfaro et al. [66] propose an algorithmic approach to detect and resolve anomalies in a stateful firewall policy. A connection-oriented protocol is modelled using general automata, whereby the permitted protocol states and transitions are encoded. Intra-redundant and intra-shadowed rules [35, 36] are considered for the stateful firewall policy. Further definitions are proposed, whereby an *intra-state* anomaly occurs in a stateful firewall policy if there are policy rules that partially match (complete) the paths of the protocol automata. For example, if the connection establishment and/or connection termination phase is permitted, and the remaining rules necessary to complete the paths of the protocol automata are either missing or conflict with the earlier rules. The proposed algorithms detect conflicting rules, and a modification is applied *"so that the resulting set gets consistent with the action with higher priority (e.g., accepting the termination phase if the establishment was accepted as well)"* [66]. In the case of missing rules, then covering-rules are suggested to the administrator as a means of completing the path of the protocol automata. Their work also considers invalid protocol states, and *inter-state* anomalies that may occur in a firewall policy that filters packets against both stateful and stateless rules. The work in [38, 66] also extends the MIRAGE [67] tool.

The work of Al-Shaer et al. [5, 6] and Cuppens et al. [35, 38] is focused on firewall rules with a target action of either *allow* or *deny*. However, to permit the logging of relevant packets for auditing purposes [152], then a firewall policy should also allow for specifying rules with a target action of *log*. In terms of the structural analysis of a policy, a rule with a target action of *log* may be shadowed by a rule with a target action of *allow* or *deny*, however, a rule with a target action of *allow* or *deny* cannot be shadowed by a rule with a target action of *log*. Following the approaches taken by both [5, 6] and [35, 38], then the rule at Index 2 in Table 2.18 will be falsely detected as intra-shadowed. Fitzgerald et al. [53] consider rules with a target action of *log* when analysing a policy for anomalies.

Index	**Src IP**	**Src Prt**	**Dst IP**	**Dst Prt**	**Protocol**	**Action**	**Anomaly** [5, 35]
1	192.168.1.*	$\geq$ 1024	*.*.*.*	6667	TCP	*log*	
2	192.168.1.*	$\geq$ 1024	*.*.*.*	6667	TCP	*deny*	Detected as Intra-shadowed by 1

Table 2.18: A false-positive anomaly detection

In this dissertation, an algebra is proposed for firewall policy configuration management. While policies in the proposed model are anomaly-free by construction, we can however define anomalies using the algebra; by considering how a policy changes when composed with other policies. We use a number of anomaly definitions from [5, 6], considered in this section, when reasoning about firewall policy anomalies in later chapters.

2.3.2 Query Systems

Firewall query analysis allows an administrator to pose questions of a policy configuration, such as, for example, *"does the policy permit SSH traffic from system x to system y?"* or *"what network services are available on system x?"*.

Mayer et al. [95], present Fang (Firewall ANalysis enGine). Based on graph algorithms and a rule-base simulator, Fang parses vendor-specific firewall policy and configuration files, and constructs a model of the network topology and a global firewall policy for the network. Fang interacts with the administrator through a query-and-answer session, and queries are constructed as triples, consisting of source IPs, destination IPs and endpoint services/ports. However, the administrator is restricted to querying only the kinds of packets that are permitted. The Lumeta Firewall Analyzer [96, 155], built as an improvement on Fang, automates the process of specifying the topology configuration file by taking as input a formatted routing table. The process of constructing the firewall queries is also automated in [96, 155].

Eronen and Zitting [47] propose an expert system for query analysis, implemented in constraint programming logic using ECLiPSe [153]. A query is constructed as a six-tuple, consisting of source and destination IP and port, network protocol, and flags. The flags attribute is for TCP connections, however, only the SYN and ACK flags are considered. Their model allows for querying both *allow* and *deny* rules. Eronen and Zitting argue that in comparison to Fang [95], the expert system is a more natural solution for query analysis.

Liu et al. [88] present the Structured Firewall Query Language (SFQL), an SQL-like query language. Liu et al. state that constructing an expert system such as [47] *"just for analysing a firewall is overwrought and impractical"* [88], however, they do not give their reasoning for this assertion. SFQL queries can be constructed over *allow* and *deny* rules for an arbitrary number of filter fields. The authors show how SFQL can be mapped to natural language, however, the example queries in [88] only consider network interface, source and destination IP address, destination port and protocol. A study of Fang beta-testers in [96] showed that a significant challenge to effective query analysis is that users often do not know what to query of a policy.

Marmorstein and Kearns [93] present ITVal, a tool that enables query analysis for iptables firewall policies. Firewall queries are constructed in terms of source and destination IP and port, network protocol and connection state. Queries may also include the SYN, ACK, FIN, PSH, RST and URG TCP flags, or ICMP packet type, depending on protocol. Querying firewall rules with a target action of *log* is also considered. The authors note that a limitation of their approach is that some queries may generate a large volume of results, and how results are presented may be difficult for an administrator to reason over. In comparison to [88], the query language proposed in [93] is closer to natural language.

While query analysis provides an administrator with a means of asking questions of a firewall policy, what can actually be queried is restricted by the collection of filter condition attributes and target actions expressible in the query language. Effective query analysis may be further hampered by the complexity of the query language, or through an administrators inability to construct useful queries [96]. The FIREMAN [159] toolkit, discussed in Section 2.3.1, can also be used for query analysis, and provides an administrator with a predefined collection of queries. A consequence of the algebra proposed in this dissertation is that it enables an administrator to perform effective query analysis of a firewall policy configuration. While we do not construct individual high-level queries, we do however demonstrate in later chapters how policies in the algebra may be tested/queried for compliance with best practice standards and recommendations.

2.3.3 High-level Specification Languages

High-level specification languages provides an administrator with the means to reduce the complexity of constructing a firewall policy configuration.

Guttman [72] reported some of the earliest research in this area. The proposed specification language has a Lisp-like syntax, and allows an administrator to specify a global access control policy for the routers of a network. Knowledge about the network topology is incorporated in the approach, and a policy configuration for individual routers is synthesised. Policy rules are conflict-free, however, the proposed approach only considers rules with a target action of *allow*, and all other traffic is denied by default. As a consequence, this restricts an administrator from explicitly denying network traffic that may, for example, be spoofed, or consist of packets with invalid combinations of TCP flags. Guttman notes a limitation of the approach is that the generated policies only administer network access control for the systems beyond the routers themselves, and as such, policies do not consider access control for services hosted locally by the router.

Bartal et al. [13] and Mayer et al. [96] present the Firmato toolkit. The proposed specification language allows an administrator to specify the network security policy and the topology for the network in terms of an entity-relationship

(ER) model. Subsequently, a policy configuration for the Lucent Managed Firewall is synthesised from the ER model. In contrast to [72], the modular separation of the network security policy from the network topology provides an administrator with greater flexibility when managing the network configuration. For example, the same high-level network security policy may reused with different network topologies. A limitation of the work is that it only applies to packet-filter policy configurations. Note, the Fang [95] query analysis tool, discussed in Section 2.3.2 is a module of Firmato.

The High Level Firewall Language (HLFL) [84] translates high-level firewall rules into usable rulesets for iptables, Cisco ACLs [31], IPFW [11] and others. However, the generated rulesets are order-dependant and may contain anomalies, and the approach does not provide support for incorporating knowledge about a network topology when specifying the high-level rules.

Cuppens et al. [39] present a specification language based on XML syntax. The language is supported by the Organization-Based Access Control (Or-BAC) [2] model, and as such, policies are specified at a high level of abstraction. Similar to [96], the network security policy is decoupled from the network topology. An XSLT translation process is applied to the high-level Or-BAC rules to synthesize a collection of firewall rules for iptables. The MIRAGE [67] tool, discussed in Section 2.3.1, incorporates the OrBAC model as a means of specifying and deploying anomaly-free network security policies.

In a different vein, Liu and Gouda [87] propose a method of *diverse firewall design*. The approach is inspired by N-version programming [8], whereby the same network security policy is given to multiple teams who independently specify different versions of the firewall policy. The firewall policies are analysed algorithmically for conflicts, and the different teams compare the results and discuss any conflicts across the specifications. Subsequently, all teams agree on the policy that is to be synthesized. A firewall policy is modelled as a *firewall decision diagram* (FDD) [71], similar to a BDD. In [71], Gouda and Liu consider the issues of consistency (correct rule ordering), completeness (matching all network traffic), and compactness (no redundant rules) when specifying a firewall policy as an FDD. A disadvantage of the approach is that specifying FDDs is complex.

Fitzgerald and Foley [52] propose using ontologies to represent knowledge about firewall policy configurations. Policies are specified using Description Logic and SWRL. Semantic Threat Graphs [60] are used to encode catalogues of best practice firewall rules, and an automated synthesis of standards-compliant rules for a policy configuration is considered. However, the administrator must manually construct the rulesets for the catalogues then populate the Semantic Threat Graphs, and this process is error-prone. The proposed model can also be used for firewall policy query analysis.

Adão et al. [3] propose a declarative policy specification language, and present Mignis (“murus ignis” (“a wall of fire” in Latin)), a tool that translates high-level access control specifications into low-level policy configurations for Netfilter. An abstract model of the Netfilter firewall is proposed, and definitions for Network Address Translation and stateful filtering are encoded. The synthesised policies consist of order-independent iptables firewall rules. However, the proposed approach is tightly coupled with Netfilter.

Similar to a shortcoming of query analysis, the policy that can be synthesized using a high-level specification language is limited by the collection of filter condition attributes and rule target actions expressible in the language. Additionally, work in this area in general has been focused on packet-filter firewalls.

2.4 Challenges of Policy Composition

An administrator may develop a firewall policy by specifying independent or related requirements, that need to be replaced by a policy that adequately captures the requirements of the individual specifications. However, while the individual specifications may themselves be consistent with the network security policy, their composition may result in a policy that enables unauthorized traversal of the firewall. Mismanagement of composition in a distributed policy architecture may allow for an attacker to traverse the network configuration in order to reach their intended target by following possibly direct or indirect paths that occur as a result of composition.

Gong and Qian [70] considered the problem of secure interoperation in networks of heterogeneous access control systems. A graph-based model is used to represent a secure access control system, whereby nodes are system entities and arcs specify the direction of positive/negative access. System interoperation is defined by composing graphs, and it is shown that the composition of individually secure systems does not necessarily result in a secure system. That is, an unauthorized user may potentially gain access to a resource by following an indirect path across the individually secure but now interoperating access control systems. Gong and Qian show that in the graph-based approach, the optimal elimination of interoperation vulnerabilities occurring as a result of composition/interoperation is NP-complete. Bistarelli et at. [19] consider the problem in [70], and propose a constraint-based model to represent a secure access control system. System reconfiguration for secure interoperation is expressed as a Constraint Satisfaction Problem [97]. The advantage of the constraint-based approach, is that trade-offs may be made over the set of all interoperation vulnerabilities that occur as a result of composing the individually secure access control systems, in reasonable time [19].

The *cascade vulnerability problem* [99] is concerned with secure interoperation in networks of multilevel secure systems. The interconnection may be between systems accredited with different levels of risk, and a cascade vulnerability occurs when confidentiality requirements are threatened, whereby an attacker *"can take advantage of network connections to compromise information across a range of security levels that is greater than the accreditation range of any of the component systems he must defeat to do so"* [99]. This is similar to the problem of security violation due to indirect paths considered in [70]. Bistarelli et at. [20] propose a constraint-based approach to model, detect and eliminate the cascade vulnerability problem in an arbitrary multilevel secure network. Cascading network paths are detected and removed by breaking a minimum number of system links in polynomial time.

Denning [42] reported some of the earliest work on lattice-based models for secure information flow in systems. An information flow policy is concerned with the flow of information between the different security classes in a system, and the information flow policy in [42] consists of the finite set of security classes, a binary ordering between security classes in terms of a "can flow" relation, and a binary class combining (join) operator. Denning demonstrated that under certain axioms, an information flow policy forms a finite lattice structure.

Jacob [77] considered the *refinement* of systems. It is shown how different refinement relations can be used to capture different kinds of system properties. When one system refines another, it is said to be *"no worse with respect to some property of interest"* [77], corresponding to one definition of refinement from [7] as *"an added development or improvement"*. However in [77], the notion of refinement does not imply one system is better than another, only no worse.

Foley [56] reinterprets the notion of refinement [77] for refinement of policy. In contrast to [42], an information flow policy is defined as a reflexive relation, whereby details about system entities are encoded along with their security classes. The policy ordering relation captures the property of restrictiveness, and when one policy refines another, the former is said to be *"no less restrictive"* [56] than the latter. Foley demonstrated that the set of all information flow policies [56] form a lattice under the refinement ordering, with a lowest upper bound operator. In [57], Foley extends the model of policy refinement [56] to allow for specifying and reasoning over integrity policies. The proposed approach extends the Clark-Wilson (CW) integrity model [32] to allow for dynamic separation of duty. The policy model forms a lattice under refinement with a lowest upper bound operator for policies. The theory of reflexive relations is extended to include ratings in [58], such that there is a degree of confidence that a given policy is being upheld. The model is used to reason about a problem related to synchronizing devices that is similar to the cascade vulnerability problem. However, Foley envisions that optimally eliminating the cascading paths using the proposed approach is NP-complete, similar to the graph-based approach taken in [70] for the optimal elimination of interoperation vulnerabilities in access control systems. This is in contrast to the later approach in [20] where cascading paths can be eliminated in polynomial time.

Zhao and Bellovin [160, 161] propose a policy algebra framework for packet-filter firewalls. The framework defines operators for addition, conjunction, subtraction, and summation, over rules and policies. Cost and risk functions associated with policy enforcement are also specified. The proposed approach allows for the detection of conflicts that arise as a result of composition, however, it does not allow for preventing them. Instead, the authors propose a choice of conflict resolution policies be presented to the administrator, who then decides if an access should be granted or denied by considering options such as, for example, the *most-specific rule* approach [90] implemented as a decorrelation algorithm, or by applying the target action of the first matching rule [131].

2.5 Discussion

A firewall is an important network security mechanism, however, the effectiveness of a firewall is hampered by how well its policy is configured. There is a rich literature of work on detecting and resolving anomalies in firewall policy configurations. However, in general, research such as [1, 5, 6, 30, 35, 73, 159] is focused on the conventional five-tuple firewall rule with a binary target action of *allow* or *deny*, and with the exception of [23, 38, 53, 66], work in this area has been focused on anomalies in packet-filter policies only. The extent as to what can be synthesized from a high-level specification language suffers from similar shortcomings, and the approaches taken by [13, 39, 71, 72] are only applicable to packet-filter firewalls. Querying a policy configuration is also limited by the filter condition attributes and rule target actions that can be specified in the query language, and with the exception of [52, 93], research in this area been focused on packet-filter policies. The query-based approach is further limited by the ability of the administrator to construct useful queries.

Firewall policies may need to be reconfigured for highly dynamic environments, and ad-hoc updates are often carried out manually. Policy updates may result in a configuration that is not consistent with a network security policy; the updates may lead to a configuration that is overly-restrictive, thereby denying legitimate access, or the updated configuration may be overly-permissive, resulting in unwanted packets traversing the firewall and potentially resulting in security violations similar to those considered in Section 2.4. The focus in this dissertation is on firewall policy composition, and we argue that there is a need for a consistent means of composing firewall policies, such that the result is anomaly-free and composition upholds the security requirements of each policy involved. Lattice-based access control models are fundamental components of computer security and provide consistent composition operators for a variety of access control policy models [56, 78, 121, 122]. In this dissertation, we reinterpret the notion of refinement [56, 57, 77] and develop a lattice-based policy model for firewalls.

Part II

Theory and Foundations

Chapter 3

Attributes of a Linux-based Firewall

The objective of this chapter is to develop a formal model for various filter condition attributes of the Linux iptables firewall. The attributes will be used to construct packet-rules for the $\mathcal{FW}_0$ firewall policy algebra in Chapter 4, and extended to construct range-based rules for the $\mathcal{FW}_1$ algebra in Chapter 5. The chapter is organised as follows. Section 3.1 provides an overview of Linux Netfilter and iptables. Section 3.2 explores iptables-rule constructs and gives a formal definition for filter condition attributes for use in $\mathcal{FW}_0$. Filter condition attributes for IPs/ports and the TCP, UDP and ICMP protocols are developed. Additional filtering specifications are also defined.

3.1 Linux iptables

Netfilter [68, 138] is a framework that enables packet filtering, Network Address Translation (NAT) and packet mangling within the Linux kernel. A front-end command-line utility called *iptables* is used to construct firewall rules that instruct Netfilter how to interpret packets. As a firewall, iptables has stateless, stateful and Deep Packet Inspection (DPI) filtering capabilities.

An iptables (firewall, NAT or mangle) rule requires the specification of a *table*, a *chain*, the accompanying *filter conditions* on packet fields that must be matched and an associated target *action*. With iptables, there are four tables: `filter`, `nat`, `mangle` and `raw`. A table is a set of chains and it defines the global context for common packet handling functionality. For example, the `filter` table defines the set for firewall rules, while the `nat` table defines the set of rules concerned with NAT. A chain is a set of rules that define the local context within a table.

Rules within a chain are applied to the context defined both by the chain itself and the particular table. This dissertation focuses on the firewalling aspects of iptables, that is, the `filter` table. There are three built-in chains defined within the `filter` table that govern traffic being routed to (INPUT chain), from (OUTPUT chain) and beyond the firewall itself (FORWARD chain). Figure 3.1 illustrates the iptables packet traversal according to its associated chain.

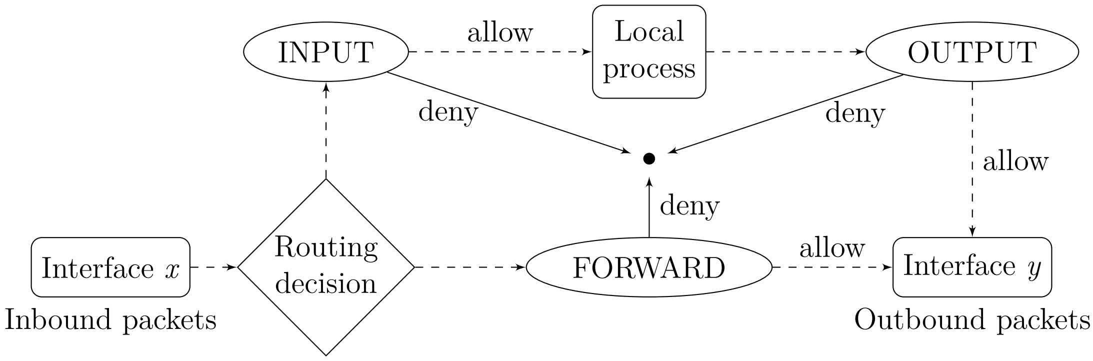


Figure 3.1: Linux iptables filter table chain packet traversal

The following example demonstrates the specification of a firewall rule using the iptables command-line syntax.

Example 1 The following iptables access-control rule specifies that inbound (`INPUT`) TCP packets (`-p tcp`) originating from the IP address 0.0.0.1 (`-s` 0.0.0.1) destined to the IP address 0.0.0.2 (`-d` 0.0.0.2) will be permitted traversal of the firewall (`-j ACCEPT`).

```
iptables -t filter -A INPUT -p tcp -s 0.0.0.1 -d 0.0.0.2 -j ACCEPT
```

Note, the filter table is the default table for iptables, therefore it is not necessary to include the (`-t filter`) option when specifying a firewall rule. △

3.2 Encoding iptables Filter Condition Attributes

In this section, the core filter condition attributes used to define firewall rules in the $\mathcal{FW}_0$ firewall algebra are formally specified. Attributes are derived from the Data Link, Network, Transport and Application Layers of the OSI model. Additional filter condition attributes are also defined. Attribute definitions are extended in Chapter 5 to specify range-based filter conditions used to define firewall rules in the $\mathcal{FW}_1$ policy algebra.

3.2.1 Data Link Layer Filtering

Packet-type. iptables allows the specification of firewall rules that filter the type of packet at the Data Link layer of the OSI model.

Example 2 The following iptables rule specifies that inbound (`INPUT`) TCP broadcast packets (`-p tcp --pkt-type broadcast`) destined for the IP address 0.0.0.1 (`-d` 0.0.0.1) be denied (`-j DROP`).

```
iptables -A INPUT -p tcp --pkt-type broadcast \
    -d 0.0.0.1 -j DROP
```

△

Let *PktTpe* be the set of Data Link-layer packet types, whereby:

$PktTpe ::= \mathsf{unicast} \mid \mathsf{broadcast} \mid \mathsf{multicast}$

Media Access Control (MAC) Addresses. iptables allows for constructing firewall rules that filter the MAC address of a Ethernet device at the Data Link layer of the OSI model. Filtering is applied to source MAC addresses entering the FORWARD or INPUT chains of the filter table [146].

Example 3 The following iptables rule specifies that inbound TCP packets (`INPUT -p tcp`) destined for the IP address 0.0.0.1 (`-d` 0.0.0.1) with source MAC address 00:0F:EA:91:04:08 (`-m mac --mac-source 00:0F:EA:91:04:08`) be denied (`-j DROP`).

```
iptables -A INPUT -p tcp -d 0.0.0.1 \
    -m mac --mac-source 00:0F:EA:91:04:08 -j DROP
```

△

Let basic type *MAC* be the set of all MAC addresses. We define:

$[MAC]$

For simplicity, we do not consider how the values of *MAC* may be constructed, other than to assume that the usual human-readable notation can be used, such as 00:0F:EA:91:04:08 and 00:0F:EA:91:04:09 $\in MAC$.

3.2.2 Network Layer Filtering

IP Addresses. iptables allows the specification of firewall rules that filter the source/destination IP address of a network packet. For simplicity, we consider only the IPv4 address range, as a firewall rule with an IPv6 address filter condition attribute must be specified using *ip6tables*; the equivalent IPv6 firewall [68]. Figure 3.2[1] depicts the IPv4 protocol packet header.

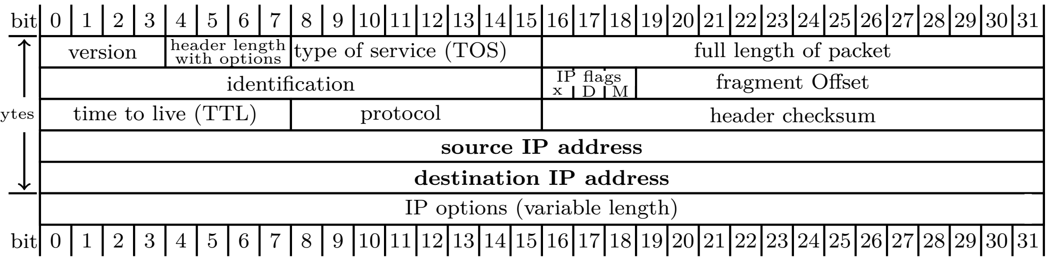


Figure 3.2: The IPv4 protocol packet header

Example 4 The following iptables rule specifies that inbound (`INPUT`) TCP packets (`-p tcp`) originating from the IP address 0.0.0.1 (`-s` 0.0.0.1) destined to the IP address 0.0.0.2 (`-d` 0.0.0.2) will be permitted traversal of the firewall (`-j ACCEPT`).

```
iptables -A INPUT -p tcp -s 0.0.0.1 -d 0.0.0.2 -j ACCEPT
```

It is also possible to specify firewall rules that filter by source/destination IP range. For example, the following iptables rule specifies that inbound (`INPUT`) TCP packets (`-p tcp`) originating from the IP range 0.0.0.1-0.0.0.3 (`-m iprange --src-range` 0.0.0.1-0.0.0.3) destined to the IP range 0.0.0.2-0.0.0.4 (`--dst-range` 0.0.0.2-0.0.0.4) will be allowed (`-j ACCEPT`).

```
iptables -A INPUT -p tcp -m iprange --src-range 0.0.0.1-0.0.0.3 \
    --dst-range 0.0.0.2-0.0.0.4 -j ACCEPT
```

△

In this dissertation, IP addresses as encoded using natural numbers, as there is a logical mapping from IPs to natural numbers, and a logical mapping from IP ranges and CIDR blocks to intervals of $\mathbb{N}$. This is done for ease of exposition, and to exploit the natural ordering of $\leq$ over $\mathbb{N}$. Let *IP* be the set of all IPv4 addresses, given as the set of all natural numbers from $0 \mathinner{.\,.} \mathsf{maxIP}$, whereby:

[1] Packet header illustrations in this dissertation are adaptations of the TikZ source at [140].

$$\mathsf{maxIP} == 2^{32} - 1$$
$$IP == \{i : \mathbb{N} \mid 0 \leq i \leq \mathsf{maxIP}\}$$

ICMP. The Internet Control Message Protocol, ICMP, is an OSI Network layer protocol. It is used by networked devices to relay error/query messages [108]. iptables allows for filtering by ICMP Type and Code.

Example 5 The following iptables rule specifies inbound (`INPUT`) ICMP Echo Request (ping) packets (`-p icmp --icmp-type 8/0`), originating from the IP address 0.0.0.1 (`-s` 0.0.0.1), destined for the IP address 0.0.0.2 (`-d` 0.0.0.2), be allowed (`-j ACCEPT`).

```
iptables -A INPUT -p icmp --icmp-type 8/0 -s 0.0.0.1 \
    -d 0.0.0.2 -j ACCEPT
```

A second rule, necessary for the bi-directional network communication channel specifies, that outbound (`OUTPUT`) ICMP Echo Response packets (`-p icmp --icmp-type 0/0`), from the IP address 0.0.0.2 (`-s` 0.0.0.2), destined for the IP address 0.0.0.1 (`-d` 0.0.0.1), be allowed (`-j ACCEPT`).

```
iptables -A OUTPUT -p icmp --icmp-type 0/0 -s 0.0.0.2 \
    -d 0.0.0.1 -j ACCEPT
```

△

Let *TypesCodes* be the set of all valid ICMP Type/Code pairs, as detailed in [110]. Most Type/Code pairs $i, j \in \mathbb{N}$ are given as $(i, j) \in \mathit{TypesCodes}$, for example, (8,0) is an ICMP Echo Request, however, some ICMP Types $i \in \mathbb{N}$; for example, Type 19 (reserved for security), have no ICMP Code, and are given as $(i, -1) \in \mathit{TypesCodes}$. Note, for ease of exposition we use syntactic sugar in the definition of *TypesCodes* as a free-type. We define:

$$\begin{aligned}\mathit{TypesCodes} ::= {} & \mathsf{(0,0)} \mid \mathsf{(3,0)} \mid \mathsf{(3,1)} \mid \mathsf{(3,2)} \mid \mathsf{(3,3)} \mid \mathsf{(3,4)} \mid \mathsf{(3,5)} \mid \mathsf{(3,6)} \mid \mathsf{(3,7)} \mid \\ & \mathsf{(3,8)} \mid \mathsf{(3,9)} \mid \mathsf{(3,10)} \mid \mathsf{(3,11)} \mid \mathsf{(3,12)} \mid \mathsf{(3,13)} \mid \mathsf{(3,14)} \mid \mathsf{(3,15)} \mid \mathsf{(4,0)} \mid \mathsf{(5,0)} \mid \\ & \mathsf{(5,1)} \mid \mathsf{(5,2)} \mid \mathsf{(5,3)} \mid \mathsf{(6,0)} \mid \mathsf{(8,0)} \mid \mathsf{(9,0)} \mid \mathsf{(9,16)} \mid \mathsf{(10,0)} \mid \mathsf{(11,0)} \mid \mathsf{(11,1)} \mid \\ & \mathsf{(12,0)} \mid \mathsf{(12,1)} \mid \mathsf{(12,2)} \mid \mathsf{(13,0)} \mid \mathsf{(14,0)} \mid \mathsf{(15,0)} \mid \mathsf{(16,0)} \mid \mathsf{(17,0)} \mid \mathsf{(18,0)} \mid \\ & \mathsf{(19, -1)} \mid \mathsf{(20, -1)} \mid \mathsf{(21, -1)} \mid \mathsf{(22, -1)} \mid \mathsf{(23, -1)} \mid \mathsf{(24, -1)} \mid \mathsf{(25, -1)} \mid \mathsf{(26, -1)} \mid \\ & \mathsf{(27, -1)} \mid \mathsf{(28, -1)} \mid \mathsf{(29, -1)} \mid \mathsf{(30, -1)} \mid \mathsf{(31, -1)} \mid \mathsf{(32, -1)} \mid \mathsf{(33, -1)} \mid \mathsf{(34, -1)} \mid \\ & \mathsf{(35, -1)} \mid \mathsf{(36, -1)} \mid \mathsf{(37, -1)} \mid \mathsf{(38, -1)} \mid \mathsf{(39, -1)} \mid \mathsf{(40,0)} \mid \mathsf{(40,1)} \mid \mathsf{(40,2)} \mid \\ & \mathsf{(40,3)} \mid \mathsf{(40,4)} \mid \mathsf{(40,5)} \mid \mathsf{(41, -1)} \mid \mathsf{(253, -1)} \mid \mathsf{(254, -1)}\end{aligned}$$

3.2.3 Transport Layer Filtering

Network Ports. A network port is a communication end-point used by the Transport layer protocols (for example, TCP/UDP) of the OSI model. A port is always associated with an IP address. The Internet Assigned Numbers Authority (IANA), maintains the official assignments of port numbers for specific uses/services [151]. For example, 22 is used for SSH and 80 is used for HTTP.

Example 6 The following iptables rule specifies that inbound (`INPUT`) TCP packets (`-p tcp`) originating from the IP address 0.0.0.1 (`-s` 0.0.0.1) destined for the HTTP and HTTPS services (`-m multiport --dports 80,443`) at the IP address 0.0.0.2 (`-d` 0.0.0.2) be denied (`-j DROP`).

```
iptables -A INPUT -p tcp -m multiport --dports 80,443 \
    -s 0.0.0.1 -d 0.0.0.2 -j DROP
```

△

A port is a 16-bit unsigned integer. Let *Ports* be the set of all ports, given as the set of all natural numbers from $0 \mathrel{..} \mathsf{maxPrt}$. We define:

$$\mathsf{maxPrt} == 2^{16} - 1$$

$$\mathit{Ports} == \{i : \mathbb{N} \mid 0 \leq i \leq \mathsf{maxPrt}\}$$

UDP. The User Datagram Protocol, UDP, is an OSI Transport layer protocol [107]. It is a transaction-oriented protocol, with no guarantee of packet delivery or duplicate protection. Figure 3.3 depicts the UDP protocol packet header.

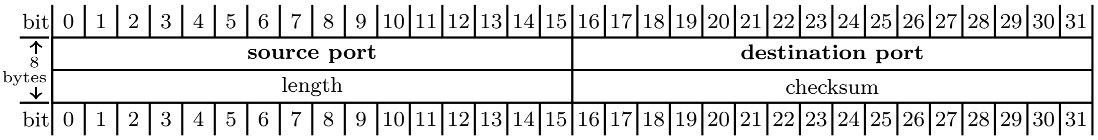


Figure 3.3: The UDP protocol packet header

iptables allows the specification of firewall rules that filter UDP traffic.

Example 7 The following iptables rule specifies that inbound (`INPUT`) UDP packets (`-p udp`) originating from the IP address 0.0.0.1 (`-s` 0.0.0.1) destined for the IP address 0.0.0.2 (`-d` 0.0.0.2) are to be denied (`-j DROP`).

```
iptables -A INPUT -p udp -s 0.0.0.1 -d 0.0.0.2 -j DROP
```

△

Let *UDP* define the set of packet values for the UDP protocol; whereby `1` signifies that a packet is using UDP or `0` signifies it is not. We define:

$$UDP ::= \texttt{1} \mid \texttt{0}$$

TCP. The Transmission Control Protocol, TCP, is a connection-oriented OSI Transport layer protocol, that provides reliable, ordered, and error-checked packets between communicating network principals [109]. Figure 3.4 depicts the TCP protocol packet header.

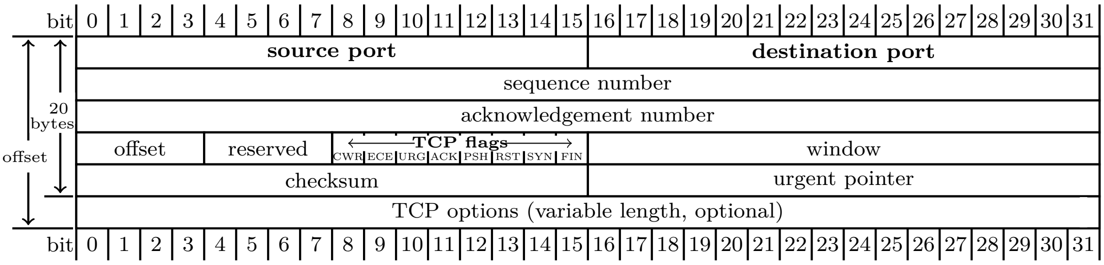

bit 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
source port
destination port
sequence number
20 bytes
acknowledgement number
offset
offset
reserved
TCP flags
CWR ECE URG ACK PSH RST SYN FIN
window
checksum
urgent pointer
TCP options (variable length, optional)
bit 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

Figure 3.4: The TCP protocol packet header

iptables allows for TCP firewall rules to be constructed using a pair of TCP flags specifications. The first, specifies the flags that are to be examined in a packet-header, and the second specifies the flags that are to be set in a packet-header, these are the *mask* and *comp* values for a TCP packet [68].

Example 8 The following iptables rule specifies that inbound (`INPUT`) TCP packets (`-p tcp`) with all flags set (a TCP XMAS port-scan pattern [91]), and all flags to be examined by the firewall (`--tcp-flags ALL, ALL`), originating from source IP address `0.0.0.1` (`-s 0.0.0.1`), destined for the IP address `0.0.0.2` (`-d 0.0.0.2`) are to be denied (`-j DROP`).

```
iptables -A INPUT -p tcp --tcp-flags ALL, ALL -s 0.0.0.1 \
    -d 0.0.0.2 -j DROP
```

The following iptables rules enable the TCP three-way handshake; by permitting HTTP traffic from IP address `0.0.0.1` to/from the IP address `0.0.0.2` while filtering network traffic based on TCP flags.

```
iptables -A INPUT -p tcp --sport 1024:65535 --dport 80 \
    --tcp-flags SYN, SYN -s 0.0.0.1 -d 0.0.0.2 -j ACCEPT

iptables -A OUTPUT -p tcp --dport 1024:65535 --sport 80 \
    --tcp-flags SYN,ACK SYN,ACK -d 0.0.0.1 -s 0.0.0.2 -j ACCEPT

iptables -A INPUT -p tcp --sport 1024:65535 --dport 80 \
    --tcp-flags ACK, ACK -s 0.0.0.1 -d 0.0.0.2 -j ACCEPT

iptables -A OUTPUT -p tcp --sport 80 --dport 1024:65535 \
    --tcp-flags ACK, ACK -d 0.0.0.1 -s 0.0.0.2 -j ACCEPT
```

$\triangle$

Let *Flags* be the set of TCP flags filterable in an iptables rule (as a mask or a comp specification), whereby:

$$Flags ::= \mathsf{syn} \mid \mathsf{ack} \mid \mathsf{fin} \mid \mathsf{psh} \mid \mathsf{rst} \mid \mathsf{urg}$$

and let $Flag_{Spec}$ be the set of all (mask, comp) pairs, we define:

$$Flag_{Spec} == \mathbb{P}\, Flags \times \mathbb{P}\, Flags$$

3.2.4 Application Layer Filtering

Layer 7 Protocol Filtering. iptables allows for constructing firewall rules that filter certain protocols at the Application Layer of the OSI model [86].

Example 9 The following iptables rule specifies that inbound (`INPUT`) SSH packets (`-m layer7 --l7proto ssh`), originating from the source IP address 0.0.0.1 (`-s` 0.0.0.1), destined for the IP address 0.0.0.2 (`-d` 0.0.0.2) be allowed (`-j ACCEPT`).

```
iptables -A INPUT -m layer7 --l7proto ssh \
    -s 0.0.0.1 -d 0.0.0.2 -j ACCEPT
```

$\triangle$

Let $Proto_7^L$ be the set of all OSI Application-layer protocols recognised by iptables [86]. Note, for ease of exposition we use syntactic sugar in the definition of $Proto_7^L$ as a free-type. We define:

$Proto_7^L$ ::= 100bao | aim | aimwebcontent | applejuice | ares | armagetron | audiogalaxy | battlefield1942 | battlefield2 | battlefield2142 | bgp | biff | bittorrent | chikka | cimd | ciscovpn | citrix | counterstrike-source | cvs | dayofdefeat-source | dazhihui | dhcp | directconnect | dns | doom3 | edonkey | fasttrack | finger | freenet | ftp | gkrellm | gnucleuslan | gnutella | goboogy | gopher | gtalk | guildwars | h323 | halflife2-deathmatch | hddtemp | hotline | http | http-rtsp | http-dap | http-freshdownload | http-itunes | httpaudio | httpcachehit | httpcachemiss | httpvideo | ident | imap | imesh | ipp | irc | jabber | kugoo | live365 | liveforspeed | lpd | mohaa | msn-filetransfer | msnmessenger | mute | napster | nbns | ncp | netbios | nntp | ntp | openft | pcanywhere | poco | pop3 | pplive | pressplay | qq | quicktime | quake-halflife | quake1 | radmin | rdp | replaytv-ivs | rlogin | rtp | rtsp | runesofmagic | shoutcast | sip | skypeout | skypetoskype | smb | smtp | snmp | snmp-mon | snmp-trap | socks | soribada | soulseek | ssdp | ssh | ssl | stun | subspace | subversion | teamfortress2 | teamspeak | telnet | tesla | tftp | thecircle | tonghuashun | tor | tsp | uucp | validcertssl | ventrilo | vnc | whois | worldofwarcraft | x11 | xboxlive | xunlei | yahoo | zmaap

The Layer 7 protocol definitions specify how these protocol names correspond to regular expressions that are matched by Netfilter on the packet Application Layer data. For example, the name smb corresponds to the signature [85]:

```
\xffsmb[\x72\x25]
```

Packet-owner/creator Filtering. For locally generated packets; iptables allows filtering by various characteristics of the packet creator through the *owner* module. Filtering is applied to the OUTPUT chain of the filter table [146].

Example 10 The following iptables rule specifies that outbound (`OUTPUT`) TCP packets (`-p tcp`) originating from the Linux application UID 1001 (`-m owner --uid-owner 1001`) destined to the IP address 0.0.0.1 (`-d 0.0.0.1`) on network port 80 (`--dport 80`) will be permitted traversal of the firewall (`-j ACCEPT`).

```
iptables -A OUTPUT -p tcp -m owner --uid-owner 1001 \
    -d 0.0.0.1 --dport 80 -j ACCEPT
```

Similarly, the following iptables rule specifies that outbound (`OUTPUT`) TCP packets (`-p tcp`) originating from the Linux application GID 1010 (`-m owner`

`--gid-owner 1010`) destined to the IP address 0.0.0.1 (`-d 0.0.0.1`) on port 80 (`--dport 80`) will be allowed (`-j ACCEPT`).

```
iptables -A OUTPUT -p tcp -m owner --gid-owner 1010 \
    -d 0.0.0.1 --dport 80 -j ACCEPT
```

△

Let *UID* be the set of 32-Bit UIDs for a Linux system, given as the set of all natural numbers from $0 \mathinner{.\,.} \mathsf{maxUID}$. We define:

$$\mathsf{maxUID} == 2^{32} - 1$$

$$\mathit{UID} == \{ i : \mathbb{N} \mid 0 \leq i \leq \mathsf{maxUID} \}$$

Similarly, let *GID* be the set of 32-Bit GIDs for a Linux system, given as the set of all natural numbers from $0 \mathinner{.\,.} \mathsf{maxGID}$, whereby:

$$\mathsf{maxGID} == 2^{32} - 1$$

$$\mathit{GID} == \{ i : \mathbb{N} \mid 0 \leq i \leq \mathsf{maxGID} \}$$

3.2.5 Additional Filtering Specifications

State. When filtering via a stateful firewall, the firewall's state-table automatically manages TCP flags [152]. The iptables conntrack module defines the state extension [146]. The state extension allows access to the connection tracking state for a packet. The notion of state in iptables is an abstraction, where the literals NEW, ESTABLISHED, RELATED, INVALID and UNTRACKED, signify user-land 'states' that packets within tracked connections can be related to. This is enabled by the conntrack framework within the Linux kernel.

Example 11 The previous four iptables rules from Example 8 that enable the TCP three-way handshake using TCP Flags may be rewritten as follows. Note, these two rules also allow for the TCP connection teardown.

```
iptables -A INPUT -p tcp -s 0.0.0.1 -d 0.0.0.2 --dport 80 \
    -m state --state NEW, ESTABLISHED  -j ACCEPT

iptables -A OUTPUT -p tcp -d 0.0.0.1 -s 0.0.0.2 --sport 80 \
    -m state --state ESTABLISHED  -j ACCEPT
```

An advantage of specifying state in this way, is that it allows end-users to construct firewall rules that abstract from the low-level operational details of the specified network protocol. The NEW state signifies the packet has started a new connection, or is otherwise associated with a connection that has not seen packets in both directions. An ESTABLISHED state signifies the packet is associated with a connection which has seen packets in both directions. The state RELATED signifies the packet is starting a new connection, but is associated with an existing connection, for example, an FTP data transfer, or an ICMP error. The INVALID state signifies that the packet is associated with no known connection. A state of UNTRACKED signifies that the packet is not tracked [68]. △

Let *State* be the set of connection tracking states for a packet/connection. We define:

$$State ::= \mathsf{new} \mid \mathsf{established} \mid \mathsf{related} \mid \mathsf{invalid} \mid \mathsf{untracked}$$

In this dissertation, the proposed model of state is based on the expressiveness of the iptables state module. A limitation of the approach, is that it is not possible to reason about complex stateful anomalies, such as those considered in [66], where a connection-oriented protocol is modelled using general automata with the permitted protocol states and transitions encoded.

Time-based Filtering. iptables allows for a filtering decision to be made if the packet arrival time/date is within a given range. The possible time range expressible in a rule is 1970-01-01T00:00:00 to 2038-01-19T04:17:07 [49], and is specified in ISO 8601 "T" notation [76].

Example 12 The following iptables rule specifies that for all inbound (`INPUT`) packets from the IP address 0.0.0.1 (`-s 0.0.0.1`), arriving between 8 a.m. January 1st 2017 and 6 p.m. January 3rd 2017 (`-m time --datestart 2017-01-01T08:00:00 --datestop 2017-01-03T18:00:00`), are to be denied.

```
iptables -A INPUT -s 0.0.0.1 -m time --datestart \
    2017-01-01T08:00:00 --datestop 2017-01-03T18:00:00 -j DROP
```

△

Let *Time* be the set of all Unix timestamps from 1970-01-01T00:00:00 up to and including 2038-01-19T04:17:07, given as the set of all natural numbers from

$0 \mathrel{..} \mathcal{T}(\textsf{2038-01-19T04:17:07})$, where $\mathcal{T}(\textsf{date})$ is a function that converts a date specified in "T" notation to a Unix timestamp. We define:

$$\textsf{maxTime} == \mathcal{T}(\textsf{2038-01-19T04:17:07})$$

$$\mathit{Time} == \{i : \mathbb{N} \mid 0 \leq i \leq \textsf{maxTime}\}$$

When reasoning about stateful/time-based policy anomalies in later chapters, we use only the anomaly definitions from [5, 6], explored in Chapter 2 Section 2.3.1, that consider redundancy, shadowing, etc., and do not consider more complex stateful anomalies, such as those considered in [66].

Network Interfaces and Direction-oriented Filtering. iptables allows for a filtering decision to be made with respect to the interface a network packet is arriving at/leaving through.

Example 13 The following iptables rule specifies that inbound (`INPUT`) network traffic arriving on the loopback interface (`-i lo`) from the local-loopback address range (`-s 127.0.0.1/8`) be allowed (`-j ACCEPT`).

```
iptables -A INPUT -i lo -s 127.0.0.1/8 -j ACCEPT
```

Similarly, the following iptables rule specifies that that outbound (`OUTPUT`) network traffic departing through the loopback interface (`-o lo`), destined for the local-loopback address range (`-d 127.0.0.1/8`) be allowed (`-j ACCEPT`).

```
iptables -A OUTPUT -o lo -d 127.0.0.1/8 -j ACCEPT
```

△

Let basic type *IFACE* be the set of all interfaces on a machine, where for simplicity, we assume elements of *IFACE* resemble lo, eth0, wlan0, tun0, etc. We define:

$$[\mathit{IFACE}]$$

Network traffic can be inbound or outbond on an interface. Direction-oriented filtering is defined as *Dir*, whereby:

$$\mathit{Dir} ::= \textsf{ingress} \mid \textsf{egress}$$

iptables Chains. Let *Chain* define the set of chains for the iptables filter table, we define:

$$Chain ::= \mathsf{input} \mid \mathsf{output} \mid \mathsf{forward}$$

The following example demonstrates the enforcement of a *default deny* policy on the INPUT, OUTPUT AND FORWARD chains, respectively.

Example 14 The following iptables rules specify that all network traffic not matched by a rule on the INPUT, OUTPUT AND FORWARD chains of the filter table be denied.

```
iptables -P INPUT -j DROP

iptables -P OUTPUT -j DROP

iptables -P FORWARD -j DROP
```

△

3.3 Discussion

In this chapter, firewall rule filter condition attributes for IP addresses, ports, and the TCP, UDP and ICMP protocols are defined. The specification includes definitions for TCP flags and ICMP Types/Codes. A notion of state has been encoded also. For the OSI Data Link Layer, attributes for packet-type and MAC address are defined. At the OSI Application Layer, filter condition attributes for Linux UIDs and GIDs, and the various application-layer protocols recognised by iptables are specified. Additional filtering specifications have also been developed. In Chapter 4, we use these filter condition attributes to develop a formal model of filtering constraints for the $\mathcal{FW}_0$ firewall policy algebra. Policies in the $\mathcal{FW}_0$ algebra are defined in terms of constraints on individual IPs, ports, protocols and the additional filter condition attributes defined in this chapter. In Chapter 5, range-based versions of the filter condition attributes defined in this chapter are formally specified for use in the $\mathcal{FW}_1$ firewall policy algebra.

Chapter 4

The $\mathcal{FW}_0$ Policy Model

The objective of this chapter is to develop a firewall policy algebra $\mathcal{FW}_0$, for constructing and reasoning over anomaly-free policies. Firewall rules are constructed using the filter condition attributes defined in Chapter 3. $\mathcal{FW}_0$ defines a simple model of iptables rules that do not consider range-based filter condition attributes; the purpose of developing $\mathcal{FW}_0$ is to demonstrate the utility of building such an algebra. In Chapter 5, the policy algebra $\mathcal{FW}_1$ defines a firewall policy in terms of rules constraining range-based filter conditions. The $\mathcal{FW}_0$ policy algebra in this chapter is an extended version of the algebra in [62]. The chapter is organised as follows. In Section 4.1 the packet-based filter condition attribute datatypes are defined. Section 4.2 defines the $\mathcal{FW}_0$ firewall policy algebra and Section 4.3 demonstrates how the algebra can be used in practice to detect anomalies in firewall policies.

4.1 Packet Attribute Datatypes

In this section, filter condition attribute datatypes for the $\mathcal{FW}_0$ policy model are defined. Examples of each attribute instance are given, and these are used to construct network packets at the end of this section for use in the running example given in Section 4.2.

OSI Layer 2. Let $\mathcal{L}_2$ define the set of all attributes of interest at Layer 2 of the OSI model, given as the set of all tuples over the set of all packet-types (*PktTpe*) and the set of all MAC address (*MAC*). We define:

$$\mathcal{L}_2 == \mathit{PktTpe} \times \mathit{MAC}$$

Example 1 The Data Link Layer attribute $\mathsf{l_{2A}} \in \mathcal{L}_2$ is defined, whereby:

$$\mathsf{l_{2A}} == (\mathsf{unicast}, \mathsf{00{:}0F{:}EA{:}91{:}04{:}08})$$

△

OSI Layer 7. Let $\mathcal{L}_7$ define the set of all attributes of interest at Layer 7 of the OSI model, given as the set of all three-tuples over the set of all Layer 7 protocols recognisable by iptables (Proto_7^L), the set of all Linux UIDs (UID) and the set of all Linux GIDs (GID). We define:

$$\mathcal{L}_7 == \mathit{Proto}_7^L \times \mathit{UID} \times \mathit{GID}$$

Example 2 The Application Layer attributes $\mathsf{l_{7A}}, \mathsf{l_{7B}}, \mathsf{l_{7C}} \in \mathcal{L}_7$ are given as:

$$\mathsf{l_{7A}} == (\mathsf{http}, \mathsf{1001}, \mathsf{1010})$$
$$\mathsf{l_{7B}} == (\mathsf{ftp}, \mathsf{2002}, \mathsf{2020})$$
$$\mathsf{l_{7C}} == (\mathsf{telnet}, \mathsf{3003}, \mathsf{3030})$$

△

The Stateful/Protocol Datatype. Let *Protocol* define the set of all stateful protocols, given as the set of all four-tuples over TCP flags ($\mathit{Flag}_{\mathit{Spec}}$), the UDP protocol ($\mathit{UDP}$), the set of all ICMP Type/Codes ($\mathit{TypesCodes}$), and the set of all connection tracking states for a packet/connection (State). We define:

$$\mathit{Protocol} == \mathit{Flag}_{\mathit{Spec}} \times \mathit{UDP} \times \mathit{TypesCodes} \times \mathit{State}$$

Example 3 The attribute instance $\mathsf{proto_A} \in \mathit{Protocol}$ is given as:

$$\mathsf{proto_A} == ((\{\mathsf{syn}\}, \{\mathsf{syn}\}), 0, \emptyset, \mathsf{new})$$

△

Additional Filtering Specifications. Let *AdditionalFC* define the set of all additional filter condition attributes of interest, given as the set of all four-tuples over the set of all Unix timestamps expressible using ISO 8601 “T” notation in an iptables rule (Time), the set of all network interfaces (IFACE), the set of directions for direction-oriented filtering (Dir) and the set of iptables chains (Chain).

$$\mathit{AdditionalFC} == \mathit{Time} \times \mathit{IFACE} \times \mathit{Dir} \times \mathit{Chain}$$

Example 4 The attribute instance $\mathsf{a_A} \in \mathit{AdditionalFC}$ is defined, whereby:

$$\mathsf{a_A} == (\mathsf{1474404256}, \mathsf{egress}, \mathsf{eth0}, \mathsf{output})$$

△

Network Packets. A network packet header is a eight-tuple (s, $sprt$, d, $dprt$, $proto$, l_2, l_7, a), representing network traffic originating from source IP address s, with source port number $sprt$, destined for destination IP address d, with destination port number $dprt$, using a stateful protocol $proto$, with additional Layer 2 attributes l_2, additional Layer 7 attributes l_7 and additional filtering specifications a. Let *Packet* define the set of all packet headers:

$$\mathit{Packet} == \mathit{IP} \times \mathit{Ports} \times \mathit{IP} \times \mathit{Ports} \times \mathit{Protocol} \times \mathcal{L}_2 \times \mathcal{L}_7 \times \mathit{AdditionalFC}$$

Example 5 The packets $\mathsf{http_1}, \mathsf{ftp_1}, \mathsf{tel_1} \in \mathit{Packet}$ are given as:

$$\mathsf{http_1} == (1, 1025, 10, 80, \mathsf{proto_A}, \mathsf{l_{2A}}, \mathsf{l_{7A}}, \mathsf{a_A})$$
$$\mathsf{ftp_1} == (1, 1025, 11, 21, \mathsf{proto_A}, \mathsf{l_{2A}}, \mathsf{l_{7B}}, \mathsf{a_A})$$
$$\mathsf{tel_1} == (1, 1025, 10, 23, \mathsf{proto_A}, \mathsf{l_{2A}}, \mathsf{l_{7C}}, \mathsf{a_A})$$

△

4.2 The $\mathcal{FW}_0$ Firewall Algebra

In this section, an algebra for constructing and reasoning about anomaly-free firewall policies is defined. We focus on stateful and stateless firewall policies that are defined in terms of constraints on individual IP addresses, ports, TCP, UDP and ICMP protocols, and additional filter condition attributes. A firewall policy defines the packets that may be allowed or denied by a firewall. Let Policy_0 define the set of all firewall policies, whereby:

$$\mathit{Policy}_0 == \{A, D : \mathbb{P}\,\mathit{Packet} \mid A \cap D = \emptyset\}$$

A firewall policy $(A, D) \in \mathit{Policy}_0$ defines that a packet $p \in A$ should be allowed by the firewall, while a packet $p \in D$ should be denied by the firewall. Given $(A, D) \in \mathit{Policy}_0$ then A and D are disjoint: this avoids any contradiction in deciding whether a packet should be allowed or denied. The policy accessor functions *allow* and *deny* are analogous to functions *first* and *second* for ordered pairs:

$$
\begin{array}{|l}
\mathit{allow}, \\
\mathit{deny} : \mathit{Policy}_0 \rightarrow \mathbb{P}\,\mathit{Packet} \\
\hline
\forall\, a, d : \mathbb{P}\ \mathit{Packet} \bullet \\
\qquad \mathit{allow}\,(a, d) = a \land \\
\qquad \mathit{deny}\,(a, d) = d
\end{array}
$$

Thus, we have for all $P \in \mathit{Policy}_0$ then $P = (\mathit{allow}(P), \mathit{deny}(P))$.

Lemma 4.2.1 *Policy_0 defines the set of anomaly-free policies.*

Proof Given a policy $(A, D) \in \mathit{Policy}_0$, as A and D are *sets* of individual packets, then A has no redundancy and D has no redundancy. Therefore, all packets allowed by the policy are distinct, as are all packets that are denied by the policy. Given that A and D are disjoint, then a policy P has no shadowing.

$$
\begin{array}{l}
\forall\, P : \mathit{Policy}_0 \bullet \\
\qquad \mathit{allow}(P) \cap \mathit{deny}(P) = \emptyset
\end{array}
$$

Given that all packets in P are distinct and P has no shadowing, then P has no generalisations and P has no correlations. Therefore, as P has no redundancy/shadowing/generalisation/correlation, then Policy_0 defines the set of *anomaly-free* policies. ▮

Note that $(A, D) \in \mathit{Policy}_0$ need not partition *Packet*: the allow and deny sets define the packets to which the policy *explicitly* applies and an *implicit* default decision is applied for those packets in $\mathit{Packet} \setminus (A \cup D)$. For the purposes of modeling iptables firewall policies it is sufficient to assume *default deny*, though we observe that the $\mathcal{FW}_0$ algebra can also be used to reason about *default allow* firewall policies.

Example 6 Consider the following network security policy that we will refer to throughout this section, where traffic from an administrators subnet is permitted access to a HTTP and a FTP server, and all Telnet traffic to the HTTP server is to be denied. The packets that are used to define this policy include http_1, ftp_1, $\mathsf{tel}_1 \in \mathit{Packet}$ from Example 5. A firewall policy $\mathsf{Admin}_1 \in \mathit{Policy}_0$ allows the HTTP and FTP packets while denying the Telnet traffic:

$$\mathsf{Admin}_1 == (\{\mathsf{http}_1, \mathsf{ftp}_1\}, \{\mathsf{tel}_1\})$$

△

Policy Refinement. An ordering can be defined over firewall policies, whereby given $P, Q \in \mathit{Policy}_0$ then $P \sqsubseteq Q$ means that P is no less restrictive than Q, that is, any packet that is denied by Q is denied by P. Intuitively, policy P is considered to be a *safe replacement* for policy Q in the sense of [56, 57, 77], and any firewall that enforces policy Q can be reconfigured to enforce policy P without any loss of security. The safe replacement ordering is defined as follows.

$$\begin{array}{|l}
\mathcal{FW}_0 \\
\bot, \top : \mathit{Policy}_0 \\
\_ \sqsubseteq \_ : \mathit{Policy}_0 \leftrightarrow \mathit{Policy}_0 \\
\_ \sqcap \_, \\
\_ \sqcup \_ : \mathit{Policy}_0 \times \mathit{Policy}_0 \rightarrow \mathit{Policy}_0 \\
\hline
\bot = (\emptyset, \mathit{Packet}) \land \top = (\mathit{Packet}, \emptyset) \\
\forall P, Q : \mathit{Policy}_0 \bullet \\
\qquad P \sqsubseteq Q \Leftrightarrow (\mathit{allow}(P) \subseteq \mathit{allow}(Q)) \land \\
\qquad\qquad\qquad (\mathit{deny}(P) \supseteq \mathit{deny}(Q)) \land \\
\qquad P \sqcap Q = (\mathit{allow}(P) \cap \mathit{allow}(Q), \\
\qquad\qquad\qquad \mathit{deny}(P) \cup \mathit{deny}(Q)) \land \\
\qquad P \sqcup Q = (\mathit{allow}(P) \cup \mathit{allow}(Q), \\
\qquad\qquad\qquad \mathit{deny}(P) \cap \mathit{deny}(Q)) \\
\hline
\end{array}$$

Formally, $P \sqsubseteq Q$ iff every packet allowed by P is allowed by Q and that any packet explicitly denied by Q is also explicitly denied by P. Note that in this definition we distinguish between packets *explicitly* denied in the policy versus packets *implicitly* denied by default. This means that, everything else being equal, a policy that explicitly denies a packet is considered more restrictive than a policy that relies on the implicit default-deny for the same packet; we shall see that this distinction is important when safely extending policies with new rules. Safe replacement is defined as the Cartesian product of subset orderings over allow and deny sets, and it therefore follows that Policy_0 is a partially ordered set under $\sqsubseteq$. $\bot$ and $\top$ define the most restrictive and least restrictive policies, that is, for any $P \in \mathit{Policy}_0$ we have $\bot \sqsubseteq P \sqsubseteq \top$. Thus, for example, any firewall enforcing a policy P can be safely reconfigured to enforce the (not very useful) firewall policy $\bot$.

Theorem 4.2.2 *The set of all policies $Policy_0$ forms a lattice under safe replacement, with greatest lower bound ($\sqcap$) and lowest upper bound ($\sqcup$) operators in $\mathcal{FW}_0$.*

Proof The powerset ordering of packets is a lattice under subset, the Cartesian product is a lattice under the definitions of glb and lub, therefore, $\mathcal{FW}_0$ is a lattice. ▮

Example 7 Consider, an update to the network security policy Admin_1, where SSH traffic is to be denied to the HTTP and FTP servers from a malicious IP address. Some packets to be considered as part of this access restriction are mal_1, $\mathsf{mal}_2 \in Packet$, whereby:

$$\begin{aligned}\mathsf{mal}_1 &== (5, 1025, 10, 22, \mathsf{proto}_\mathsf{A}, (\mathsf{unicast}, \emptyset), (\mathsf{ssh}, \emptyset, \emptyset),\\ &\quad (1474404256, \mathsf{ingress}, \mathsf{eth0}, \mathsf{input}))\\ \mathsf{mal}_2 &== (5, 1025, 11, 22, \mathsf{proto}_\mathsf{A}, (\mathsf{unicast}, \emptyset), (\mathsf{ssh}, \emptyset, \emptyset),\\ &\quad (1474404256, \mathsf{ingress}, \mathsf{eth0}, \mathsf{input}))\end{aligned}$$

A firewall policy $\mathsf{Admin}_2 \in Policy_0$ extends the previous policy Admin_1 to deny these packets from this malicious host:

$$\mathsf{Admin}_2 == (\{\mathsf{http}_1, \mathsf{ftp}_1\}, \{\mathsf{tel}_1, \mathsf{mal}_1, \mathsf{mal}_2\})$$

We observe that Admin_2 safely replaces Admin_1: $\mathsf{Admin}_2 \sqsubseteq \mathsf{Admin}_1$, but Admin_1 is not a safe replacement for Admin_2. △

Policy Intersection. Under this ordering, the *meet*, or intersection $P \sqcap Q$, of two firewall policies P and Q is defined as the policy that denies any packet that is explicitly denied by *either* P or Q, but allows packets that are allowed by *both* P and Q. Intuitively, this means that if a firewall is required to enforce both policies P and Q then it can be configured to enforce the policy $P \sqcap Q$, since $P \sqcap Q$ is a safe replacement for both P and Q, that is $(P \sqcap Q) \sqsubseteq P$ and $(P \sqcap Q) \sqsubseteq Q$. Thus, $P \sqcap Q$ provides the 'best'/least restrictive safe replacement for both P and Q under $\sqsubseteq$.

Example 8 Consider, a policy $\mathsf{Admin}_3 \in Policy_0$, and packets ssh_1, ssh_2 and $\mathsf{mal}_3 \in Packet$, whereby:

$$\begin{aligned}\mathsf{ssh}_1 &== (1, 1025, 10, 22, \mathsf{proto}_\mathsf{A}, (\mathsf{unicast}, \emptyset), (\mathsf{ssh}, \emptyset, \emptyset),\\ &\quad (1474404256, \mathsf{ingress}, \mathsf{eth0}, \mathsf{input}))\end{aligned}$$

$$\begin{aligned}&\mathsf{ssh}_2 == (1, 1025, 11, 22, \mathsf{proto_A}, (\mathsf{unicast}, \emptyset), (\mathsf{ssh}, \emptyset, \emptyset),\\&\qquad (1474404256, \mathsf{ingress}, \mathsf{eth0}, \mathsf{input}))\end{aligned}$$

$$\mathsf{mal}_3 == (1, 1025, 5, 6667, \mathsf{proto_A}, \mathsf{l_{2A}}, (\mathsf{irc}, \emptyset, \emptyset), \mathsf{a_A})$$

The policy Admin_3 permits the administrators SSH access to the HTTP and FTP servers, and denies all network traffic from the client destined for a malicious IP, according to packets ssh_1, ssh_2 and $\mathsf{mal}_3 \in \mathit{Packet}$.

$$\mathsf{Admin}_3 == (\{\mathsf{ssh}_1, \mathsf{ssh}_2\}, \{\mathsf{mal}_3\})$$

We have $\mathsf{Admin}_2 \sqcap \mathsf{Admin}_3 = (\emptyset,\ \{\mathsf{tel}_1, \mathsf{mal}_1, \mathsf{mal}_2, \mathsf{mal}_3\})$, a safe replacement for both policies Admin_2 and Admin_3 under $\sqsubseteq$. △

Policy Union. The *join* of two firewall policies P and Q is defined as the policy that allows any packet allowed by *either* P or Q, but denies packets that are explicitly denied by *both* P and Q. For example, we have $\mathsf{Admin}_2 \sqcup \mathsf{Admin}_3$ $= (\{\{\mathsf{http}_1, \mathsf{ftp}_1, \mathsf{ssh}_1, \mathsf{ssh}_2\}\},\ \emptyset)$. Intuitively, this means that a firewall that is required to enforce either policy P or Q can be safely configured to enforce the policy $P \sqcup Q$ since $\sqcup$ provides a lowest upper bound operator, and we have $P \sqsubseteq (P \sqcup Q)$ and $Q \sqsubseteq (P \sqcup Q)$.

4.2.1 Lattice Properties of the $\mathcal{FW}_0$ Algebra

The ordering relation $\sqsubseteq$ is a non-strict partial order over Policy_0; given the properties of properties of reflexivity, transitivity and antisymmetry [18].

$$\begin{aligned}&\forall\, \mathcal{FW}_0;\ P, Q, R : \mathit{Policy}_0 \bullet\\&\qquad P \sqsubseteq P \land\\&\qquad (P \sqsubseteq Q \land Q \sqsubseteq R \Rightarrow P \sqsubseteq R) \land\\&\qquad (P \sqsubseteq Q \land Q \sqsubseteq P \Rightarrow P = Q)\end{aligned}$$

The ordering relation $\sqsubseteq$ is reflexive, as any policy should be a safe replacement for itself. If for any $P \in \mathit{Policy}_0$, $P \sqsubseteq P$ did not hold, then an inconsistency would exist. Transitivity follows from a similar requirement, where for any $P, Q, R \in \mathit{Policy}_0$, if P safely replaces Q and Q safely replaces R, then P should also safely replace R. Antisymmetry follows from the assumption of irredundant policies, where if P is a safe replacement for Q and Q is a safe replacement for P, then one of them is unnecessary.

Figure 4.1 depicts a lattice fragment/partial Hasse diagram, for composition of Admin_2 and Admin_3, under the relative ordering of $\sqsubseteq$ over $Policy_0$.

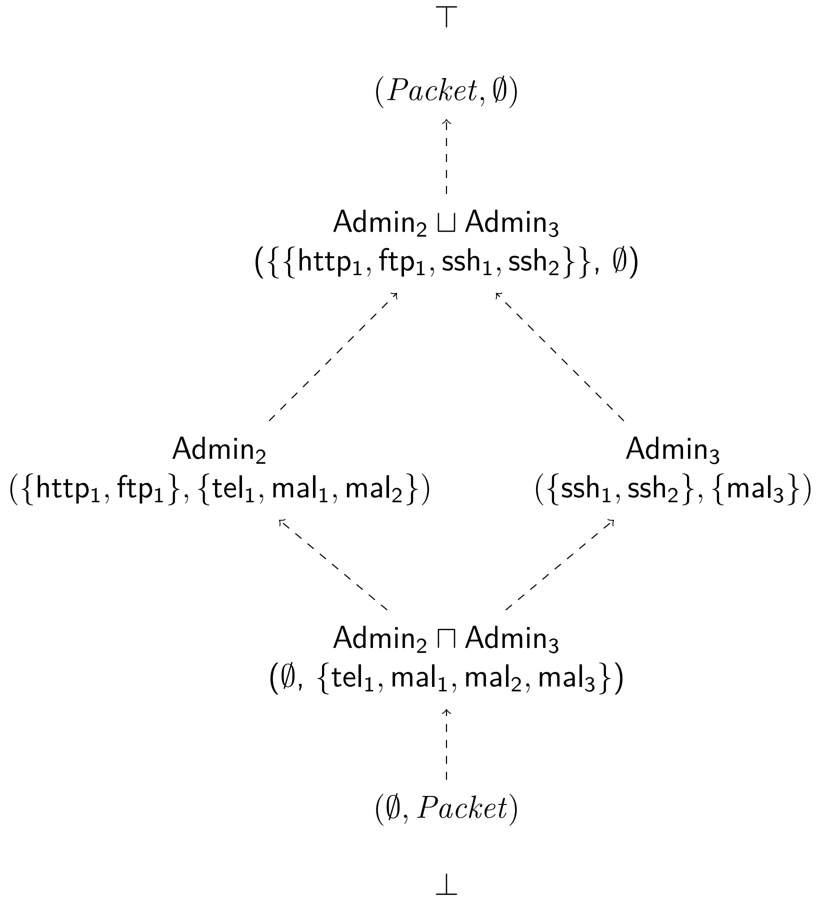

$\top$
$(Packet, \emptyset)$
$\mathsf{Admin}_2 \sqcup \mathsf{Admin}_3$
$(\{\{\mathsf{http}_1, \mathsf{ftp}_1, \mathsf{ssh}_1, \mathsf{ssh}_2\}\}, \emptyset)$
Admin_2
$(\{\mathsf{http}_1, \mathsf{ftp}_1\}, \{\mathsf{tel}_1, \mathsf{mal}_1, \mathsf{mal}_2\})$
Admin_3
$(\{\mathsf{ssh}_1, \mathsf{ssh}_2\}, \{\mathsf{mal}_3\})$
$\mathsf{Admin}_2 \sqcap \mathsf{Admin}_3$
$(\emptyset, \{\mathsf{tel}_1, \mathsf{mal}_1, \mathsf{mal}_2, \mathsf{mal}_3\})$
$(\emptyset, Packet)$
$\perp$

Figure 4.1: $\mathcal{FW}_0$ lattice fragment

Commutative Laws. We observe that changing the order of the policies/-operands does not change the composition result.

$$\begin{aligned}
&\forall\, \mathcal{FW}_0;\ P, Q : Policy_0 \bullet \\
&\qquad P \sqcup Q = Q \sqcup P \land \\
&\qquad P \sqcap Q = Q \sqcap P
\end{aligned}$$

Associative Laws. We observe that the order in which the operations are performed does not change the outcome of the operation.

$$\begin{aligned}
&\forall\, \mathcal{FW}_0;\ P, Q, R : Policy_0 \bullet \\
&\qquad P \sqcup (Q \sqcup R) = (P \sqcup Q) \sqcup R \land \\
&\qquad P \sqcap (Q \sqcap R) = (P \sqcap Q) \sqcap R
\end{aligned}$$

Absorption Laws. The following identities link $\sqcup$ and $\sqcap$.

$$\begin{array}{l}\forall\,\mathcal{FW}_0;\ P, Q : \mathit{Policy}_0 \bullet \\ \qquad P \sqcup (P \sqcap Q) = P \land \\ \qquad P \sqcap (P \sqcup Q) = P\end{array}$$

Idempotent Laws. We observe that for all $P \in \mathit{Policy}_0$, P is idempotent with respect to $\sqcup$ and $\sqcap$.

$$\begin{array}{l}\forall\,\mathcal{FW}_0;\ P : \mathit{Policy}_0 \bullet \\ \qquad P \sqcup P = P \land \\ \qquad P \sqcap P = P\end{array}$$

Identity Laws. We observe that $\mathcal{FW}_0$ is a bounded lattice/algebraic structure, such that $(\mathit{Policy}, \sqcup, \sqcap)$ is a lattice, $\bot$ is the identity element for the join operation $\sqcup$, and $\top$ is the identity element for the meet operation $\sqcap$.

$$\begin{array}{l}\forall\,\mathcal{FW}_0;\ P : \mathit{Policy}_0 \bullet \\ \qquad P \sqcup \bot = P \land \\ \qquad P \sqcap \top = P\end{array}$$

4.2.2 Constructing Firewall Policies

The lattice of policies $\mathcal{FW}_0$ provides us with an algebra for constructing and interpreting firewall polices. The following constructor functions are used to build primitive policies.

Given a set of packets A then $(\mathsf{Allow}\,A)$ is a policy that allows packets in A, and $(\mathsf{Deny}\,D)$ is a policy that explicitly denies packets in D.

$$\begin{array}{|l}\mathsf{Allow}, \\ \mathsf{Deny} : \mathbb{P}\,\mathit{Packet} \rightarrow \mathit{Policy}_0 \\ \hline \forall\, S : \mathbb{P}\,\mathit{Packet} \bullet \\ \qquad \mathsf{Allow}\, S = (S, \emptyset) \land \\ \qquad \mathsf{Deny}\, S = (\emptyset, S)\end{array}$$

This provides what we refer to as a *weak* interpretation of allow and deny: packets that are not explicitly mentioned in parameter S are default-deny and are therefore not specified in the deny set of the policy. The following provides us with a *strong* interpretation for these constructors:

$$
\begin{array}{|l}
\mathsf{Allow}^+, \\
\mathsf{Deny}^+ : \mathbb{P}\, Packet \to Policy_0 \\
\hline
\forall\, S : \mathbb{P}\, Packet \bullet \\
\quad \mathsf{Allow}^+\, S = (S, Packet \setminus S) \wedge \\
\quad \mathsf{Deny}^+\, S = (Packet \setminus S, S)
\end{array}
$$

In this case $(\mathsf{Allow}^+\, A)$ allows packets specified in A while explicitly denying all other packets, and $(\mathsf{Deny}^+\, D)$ denies packets specified in D while allowing all other packets.

A firewall policy $P \in Policy_0$ can be decomposed into its corresponding allow and deny policies and re-constructed using the algebra.

Lemma 4.2.3 *Given $A, D \in \mathbb{P}\, Packet$, then:*

$$(\mathsf{Allow}^+\, A) \sqcup (\mathsf{Deny}\, D) = (A, Packet \setminus A) \sqcup (\emptyset, D)$$

Proof

$$
\begin{aligned}
(\mathsf{Allow}^+\, A) \sqcup (\mathsf{Deny}\, D) &= (\mathsf{Allow}\, A) \sqcap (\mathsf{Deny}^+\, D) \\
&= (A, D)
\end{aligned}
$$

∎

Example 9 An alternative specification for policy Admin_3 is:

$$\mathsf{Admin}_3 == (\mathsf{Allow}^+\, (\{\mathsf{ssh}_1, \mathsf{ssh}_2\}) \sqcup \mathsf{Deny}\, (\{\mathsf{mal}_3\})$$

△

Corollary 4.2.4 *Given $A, D \in \mathbb{P}\, Packet$, then:*

$$(\mathsf{Deny}^+\, D) \sqcap (\mathsf{Allow}\, A) = (A, \emptyset) \sqcap (Packet \setminus D, D)$$

Proof It follows from Lemma 4.2.3 that Corollary 4.2.4 holds. ∎

4.3 Reasoning About Policies in Practice

Sequential Composition. A firewall policy is conventionally constructed as a sequence of rules, whereby for a given network packet, the decision to allow or deny the packet is checked against each policy rule, starting from the first, in sequence, and the first rule that matches gives the result that is returned. The algebra $\mathcal{FW}_0$ can be extended to include a similar form of sequential composition of policies. The policy constructions in Section 4.2.2 can be regarded as representing the individual rules of a conventional firewall policy.

Let $(\mathsf{Allow}\, A) \fatsemi Q$ denote a sequential composition of an allow rule $(\mathsf{Allow}\, A)$ with policy Q, with the interpretation that a packet that is matched by A is allowed; if it does not match A then policy Q is enforced. The resulting policy either: allows packets in A (and denies all other packets), or allows/denies packets according to policy Q. This is defined as:

$$\begin{aligned}(\mathsf{Allow}\, A) \fatsemi Q &= (\mathsf{Allow}^+\, A) \sqcup Q\\ &= ((A \cup \mathit{allow}(Q)), ((\mathit{Packet} \setminus A) \cap \mathit{deny}(Q)))\\ &= ((A \cup \mathit{allow}(Q)), (\mathit{deny}(Q) \setminus A))\end{aligned}$$

which is as expected. A similar definition can be provided for the sequential composition $(\mathsf{Deny}\, D) \fatsemi Q$, whereby a packet that is matched by D is denied; if it does not match D then policy Q is enforced. This is defined as:

$$\begin{aligned}(\mathsf{Deny}\, D) \fatsemi Q &= (\mathsf{Deny}^+\, D) \sqcap Q\\ &= (((\mathit{Packet} \setminus D) \cap \mathit{allow}(Q)), \mathit{deny}(Q) \cup D)\\ &= (\mathit{allow}(Q) \setminus D, \mathit{deny}(Q) \cup D)\end{aligned}$$

While in practice it is usual to write a firewall policy in terms of many constructions of allow and deny rules, in principle, any firewall policy $P \in \mathit{Policy}_0$ can be defined in terms of one allow policy $(\mathsf{Allow}\, \mathit{allow}(P))$ and one deny policy $(\mathsf{Deny}\, \mathit{deny}(P))$ and since the allow and deny sets of P are disjoint we have $P \fatsemi Q = (\mathsf{Deny}\, \mathit{deny}(P)) \fatsemi (\mathsf{Allow}\, \mathit{allow}(P)) \fatsemi Q$. We have:

$$\begin{array}{|l}
\_ \fatsemi \_ : \mathit{Policy}_0 \times \mathit{Policy}_0 \rightarrow \mathit{Policy}_0\\
\hline
\forall\, \mathcal{FW}_0;\ P, Q : \mathit{Policy}_0 \bullet\\
\qquad P \fatsemi Q = (\mathsf{Deny}^+\, (\mathit{deny}(P))) \sqcap\\
\qquad\qquad (Q \sqcup (\mathsf{Allow}^+\, (\mathit{allow}(P))))
\end{array}$$

Example 10 The policy Admin_4 is given as the sequential composition of Admin_2

and Admin_3. We have:

$$\begin{aligned}\mathsf{Admin}_4 &= \mathsf{Admin}_2 \fatsemi \mathsf{Admin}_3\\ &= (\{\mathsf{http}_1, \mathsf{ftp}_1\}, \{\mathsf{tel}_1, \mathsf{mal}_1, \mathsf{mal}_2\}) \fatsemi (\{\mathsf{ssh}_1, \mathsf{ssh}_2\}, \{\mathsf{mal}_3\})\\ &= (\mathit{Packet} \setminus \{\mathsf{mal}_1, \mathsf{mal}_2, \mathsf{tel}_1\}), \{\mathsf{mal}_1, \mathsf{mal}_2, \mathsf{tel}_1\}) \sqcap\\ &\qquad ((\{\mathsf{ssh}_1, \mathsf{ssh}_2\}, \{\mathsf{mal}_3\}) \sqcup\\ &\qquad\qquad (\{\mathsf{http}_1, \mathsf{ftp}_1\}, \mathit{Packet} \setminus \{\mathsf{http}_1, \mathsf{ftp}_1\}))\\ &= (\{\mathsf{ssh}_1, \mathsf{ssh}_2, \mathsf{http}_1, \mathsf{ftp}_1\}, \{\mathsf{tel}_1, \mathsf{mal}_1, \mathsf{mal}_2, \mathsf{mal}_3\})\end{aligned}$$

△

A firewall rule specifies a set of packets and an associated target action. Let *Rule* define the set of all packet-based firewall rules, whereby:

$$\begin{aligned}&\mathit{Rule} ::= \mathsf{allow}\, \langle\!\langle \mathbb{P}\, \mathit{Packet} \rangle\!\rangle \mid\\ &\qquad \mathsf{deny}\, \langle\!\langle \mathbb{P}\, \mathit{Packet} \rangle\!\rangle\end{aligned}$$

We define a rule interpretation function $\mathcal{I}$ as:

$$\begin{array}{|l}\mathcal{I} : \mathit{Rule} \rightarrow \mathit{Policy}_0\\ \hline \forall\, S : \mathbb{P}\, \mathit{Packet} \bullet\\ \qquad \mathcal{I}(\mathsf{allow}\, S) = \mathsf{Allow}\, S \land\\ \qquad \mathcal{I}(\mathsf{deny}\, S) = \mathsf{Deny}\, S\end{array}$$

A firewall policy is defined as a sequence of rules $\langle r_1, r_2, .., r_n \rangle$, for $r_i \in \mathit{Rule}$, and is encoded in the policy algebra as $\mathcal{I}(r_1) \fatsemi \mathcal{I}(r_2) \fatsemi .. \fatsemi \mathcal{I}(r_n)$.

Policy Negation. The policy negation of $P \in \mathit{Policy}_0$ allows packets explicitly denied by P and explicitly denies packets allowed by P. We define:

$$\begin{array}{|l}\mathbf{not} : \mathit{Policy}_0 \rightarrow \mathit{Policy}_0\\ \hline \forall\, \mathcal{FW}_0;\ P : \mathit{Policy}_0 \bullet\\ \qquad \mathbf{not}\ P = (\mathsf{Allow}^+\, (\mathit{deny}\, (P))) \sqcup\\ \qquad\qquad (\mathsf{Deny}\, (\mathit{allow}\, (P)))\end{array}$$

Example 11 The negation of policy Admin_4 is:

$$\begin{aligned}\textbf{not}\ \mathsf{Admin}_4 = {} & (\mathsf{Allow}^{+}\ (\{\mathsf{tel}_1, \mathsf{mal}_1, \mathsf{mal}_2, \mathsf{mal}_3\}) \sqcup \\ & (\mathsf{Deny}\ (\{\mathsf{ssh}_1, \mathsf{ssh}_2, \mathsf{http}_1, \mathsf{ftp}_1\})) \\ & = (\{\mathsf{tel}_1, \mathsf{mal}_1, \mathsf{mal}_2, \mathsf{mal}_3\}, \{\mathsf{ssh}_1, \mathsf{ssh}_2, \mathsf{http}_1, \mathsf{ftp}_1\})\end{aligned}$$

△

From this definition it follows that ($\textbf{not}\ P$) is simply $(\mathit{deny}\,(P), \mathit{allow}\,(P))$ and thus $\textbf{not}\,(\mathsf{Deny}\,D) = (\mathsf{Allow}\,D)$ and $\textbf{not}\,(\mathsf{Allow}\,A) = (\mathsf{Deny}\,A)$. Note however, that in general policy negation does not define a complement operator in the algebra Policy_0, that is, it not necessarily the case that $(P \sqcup \textbf{not}\ P) = \top$ and $(P \sqcap \textbf{not}\ P) = \bot$.

Example 12 Given the firewall policy Admin_4, then:

$$\begin{aligned}\mathsf{Admin}_4 \sqcup (\textbf{not}\ \mathsf{Admin}_4) &= (\{\mathsf{ssh}_1, \mathsf{ssh}_2, \mathsf{http}_1, \mathsf{ftp}_1, \mathsf{tel}_1, \mathsf{mal}_1, \mathsf{mal}_2, \mathsf{mal}_3\}, \emptyset) \\ &\neq \top\end{aligned}$$

and:

$$\begin{aligned}\mathsf{Admin}_4 \sqcap (\textbf{not}\ \mathsf{Admin}_4) &= (\emptyset, \{\mathsf{ssh}_1, \mathsf{ssh}_2, \mathsf{http}_1, \mathsf{ftp}_1, \mathsf{tel}_1, \mathsf{mal}_1, \mathsf{mal}_2, \mathsf{mal}_3\}) \\ &\neq \bot\end{aligned}$$

△

However, the sub-lattice of policies with allow and deny sets that exactly partition the same set $S \subseteq \mathit{Packet}$ has policy negation as complement ($\mathit{allow}\,(P) \cup \mathit{deny}\,(P) = S$ for all P in the sub-lattice).

Policy Projection. Useful policies can be constructed through filtering a policy by a given filter condition attribute. The projection operators $@^u$ and $@^d$ filter a policy by a set of IP addresses. Firstly, let $\mathcal{S}(S)$ give the set of all packets that have $s \in S$ as source IP, and similarly $\mathcal{D}(D)$ gives all packets with a destination IP address $d \in D$.

$$
\begin{array}{|l}
\mathcal{S}, \\
\mathcal{D} : \mathbb{P}\, IP \rightarrow \mathbb{P}\, Packet \\
\hline
\forall\, S, D : \mathbb{P}\, IP \bullet \\
\qquad \mathcal{S}(S) = S \times Port \times IP \times Port \times Protocol \times \mathcal{L}_2 \times \mathcal{L}_7 \times AdditionalFC\ \wedge \\
\qquad \mathcal{D}(D) = IP \times Port \times D \times Port \times Protocol \times \mathcal{L}_2 \times \mathcal{L}_7 \times AdditionalFC
\end{array}
$$

For a policy P and a set of IP addresses S, $P@^u S$ is the upstream projection of P, and consists of the allow and deny packets from P where each packet has as source IP some member of S. Similarly, $P@^d S$ is the downstream projection of P, it consists of the allow and deny packets from P whereby each packet has as destination IP some member of S.

$$
\begin{array}{|l}
\_@^u\_, \\
\_@^d\_ : Policy_0 \times \mathbb{P}\ IP \rightarrow Policy_0 \\
\hline
\forall\, P : Policy_0;\ S : \mathbb{P}\ IP \bullet \\
\qquad P\ @^u\ S = (allow(P) \cap \mathcal{S}(S), deny(P) \cap \mathcal{S}(S))\ \wedge \\
\qquad P\ @^d\ S = (allow(P) \cap \mathcal{D}(S), deny(P) \cap \mathcal{D}(S))
\end{array}
$$

Example 13 Suppose there is a requirement to specify all the packets allowed/-denied to the HTTP server by the policy Admin_4, then we have:

$$\mathsf{Admin}_4\ @^d\ \{10\} = (\{\mathsf{http}_1, \mathsf{ssh}_1\}, \{\mathsf{tel}_1, \mathsf{mal}_1\})$$

where $\mathsf{10}$ is the IP address of the HTTP server. △

4.3.1 Anomaly Detection

A firewall policy is conventionally constructed as a sequence of order-dependent rules, and when a network packet matches with two or more policy rules, the policy is anomalous [5, 6, 35]. By definition, the allow and deny sets of some $P \in Policy_0$ are disjoint, therefore P is anomaly-free by construction. We can however define anomalies using the algebra; by considering how a policy changes when composed with other policies.

Redundancy. A policy P is redundant given policy Q if their composition results in no difference between the resulting policy and Q, in particular, if:

$$P \fatsemi Q = Q$$

Example 14 Considering policies Admin_1 and Admin_2, we see that Admin_1 is redundant to Admin_2 since:

$$\mathsf{Admin}_1 \mathbin{;} \mathsf{Admin}_2 = \mathsf{Admin}_2$$

as:

$$(\{\mathsf{http}_1, \mathsf{ftp}_1\}, \{\mathsf{tel}_1\}) \mathbin{;} (\{\mathsf{http}_1, \mathsf{ftp}_1\}, \{\mathsf{tel}_1, \mathsf{mal}_1, \mathsf{mal}_2\}) = \\ (\{\mathsf{http}_1, \mathsf{ftp}_1\}, \{\mathsf{tel}_1, \mathsf{mal}_1, \mathsf{mal}_2\})$$

△

Further definitions may be given for redundancy. For example, there are redundant packets with a target action of allow between policies P and Q, if:

$$\mathsf{Allow}(\mathit{allow}\,(P)) \sqcap \mathsf{Allow}(\mathit{allow}\,(Q)) \neq (\emptyset, \emptyset)$$

as:

$$\mathsf{Allow}(\mathit{allow}\,(P)) \sqcap \mathsf{Allow}(\mathit{allow}\,(Q)) = (\mathit{allow}\,(P) \cap \mathit{allow}\,(Q), \emptyset)$$

A similar interpretation follows for redundant packets with a target action of deny between policies P and Q. We have redundant denies if:

$$\mathsf{Deny}(\mathit{deny}\,(P)) \sqcup \mathsf{Deny}(\mathit{deny}\,(Q)) \neq (\emptyset, \emptyset)$$

as:

$$\mathsf{Deny}(\mathit{deny}\,(P)) \sqcup \mathsf{Deny}(\mathit{deny}\,(Q)) = (\emptyset, \mathit{deny}\,(P) \cap \mathit{deny}\,(Q))$$

Shadowing. Some part of policy Q is shadowed by the entire policy P in the composition $P \mathbin{;} Q$ if the packet constraints that are specified by P contradict the constraints that are specified by Q, in particular, if:

$$(\mathbf{not}\ P) \mathbin{;} Q = Q$$

This is a very general definition for shadowing. Perhaps a more familiar interpretation of this definition is one where the policy P is a specific allow/deny rule that shadows a part or all of the policy with which it is composed. Recall that $(\mathbf{not}(\mathsf{Allow}\,A)) = (\mathsf{Deny}\,A)$ and, for example, in $(\mathsf{Allow}\,A) \mathbin{;} Q$ all or part of policy

Q is shadowed by the rule/primitive policy $(\mathsf{Allow}\, A)$ if Q denies the packets specified in A, that is, $(\mathsf{Deny}\, A) \fatsemi Q = Q$. Similarly, in $(\mathsf{Deny}\, D) \fatsemi Q$ part or all of policy Q is shadowed by the rule/primitive policy $(\mathsf{Deny}\, D)$ if $(\mathbf{not}\, (\mathsf{Deny}\, D)) \fatsemi Q = Q$.

Example 15 Consider, a rule that allows packets $\mathsf{tel}_1, \mathsf{mal}_1$ and mal_2, and has no deny constraints:

$$\mathsf{Allow}(\{\mathsf{tel}_1, \mathsf{mal}_1, \mathsf{mal}_2\})$$

Then this allow rule shadows the policy Admin_2 in the composition $\mathsf{Allow}(\{\mathsf{tel}_1, \mathsf{mal}_1, \mathsf{mal}_2\}) \fatsemi \mathsf{Admin}_2$, as:

$$\begin{aligned}&\mathsf{Deny}(\{\mathsf{tel}_1, \mathsf{mal}_1, \mathsf{mal}_2\}) \fatsemi (\{\mathsf{http}_1, \mathsf{ftp}_1\}, \{\mathsf{tel}_1, \mathsf{mal}_1, \mathsf{mal}_2\}) = \\ &\quad (\{\mathsf{http}_1, \mathsf{ftp}_1\}, \{\mathsf{tel}_1, \mathsf{mal}_1, \mathsf{mal}_2\})\end{aligned}$$

$\triangle$

Further definitions may also be given for shadowing. For example, we have that some of the packets denied by a policy P shadow some of the packets allowed by a policy Q if:

$$\mathsf{Deny}(\mathit{deny}\,(P)) \sqcup \mathsf{Deny}(\mathit{allow}\,(Q)) \neq (\emptyset, \emptyset)$$

as:

$$\mathsf{Deny}(\mathit{deny}\,(P)) \sqcup \mathsf{Deny}(\mathit{allow}\,(Q)) = (\emptyset, \mathit{deny}\,(P) \cap \mathit{allow}\,(Q))$$

Similarly, some of the packets allowed by a policy P shadow some of the packets denied by a policy Q if:

$$\mathsf{Allow}(\mathit{allow}\,(P)) \sqcap \mathsf{Allow}(\mathit{deny}\,(Q)) \neq (\emptyset, \emptyset)$$

as:

$$\mathsf{Allow}(\mathit{allow}\,(P)) \sqcap \mathsf{Allow}(\mathit{deny}\,(Q)) = (\mathit{allow}\,(P) \cap \mathit{deny}\,(Q), \emptyset)$$

Generalisation. A generalisation anomaly exists between P and Q if some of the packets allowed by P shadow some of the packets denied by Q, in particular, if:

$$\mathsf{Allow}(\mathit{allow}\,(P)) \sqcap \mathsf{Allow}(\mathit{deny}\,(Q)) \neq (\emptyset, \emptyset)$$

and, those packets shadowed in Q are a proper subset of Q's denies:

$$\textbf{not}\,(\mathsf{Allow}(\mathit{allow}\,(P)) \sqcap \mathsf{Allow}(\mathit{deny}\,(Q))) \neq \mathsf{Deny}(\mathit{deny}\,(Q))$$

whereby:

$$\textbf{not}\,(\mathsf{Allow}(\mathit{allow}\,(P)) \sqcap \mathsf{Allow}(\mathit{deny}\,(Q))) = (\emptyset, \mathit{allow}\,(P) \cap \mathit{deny}\,(Q))$$

Similarly, A generalisation anomaly exists between P and Q if some of the packets denied by P shadow some of the packets allowed by Q, in particular, if:

$$\mathsf{Deny}(\mathit{deny}\,(P)) \sqcup \mathsf{Deny}(\mathit{allow}\,(Q)) \neq (\emptyset, \emptyset)$$

and, those packets shadowed in Q are a proper subset of Q's allows:

$$\textbf{not}\,(\mathsf{Deny}(\mathit{deny}\,(P)) \sqcup \mathsf{Deny}(\mathit{allow}\,(Q))) \neq \mathsf{Allow}(\mathit{allow}\,(Q))$$

as:

$$\textbf{not}\,(\mathsf{Deny}(\mathit{deny}\,(P)) \sqcup \mathsf{Deny}(\mathit{allow}\,(Q))) = (\mathit{deny}\,(P) \cap \mathit{allow}\,(Q), \emptyset)$$

Inter-policy Anomalies. Anomalies can also occur between the different policies of distributed firewall configurations [6]. In the following, assume that P is a policy on an *upstream* firewall and Q is a policy on a *downstream* firewall.

Redundancy. An inter-redundancy anomaly exists between policies P and Q if some part of Q is redundant to some part of P, whereby the target action of the redundant filter conditions is *deny*. Given the set of packets A denied by P, and the set of packets B denied by Q, then there exists an inter-redundancy between P and Q, if:

$$(\mathsf{Deny}\; A) \fatsemi (\mathsf{Deny}\; B) = (\mathsf{Deny}\; A)$$

Further definitions may be given for inter-redundancy. For example, there are redundant packets with a target action of deny between upstream policy P and downstream policy Q, if:

$$\mathsf{Deny}(\mathit{deny}\,(P)) \sqcup \mathsf{Deny}(\mathit{deny}\,(Q)) \neq (\emptyset, \emptyset)$$

Shadowing. An inter-shadowing anomaly exists between policies P and Q if some part of Q's allows are shadowed by some part of P's denies. Given a set of packets A denied by P, and the set of packets B allowed by Q, then there is an inter-shadowing anomaly between P and Q, if:

$$(\mathsf{Deny}\, A) \fatsemi (\mathsf{Allow}\, B) = (\mathsf{Deny}\, A)$$

Further definitions may also be given for shadowing. For example, we have that some of the packets denied by a policy P shadow some of the packets allowed by a policy Q if:

$$\mathsf{Deny}(\mathit{deny}\,(P)) \sqcup \mathsf{Deny}(\mathit{allow}\,(Q)) \neq (\emptyset, \emptyset)$$

Spuriousness. An inter-spuriousness anomaly exists between policies P and Q if some part of Q's denies are shadowed by some part of P's allows. Given some set of packets A allowed by P, and some set of packets B denied by Q, then there exists an inter-spuriousness anomaly between P and Q, if:

$$(\mathsf{Allow}\, A) \fatsemi (\mathsf{Deny}\, B) = (\mathsf{Allow}\, A)$$

A further definition may also be given for spuriousness, whereby some of the packets allowed by a policy P shadow some of the packets denied by a policy Q. We have an inter-spuriousness anomaly from an upstream policy P to a downstream policy Q, if:

$$\mathsf{Allow}(\mathit{allow}\,(P)) \sqcap \mathsf{Allow}(\mathit{deny}\,(Q)) \neq (\emptyset, \emptyset)$$

4.4 Discussion

In this chapter, a policy algebra $\mathcal{FW}_0$ is defined whereby firewall policies can be specified and reasoned about. At the heart of this algebra is the notion of safe replacement, that is, whether it is secure to replace one firewall policy by another. The set of policies form a lattice under safe replacement and this enables consistent operators for safe composition to be defined. Policies in this lattice are anomaly-free by construction, and thus, composition under greatest lower and lowest upper bound operators preserves anomaly-freedom. A policy sequential composition operator is also proposed that can be used to interpret firewall policies defined more conventionally as sequences of firewall rules. The algebra can be used to characterize anomalies, such as redundancy and shadowing, that arise from policy composition.

The proposed algebra $\mathcal{FW}_0$ provides a semantics for firewall policies. While useful for the purposes of reasoning, it is not efficient to naïvely implement the algebra since a policy is defined in terms of rules constraining *individual* IP addresses and ports. For example, a policy constraining access from a subnet range `192.168.*.*` involves more than 65K individual packet rules, whatever about the impact of combining these with further constraints on destination IPs and ports. In practice, firewall rules are defined in terms of *ranges* of IP addresses and ports. Extending the policy algebra to include ranges is non-trivial, as is illustrated using the following example.

Example 16 Given a policy defined in terms of (allow and deny) sets of source/destination IP address and port ranges, whereby the policy

$$P_1 == (\{([1 \mathinner{.\,.} 3], [1 \mathinner{.\,.} 3], [1 \mathinner{.\,.} 3], [1 \mathinner{.\,.} 3])\}, \emptyset)$$

has no deny constraints ($\emptyset$) and has one accept rule that permits any packet matching $([1 \mathinner{.\,.} 3], [1 \mathinner{.\,.} 3], [1 \mathinner{.\,.} 3], [1 \mathinner{.\,.} 3])$ (ranges of source and destination IP addresses and ports). A second policy is similarly defined:

$$P_2 == (\{([2 \mathinner{.\,.} 4], [2 \mathinner{.\,.} 4], [2 \mathinner{.\,.} 4], [2 \mathinner{.\,.} 4])\}, \emptyset)$$

In composing these policies under a lowest-upper-bound style operation one cannot simply take a union of the sets of intervals as in some cases they may coalesce and in other cases they may partition into a number of disjoint intervals. The composition of the P_1 and P_2 policies is as follows:

$$\begin{aligned}
P_1 \sqcup P_2 = (\{&([1 \mathinner{.\,.} 4], [2 \mathinner{.\,.} 3], [2 \mathinner{.\,.} 3], [2 \mathinner{.\,.} 3]),\\
&([1 \mathinner{.\,.} 3], [1 \mathinner{.\,.} 1], [1 \mathinner{.\,.} 3], [1 \mathinner{.\,.} 3]),\\
&([2 \mathinner{.\,.} 4], [4 \mathinner{.\,.} 4], [2 \mathinner{.\,.} 4], [2 \mathinner{.\,.} 4]),\\
&([1 \mathinner{.\,.} 3], [2 \mathinner{.\,.} 3], [2 \mathinner{.\,.} 3], [1 \mathinner{.\,.} 1]),\\
&([1 \mathinner{.\,.} 3], [2 \mathinner{.\,.} 3], [1 \mathinner{.\,.} 1], [1 \mathinner{.\,.} 3]),\\
&([2 \mathinner{.\,.} 4], [2 \mathinner{.\,.} 3], [2 \mathinner{.\,.} 3], [4 \mathinner{.\,.} 4]),\\
&([2 \mathinner{.\,.} 4], [2 \mathinner{.\,.} 3], [4 \mathinner{.\,.} 4], [2 \mathinner{.\,.} 4])\}, \emptyset)
\end{aligned}$$

△

The policy algebra $\mathcal{FW}_1$, presented in Chapter 5, defines a firewall policy in terms of rules constraining *ranges* of IP addresses and ports.

Chapter 5

The $\mathcal{FW}_1$ Policy Model

The objective of this chapter is to develop a firewall policy algebra $\mathcal{FW}_1$, for constructing and reasoning over anomaly-free policies. Firewall rules comprise range-based filter condition attributes. The $\mathcal{FW}_1$ policy algebra is an extended version of the algebra in [101]. The chapter is organised as follows. Section 5.1 introduces the notion of adjacency, which is at the heart of reasoning about/composing firewall rules that involve range-based attributes, datatypes for firewall rule attributes and firewall rulesets are also defined. In Section 5.2 the attribute definitions given in Chapter 3 are extended for $\mathcal{FW}_1$ filter conditions. Section 5.3 defines the $\mathcal{FW}_1$ firewall policy algebra, and in Section 5.4 we show how $\mathcal{FW}_1$ can be used to reason about firewall policies in practice.

5.1 A Theory of Adjacency

Range-based filter condition attributes (for example, IPs/ports) have logical mappings to intervals of $\mathbb{N}$. For example, the port range that includes all ports from SSH upto and including HTTP can be written as the interval $[22 \mathinner{\ldotp\ldotp} 80]$.

Example 1 Consider, as part of a running example, a system that is capable of enforcing firewall rules whereby the filter condition attribute for the rules is a destination port range only. A rule that allowed all ports from SSH to HTTP would be:

Index	**Dst Prt**	**Action**
i	$[22 \mathinner{\ldotp\ldotp} 80]$	*allow*

where i is the index of the rule in the policy, $[22 \mathinner{\ldotp\ldotp} 80]$ is the required port range, and *allow* means that network traffic matching this pattern be permitted traversal of the firewall. Suppose we had a second rule, that specifies *allow* everything from Quote Of The Day (QOTD) up to and including FTP Control, then:

Index	**Dst Prt**	**Action**
j	$[17 \mathinner{\ldotp\ldotp} 21]$	*allow*

specifies that for the rule at index j; the required port range $[17 \mathinner{\ldotp\ldotp} 21]$ is allowed. Intuitively we can see that the port ranges for the rules at index i and index j are *adjacent*, and we may want to join rule i and rule j into a single firewall rule that looks like:

Index	**Dst Prt**	**Action**
k	$[17 \mathinner{\ldotp\ldotp} 80]$	*allow*

This notion of adjacency becomes more complex when we consider comparing/-composing firewall rules comprising two or more filter condition attributes. △

5.1.1 The Adjacency Specification

In this section we define the filter condition attribute relationships of *adjacency, disjointness* and *subsumption.* These can be applied to any ordered set, not just number intervals. These relationships are at the heart of adjacency, and ultimately the $\mathcal{FW}_1$ algebra.

Let $\mathcal{IV}[min, max]$ be the set of all intervals on the natural numbers, from *min* up to and including *max*. Intervals are defined by their corresponding sets.

$$\mathcal{IV}[min, max] == \{S : \mathbb{P}\,\mathbb{N} \mid \exists \bot, \top : S \bullet \forall x : S \bullet min \leq \bot \leq x \leq \top \leq max\}$$

Example 2 For ease of exposition and when no ambiguity arises, we may write an interval as a pair $[\bot \mathinner{\ldotp\ldotp} \top]$, rather than by the set it defines.

$$\mathcal{IV}[1, 3] = \{[1 \mathinner{\ldotp\ldotp} 1], [1 \mathinner{\ldotp\ldotp} 2], [1 \mathinner{\ldotp\ldotp} 3], [2 \mathinner{\ldotp\ldotp} 2], [2 \mathinner{\ldotp\ldotp} 3], [3 \mathinner{\ldotp\ldotp} 3]\}$$

△

Let *IPv*4 define the set of all possible IPv4 address ranges, and similarly, let

Port define the set of all possible network port ranges, whereby:

$$IPv4 == \mathcal{IV}[0, \mathsf{maxIP}]$$

$$Port == \mathcal{IV}[0, \mathsf{maxPrt}]$$

Adjacency. The relation $(\_ \wr_{\_} \_)$ defines adjacency over any ordered set. Adjacency is a general notion that is not limited to $\mathbb{N}$. We generalize adjacency to any attribute of generic type X, whereby for $a, b \in X$, if $a \wr_X b$, then a and b are adjacent in the set X. The property of reflexivity is required as any $a \in X$ should be adjacent to itself, that is; if for any a, that $a \wr_X a$ did not hold, then an inconsistency would exist. Symmetry follows from a similar requirement, where for $a, b \in X$, if a is adjacent to b in X, then b must also be adjacent to a in X. The following schema defines a generic adjacency relation that can be instantiated for adjacency over different datatypes.

$$\begin{array}{l}
[X] \\
\hline
\_ \wr_{\_} \_ : \mathbb{P}\, X \nrightarrow (X \leftrightarrow X) \\
\hline
\forall\, a, b : X \bullet \\
\qquad a \wr_X a \land \\
\qquad (a \wr_X b \Rightarrow b \wr_X a) \\
\hline
\end{array}$$

Given a set $S \in \mathbb{P}\, X$, then the transitive closure of the adjacency relation for elements in S is defined as follows.

$$\begin{array}{l}
[X] \\
\hline
\_ \overset{+}{\wr}_{\_} \_ : \mathbb{P}\, X \nrightarrow (X \leftrightarrow X) \\
\hline
\forall\, S : \mathbb{P}\, X \bullet \\
\qquad (\_ \overset{+}{\wr}_S \_) = (S \lhd (\_ \wr_X \_) \rhd S)^+ \\
\hline
\end{array}$$

Interval Adjacency. Two intervals on the set of natural numbers are adjacent if their union defines a single interval. For a given maximum value $\mathsf{max} \in \mathbb{N}$, we define:

$$\begin{array}{l}
\forall\, I, J : \mathcal{IV}[0, \mathsf{max}] \bullet \\
\qquad I \wr_{\mathcal{IV}[0,\mathsf{max}]} J \Leftrightarrow I \cup J \in \mathcal{IV}[0, \mathsf{max}]
\end{array}$$

Example 3 Interval $[1 \mathinner{\ldotp\ldotp} 2]$ is adjacent to interval $[3 \mathinner{\ldotp\ldotp} 3]$ since $[1 \mathinner{\ldotp\ldotp} 2] \cup [3 \mathinner{\ldotp\ldotp} 3] =$

$[1\,..\,3]$, thus $[1\,..\,2] \wr_{IPv4} [3\,..\,3]$. △

Number Adjacency. Two numbers are adjacent if they are the same or if they are different by a value of one. We define:

$$\begin{array}{l} \forall\, a, b : \mathbb{N} \bullet \\ \qquad a \wr_{\mathbb{N}} b \Leftrightarrow (a = b \lor a + 1 = b \lor b + 1 = a) \end{array}$$

Set Adjacency. For a generic type X, and sets S, $T \in \mathbb{P}\, X$, then S and T are adjacent, as $S \cup T \in \mathbb{P}\ X$. We define:

$$\begin{array}{l} \forall\, S, T : \mathbb{P}\, X \bullet \\ \qquad S \wr_{\mathbb{P}\, X}\ T \end{array}$$

Disjointness. The relation $(\_ \mid_{\_} \_)$ is used to define the notion of disjointness over any ordered set. Given $a, b \in X$, then $a \mid_X b$ denotes a and b are disjoint in X. The property of irreflexivity is required, as a cannot be disjoint from itself, that is; if for any a, that $\neg\ (a \mid_X a)$ did not hold, then an inconsistency would exist. Symmetry is also required for consistency, as if a and b are disjoint in X, then b and a must be disjoint in X also. The following schema specification defines a generic disjointness relation that can be instantiated for disjointness over different datatypes.

$$\begin{array}{|l} [X] \\ \hline \_ \mid_{\_} \_ : \mathbb{P}\ X \twoheadrightarrow (X \leftrightarrow X) \\ \hline \forall\, a, b : X \bullet \\ \qquad \neg\ (a \mid_X a) \land \\ \qquad (a \mid_X b \Rightarrow b \mid_X a) \\ \hline \end{array}$$

Interval Disjointness. Two intervals are disjoint if they don't intersect. For a given maximum value $\mathsf{max} \in \mathbb{N}$, we define:

$$\begin{array}{l} \forall\, I, J : \mathcal{IV}[0, \mathsf{max}] \bullet \\ \qquad I \mid_{\mathcal{IV}[0,\mathsf{max}]} J \Leftrightarrow I \cap J = \emptyset \end{array}$$

Example 4 Intervals $[1\,..\,2]$ and $[3\,..\,3]$ are disjoint, since $[1\,..\,2] \cap [3\,..\,3] = \emptyset$, thus $[1\,..\,2] \mid_{IPv4} [3\,..\,3]$. △

Number Disjointness. Two natural numbers are disjoint if they are different. We define:

$$\begin{aligned}&\forall\, a, b : \mathbb{N} \bullet\\ &\qquad a \mid_{\mathbb{N}} b \Leftrightarrow a \neq b\end{aligned}$$

Set Disjointness. Two sets are disjoint if they don't intersect. We define:

$$\begin{aligned}&\forall\, S, T : \mathbb{P}\, X \bullet\\ &\qquad S \mid_{\mathbb{P}\,X} T \Leftrightarrow S \cap T = \emptyset\end{aligned}$$

Subsumption. The relation ($\_ \overset{-}{\leftarrow} \_$) is used to define subsumption over any ordered set. For $a, b \in X$, if $a \overset{X}{\leftarrow} b$, then b covers a in X. Reflexivity is required, as any a must cover itself. Transitivity follows from a similar requirement, where for $a, b, c \in X$, if a covers b and b covers c, then a must cover c. Antisymmetry follows from the assumption of irredundant elements, where if a covers b and b covers a then one of them is unnecessary [42]. The properties of reflexivity, transitivity and antisymmetry define $\overset{X}{\leftarrow}$ as a non-strict partial order over X [18]. The following schema defines a generic subsumption relation that can be instantiated for subsumption over different datatypes.

$$\begin{aligned}&[X]\\ &\_ \overset{-}{\leftarrow} \_ : \mathbb{P}\ X \mathrel{+\!\!\!\!\!\!\to} (X \leftrightarrow X)\\ \hline &\forall\, a, b, c : X \bullet\\ &\qquad a \overset{X}{\leftarrow} a\ \wedge\\ &\qquad (a \overset{X}{\leftarrow} b \wedge b \overset{X}{\leftarrow} c \Rightarrow a \overset{X}{\leftarrow} c)\ \wedge\\ &\qquad (a \overset{X}{\leftarrow} b \wedge b \overset{X}{\leftarrow} a \Rightarrow a = b)\end{aligned}$$

Some subsumption orderings (for example, subset) may form a lattice with glb $\_ \cap_X \_$ and lub $\_ \cup_X \_$ operators for values in X.

Interval Subsumption. An interval I subsumes (covers) an interval J, if $J \subseteq I$. For a given maximum value $\mathsf{max} \in \mathbb{N}$, we define:

$$\begin{aligned}&\forall\, I, J : \mathcal{IV}[0, \mathsf{max}] \bullet\\ &\qquad J \overset{\mathcal{IV}[0,\mathsf{max}]}{\leftarrow} I \Leftrightarrow J \subseteq I\end{aligned}$$

Example 5 Interval $[1 \mathrel{..} 3]$ covers interval $[3 \mathrel{..} 3]$, since $[3 \mathrel{..} 3] \subseteq [1 \mathrel{..} 3]$, thus: $[3 \mathrel{..} 3] \overset{IPv4}{\leftarrow} [1 \mathrel{..} 3]$. △

Number Subsumption. For $a, b \in \mathbb{N}$ then b covers a if a equals b. We define:

$$\begin{array}{l} \forall\, a, b : \mathbb{N} \bullet \\ \qquad a \overset{\mathbb{N}}{\leftarrow} b \Leftrightarrow a = b \end{array}$$

The equality relation is both symmetric and antisymmetric, and defines both an equivalence relation and a non-strict partial order [118]. Thus, $a \overset{\mathbb{N}}{\leftarrow} b$ denotes that b subsumes/covers a.

Set Subsumption. The definition for set subsumption is as expected, we define:

$$\begin{array}{l} \forall\, S, T : \mathbb{P}\, X \bullet \\ \qquad S \overset{\mathbb{P}\, X}{\leftarrow} T \Leftrightarrow S \subseteq T \end{array}$$

For a generic type X, and $S \in \mathbb{P}\, X$, then the flattening function $\lceil S \rceil$ gives the cover-set for the elements of S, whereby the cover-set of S has no subsumption. We define:

$$\begin{array}{|l} [X] \\ \hline \lceil \_ \rceil : \mathbb{P}\, X \nrightarrow \mathbb{P}\, X \\ \hline \forall\, S : \mathbb{P}\, X \bullet \\ \qquad \lceil S \rceil = S \setminus \{a, a' : S \mid a \overset{X}{\leftarrow} a' \wedge a \neq a' \bullet a\} \\ \hline \end{array}$$

Example 6 Given the set of all intervals on the natural numbers from $1 \mathrel{..} 3$, then we have:

$$\begin{array}{l} \lceil \mathcal{IV}[1,3] \rceil = \{[1 \mathrel{..} 1], [1 \mathrel{..} 2], [1 \mathrel{..} 3], [2 \mathrel{..} 2], [2 \mathrel{..} 3], [3 \mathrel{..} 3]\} \setminus \\ \qquad \{[1 \mathrel{..} 1], [1 \mathrel{..} 2], [2 \mathrel{..} 2], [2 \mathrel{..} 3], [3 \mathrel{..} 3]\} \\ \qquad\qquad = \{[1 \mathrel{..} 3]\} \end{array}$$

△

We define a *difference* operator for $S, T \in \mathbb{P}\, X$, where $S \setminus_{\mathbb{P}\, X} T$ gives the relative compliment of T in S. That is, everything that is of type X that is covered in S, but not in T. We define this as:

$$[X]$$

$$\_ \setminus_{\_} \_ : \mathbb{P}(\mathbb{P}\, X) \nrightarrow \mathbb{P}\, X \times \mathbb{P}\, X \rightarrow \mathbb{P}\, X$$

$$\begin{array}{l} \forall\, S, T : \mathbb{P}\, X \bullet \\ \qquad S \setminus_{\mathbb{P}\, X} T = \lceil \{ a : S;\ c : X \mid c \overset{X}{\leftarrow} a \land (\forall\, b : T \bullet \neg\, (c \overset{X}{\leftarrow} b)) \bullet c \} \rceil \end{array}$$

Example 7 Given the cover-set for the set of all intervals on the natural numbers from $1 \mathrel{..} 3$, and the set $\{[1 \mathrel{..} 1], [3 \mathrel{..} 3]\}$, we have:

$$\lceil \mathcal{IV}[1, 3] \rceil \setminus_{IPv4} \{[1 \mathrel{..} 1], [3 \mathrel{..} 3]\} = \{[2 \mathrel{..} 2]\}$$

△

5.1.2 The Adjacency Datatype

For a generic type X, the Adjacency datatype $\alpha[X]$, is the set of all closed subsets of X partitioned by adjacency.

$$\alpha[X] == \{ S : \mathbb{P}\, X \mid (\forall\, a, b : S \mid a \neq b \bullet \neg\, (a \wr_X b)) \}$$

Example 8 We can use this to define all the ways that an interval can be partitioned into sets of non-adjacent intervals.

$$\begin{array}{l} \alpha[\mathcal{IV}[1, 3]] = \{\{[1 \mathrel{..} 1]\}, \{[1 \mathrel{..} 2]\}, \{[1 \mathrel{..} 3]\}, \{[2 \mathrel{..} 2]\}, \\ \qquad\qquad \{[2 \mathrel{..} 3]\}, \{[3 \mathrel{..} 3]\}, \{[1 \mathrel{..} 1], [3 \mathrel{..} 3]\}\} \end{array}$$

△

Let IP_{Spec} be the set of all closed subsets for the intervals of the IPv4 address range, partitioned by adjacency, and similarly, let Prt_{Spec} be the set of all closed subsets for the intervals of the network port range, partitioned by adjacency. We define:

$$IP_{Spec} == \alpha[IPv4]$$

$$Prt_{Spec} == \alpha[Port]$$

Adjacency Datatype Ordering. An ordering can be defined over Adjacency-sets of a generic type X as follows:

$$
\begin{array}{l}
[X] \\
\hline
\bot, \top : \alpha[X] \\
\mathbf{not} : \alpha[X] \rightarrow \alpha[X] \\
\_ \leq \_ : \alpha[X] \leftrightarrow \alpha[X] \\
\_ \otimes \_, \\
\_ \oplus \_ : \alpha[X] \times \alpha[X] \rightarrow \alpha[X] \\
\hline
\bot = \emptyset \land \top = \lceil X \rceil \\
\forall\, S, T : \alpha[X] \bullet \\
\qquad \mathbf{not}\, S = \top \setminus_{\alpha[X]} S \land \\
\qquad S \leq T \Leftrightarrow (\forall\, a : S \bullet \exists\, b : T \bullet a \overset{X}{\leftarrow} b) \land \\
\qquad S \otimes T = \lceil \bigcup \{ U : \alpha[X] \mid \forall\, c : U \bullet \exists\, a : S;\ b : T \bullet c \overset{X}{\leftarrow} a \land c \overset{X}{\leftarrow} b \} \rceil \land \\
\qquad S \oplus T = \bigcap \{ U : \alpha[X] \mid \forall\, c : U \bullet \exists\, a : S;\ b : T \bullet a \overset{X}{\leftarrow} c \lor b \overset{X}{\leftarrow} c \}
\end{array}
$$

Lemma 5.1.1 *The ordering relation $\leq$ is a non-strict partial order over $\alpha[X]$.*

Proof For $S, T \in \alpha[X]$, then $S \leq T$ means that T covers S, that is, every $a \in S$ is covered by some $b \in T$. The ordering relation $\leq$, is defined as a subsumption ordering/an antisymmetric preorder, where the properties of reflexivity, transitivity and antisymmetry hold for $\leq$ over $\alpha[X]$ as $(\_ \overset{X}{\leftarrow} \_)$ is a non-strict partial order for elements of type X. We have:

$$
\begin{array}{l}
\forall\, S, T, U : \alpha[X] \bullet \\
\qquad S \leq S \land \\
\qquad (S \leq T \land T \leq U \Rightarrow S \leq U) \land \\
\qquad (S \leq T \land T \leq S \Rightarrow S = T)
\end{array}
$$

The elements $\bot, \top \in \alpha[X]$ define the least and greatest bounds, respectively, on $\alpha[X]$, where $\bot$ is the unique minimal element that is covered by all elements, and $\top$ is the unique maximal element that covers all other elements. We have:

$$
\begin{array}{l}
\forall\, S : \alpha[X] \bullet \\
\qquad \bot \leq S \leq \top
\end{array}
$$

$\blacksquare$

Example 9 Given $\mathsf{ranges}_1, \mathsf{ranges}_2 \in \mathit{IP}_{\mathit{Spec}}$, where:

$$
\begin{array}{l}
\mathsf{ranges}_1 == \{[1 \mathinner{..} 3], [6 \mathinner{..} 6], [8 \mathinner{..} 8]\} \\
\mathsf{ranges}_2 == \{[2 \mathinner{..} 4], [7 \mathinner{..} 7], [10 \mathinner{..} 10]\}
\end{array}
$$

then Figure 5.1 depicts a partial Hasse diagram, for the composition of ranges_1 and ranges_2 under the relative ordering of $\leq$ over IP_{Spec}. $\triangle$

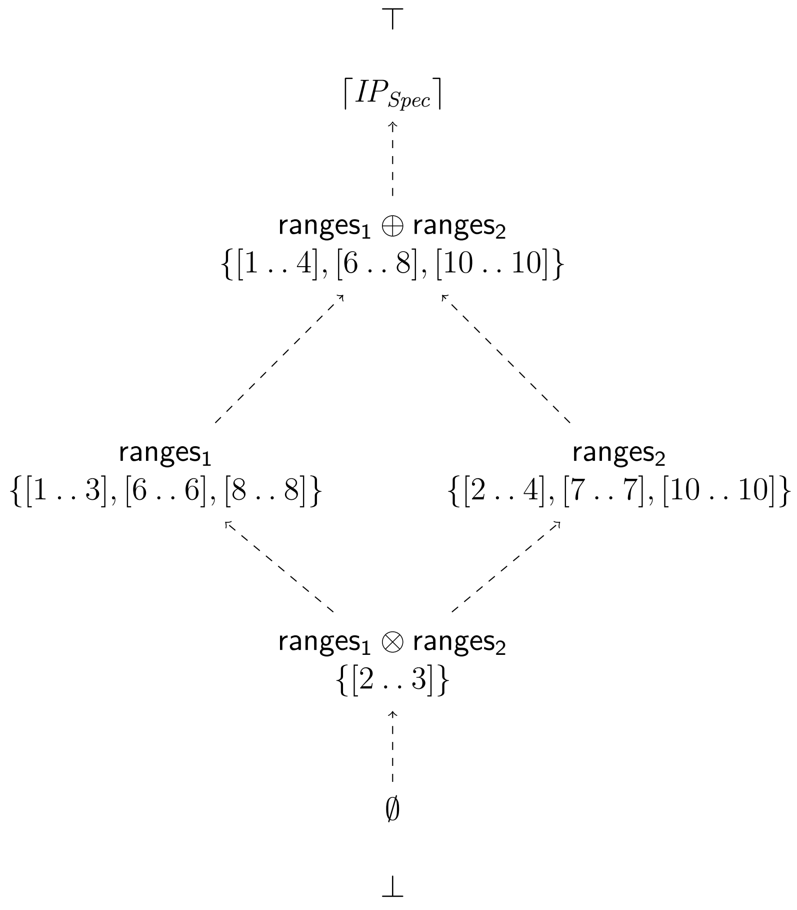


Figure 5.1: IP_{Spec} ordering fragment

Adjacency Datatype Union. The *join* of $S, T \in \alpha[X]$ is defined using subsumption, as the generalized intersection of all Adjacency-sets, where each element of $(S \oplus T)$ covers an element in *either* S or T. Intuitively, this means that the values of the join are exactly a union of the elements from both S and T.

Lemma 5.1.2 *The operator $\oplus$ is a least upper bound operator on $\alpha[X]$.*

Proof The generalized intersection in the join operation for some $S, T \in \alpha[X]$ defines the smallest collection of $x \in X$ that cover all of the elements from both S and T by subsumption. If we take some $U \in \alpha[X]$, such that $U \leq (S \oplus T)$ and $S \leq U \wedge T \leq U$, then $(S \oplus T) = U$. Thus, Adjacency join provides a lowest upper bound operator. Since $\oplus$ provides a lub operator we have $S \leq (S \oplus T)$ and $T \leq (S \oplus T)$. ▮

Adjacency Datatype Intersection. Under this ordering, the *meet*, or intersection $S \otimes T$ of $S, T \in \alpha[X]$, is defined using subsumption, as the cover-set for the generalized union of all Adjacency-sets, where each element of $(S \otimes T)$ is covered by an element in *both* S and T. Intuitively, this means that the values of the meet are all non-empty intersections of each value in S with each value in T.

Corollary 5.1.3 *The operator $\otimes$ is a greatest lower bound operator on $\alpha[X]$.*

Proof It follows from Lemma 5.1.2 by an analogous argument that Corollary 5.1.3 holds. ▮

Since $\otimes$ provides the glb operator, then for $S, T \in \alpha[X]$ we have $S \otimes T$ is covered by both S and T, that is $(S \otimes T) \leq S$ and $(S \otimes T) \leq T$.

Adjacency Datatype Negation. Given $S \in \alpha[X]$, then $\mathbf{not}\, S$ defines a complement operator in $\alpha[X]$, where $\mathbf{not}\, S$ is the cover-set for all elements of type X that are not covered by some member of S. We have:

$$\begin{aligned} &\forall\, S : \alpha[X] \bullet \\ &\qquad (S \oplus \mathbf{not}\, S) = \top \wedge (S \otimes \mathbf{not}\, S) = \bot \end{aligned}$$

Theorem 5.1.4 *The poset $(\alpha[X], \leq)$ forms a lattice with complement operator* **not***.*

Proof This follows from the definition of $\leq$ as a subsumption ordering/an antisymmetric preorder, the properties of **not**, the definition of the meet as the cover-set for the generalized union of all Adjacency-sets, where for $S, T \in \alpha[X]$, each element of $(S \otimes T)$ is covered by an element in *both* S and T, and the definition of the join as the generalized intersection of all Adjacency-sets, where each element of $(S \oplus T)$ covers an element in *either* S or T. ▮

5.1.3 The Duplet Datatype

The notion of adjacency becomes more complex when we consider comparing/-composing firewall rules comprising two or more filter condition attributes. When joining adjacent firewall rules, in some cases the rules may coalesce and in other cases they may partition into a number of disjoint rules.

Example 10 Recall from Example 1, the firewall system that supports only destination port range filter conditions. Suppose we want to extend the expressiveness of the policy rules for this system to include a definition for destination

$\mathsf{IP\ range}$. Consider, two policy requirements; whereby network traffic is to be allowed to the IP range $[1\,..\,3]$ on ports $[1\,..\,3]$, and to the IP range $[2\,..\,4]$ on ports $[2\,..\,4]$. Then modelling this using adjacency-free IP/port range pairs, we have $\mathsf{p}_1, \mathsf{p}_2 \in (IP_{Spec} \times Prt_{Spec})$, whereby:

$$\begin{aligned}\mathsf{p}_1 &== (\{[1\,..\,3]\}, \{[1\,..\,3]\})\\ \mathsf{p}_2 &== (\{[2\,..\,4]\}, \{[2\,..\,4]\})\end{aligned}$$

If we consider the attributes separately, we observe that the IP range in p_1 is adjacent to the IP range in p_2, and the port ranges in p_1 and p_2 are also adjacent. However, in composing p_1 and p_2 under a lowest-upper-bound style operation one cannot simply take a union of the sets of intervals to be the IP/port range pair: $(\{[1\,..\,4]\}, \{[1\,..\,4]\})$, as this results in an overly permissive policy, given that network traffic is permitted to IP $\mathsf{1}$ on port $\mathsf{4}$, and to IP $\mathsf{4}$ on port $\mathsf{1}$ as a result of composition. Conversely, this would result in an overly restrictive policy if we were composing *deny* rules.

If we consider how the join of p_1 and p_2 may be defined, whereby the desired result is the smallest number of non-adjacent rules that cover both p_1 and p_2, then we can apply an adjacency-precedence to the IP ranges in p_1 and p_2, and observe that the port ranges in p_1 and p_2 are not disjoint. We refer to this as the 1^{st} *attribute major ordering*, and the cover for p_1 and p_2 is given as:

$$\begin{aligned}\mathsf{p}_1 \uplus^{1^{st}}_{Mjr} \mathsf{p}_2 = \{&(\{[1\,..\,4]\}, \{[2\,..\,3]\}),\\ &(\{[1\,..\,3]\}, \{[1\,..\,1]\}),\\ &(\{[2\,..\,4]\}, \{[4\,..\,4]\})\}\end{aligned}$$

In this case, the result is a set of disjoint rules that exactly cover the IP/port-range pair constraints from p_1 and p_2. The resulting operations are encoded in the matrix in Table 5.1, whereby the label R signifies the new rule, and the label A means filter condition attribute.

A R	$\boxed{1}$	$\boxed{2}$
$\boxed{1}$	$\boxed{\{[1\,..\,3]\} \cup_{IP_{Spec}} \{[2\,..\,4]\}}$	$\boxed{\{[1\,..\,3]\} \cap_{Prt_{Spec}} \{[2\,..\,4]\}}$
$\boxed{2}$	$\boxed{\{[1\,..\,3]\}}$	$\boxed{\{[1\,..\,3]\} \setminus_{Prt_{Spec}} \{[2\,..\,4]\}}$
$\boxed{3}$	$\boxed{\{[2\,..\,4]\}}$	$\boxed{\{[2\,..\,4]\} \setminus_{Prt_{Spec}} \{[1\,..\,3]\}}$

Table 5.1: A two-attribute rule join, 1^{st} attribute major ordering

We note, however, that instead, the adjacency-precedence may be applied to the second attribute, where in this case we observe that the port ranges in p_1 and p_2 are adjacent, and the IP ranges in p_1 and p_2 are not disjoint. We refer to this as a 2^{nd} *attribute major ordering*, and would therefore expect the set of disjoint rules that exactly cover the IP/port-range pair constraints from p_1 and p_2 to be:

$$\begin{aligned}\mathsf{p}_1 \uplus^{2^{nd}}_{mjr} \mathsf{p}_2 = \{&(\{[2 \ldotp\ldotp 3]\}, \{[1 \ldotp\ldotp 4]\}),\\ &(\{[1 \ldotp\ldotp 1]\}, \{[1 \ldotp\ldotp 3]\}),\\ &(\{[4 \ldotp\ldotp 4]\}, \{[2 \ldotp\ldotp 4]\})\}\end{aligned}$$

The operations for the 2^{nd} attribute major ordering the are encoded in the matrix in Table 5.2.

A R	$\boxed{1}$	$\boxed{2}$
$\boxed{1}$	$\boxed{\{[1 \ldotp\ldotp 3]\} \cap_{IP_{Spec}} \{[2 \ldotp\ldotp 4]\}}$	$\boxed{\{[1 \ldotp\ldotp 3]\} \cup_{Prt_{Spec}} \{[2 \ldotp\ldotp 4]\}}$
$\boxed{2}$	$\boxed{\{[1 \ldotp\ldotp 3]\} \setminus_{IP_{Spec}} \{[2 \ldotp\ldotp 4]\}}$	$\boxed{\{[1 \ldotp\ldotp 3]\}}$
$\boxed{3}$	$\boxed{\{[2 \ldotp\ldotp 4]\} \setminus_{IP_{Spec}} \{[1 \ldotp\ldotp 3]\}}$	$\boxed{\{[2 \ldotp\ldotp 4]\}}$

Table 5.2: A two-attribute rule join, 2^{nd} attribute major ordering

For the remainder of this dissertation, we consider firewall rule join in terms of the 1^{st} attribute major ordering. However, we also consider the join of rules where there is an adjacency in other than the first attribute, we refer to this type of adjacency as *forward* adjacency. △

Duplets. A duplet is an ordered pair, where the set of all duplets for generic types X, Y, is defined as $\delta[X, Y]$, whereby:

$$\delta[X, Y] == X \times Y$$

Example 11 For $\mathcal{IV}[1, 1]$ and $\mathcal{IV}[1, 2]$, we have:

$$\delta[\mathcal{IV}[1,1], \mathcal{IV}[1,2]] = \{([1 \ldotp\ldotp 1], [1 \ldotp\ldotp 1]), ([1 \ldotp\ldotp 1], [1 \ldotp\ldotp 2]), ([1 \ldotp\ldotp 1], [2 \ldotp\ldotp 2])\}$$

and $\delta[IP_{Spec}, Prt_{Spec}]$ gives the set of all duplets for adjacency-free IP/port-range pairs. △

Lemma 5.1.5 *If the ordering over X is a lattice and the ordering over Y is a lattice, then the ordering over $\delta[X, Y]$ is also a lattice.*

Proof Given the definition of $\delta[X, Y]$ as the Cartesian product of X and Y, then if the ordering over X is a lattice and the ordering over Y is a lattice; it follows that $\delta[X, Y]$ forms a lattice under the product order of X and Y. ∎

Forward Adjacency. A pair of duplets are forward adjacent to each other if the attributes in the first coordinate are equal and the attributes in the second coordinate are adjacent. For $(a_1, b_1), (a_2, b_2) \in \delta[X, Y]$, we define forward adjacency, whereby:

$$(a_1 = a_2 \wedge b_1 \wr_Y b_2)$$

Example 12 Given duplets $(\{[1 \mathinner{..} 3]\}, \{[2 \mathinner{..} 3]\}), (\{[1 \mathinner{..} 3]\}, \{[1 \mathinner{..} 1]\}) \in \delta[IP_{Spec}, Prt_{Spec}]$, then these duplets are forward adjacent, as:

$$(\{[1 \mathinner{..} 3]\} = \{[1 \mathinner{..} 3]\}) \wedge (\{[2 \mathinner{..} 3]\} \wr_{Prt_{Spec}} \{[1 \mathinner{..} 1]\})$$

△

Duplet Adjacency. A pair of duplets $a, b \in \delta[X, Y]$ are adjacent, if the attributes in the first coordinate are adjacent, and the attributes in the second coordinate are not disjoint, or a and b are forward adjacent. For $(a_1, b_1), (a_2, b_2) \in \delta[X, Y]$, we define:

$$(a_1, b_1) \wr_{\delta[X,Y]} (a_2, b_2) \Leftrightarrow ((a_1 \wr_X a_2 \wedge \neg\ (b_1 \mid_Y b_2)) \vee (a_1 = a_2 \wedge b_1 \wr_Y b_2))$$

Example 13 We have $\mathsf{p}_1 \wr_{\delta[IP_{Spec}, Prt_{Spec}]} \mathsf{p}_2$, since the IP ranges are adjacent and the port ranges are not disjoint.

$$\{[1 \mathinner{..} 3]\} \wr_{IP_{Spec}} \{[2 \mathinner{..} 4]\} \wedge \neg\ (\{[1 \mathinner{..} 3]\} \mid_{Prt_{Spec}} \{[2 \mathinner{..} 4]\})$$

△

Duplet Disjointness. A pair of duplets are disjoint if the attributes in the first coordinate are disjoint, and/or the attributes in the second coordinate are disjoint. For $(a_1, b_1), (a_2, b_2) \in \delta[X, Y]$, we define:

$$(a_1, b_1) \mid_{\delta[X,Y]} (a_2, b_2) \Leftrightarrow (a_1 \mid_X a_2 \vee b_1 \mid_Y b_2)$$

Example 14 We have $\neg\ (\mathsf{p}_1 \mid_{\delta[IP_{Spec}, Prt_{Spec}]} \mathsf{p}_2)$, since:

$$\neg\ (\{[1 \mathinner{..} 3]\} \mid_{IP_{Spec}} \{[2 \mathinner{..} 4]\} \wedge \{[1 \mathinner{..} 3]\} \mid_{Prt_{Spec}} \{[2 \mathinner{..} 4]\})$$

△

Duplet Intersection. The definition for duplet intersection is defined as the intersection of the attributes in each coordinate under their respective orderings. For $(a_1, b_1), (a_2, b_2) \in \delta[X, Y]$, we define:

$$(a_1, b_1) \cap_{\delta[X,Y]} (a_2, b_2) = ((a_1 \cap_X a_2), (b_1 \cap_Y b_2))$$

Example 15 For p_1 and p_2, we have:

$$\mathsf{p}_1 \cap_{\delta[IP_{Spec}, Prt_{Spec}]} \mathsf{p}_2 = (\{[2 \mathinner{..} 3]\}, \{[2 \mathinner{..} 3]\})$$

△

Duplet Merge. The definition for duplet merge is defined as the union of the attributes in each coordinate under their respective orderings. For $(a_1, b_1), (a_2, b_2) \in \delta[X, Y]$, we define:

$$(a_1, b_1) \cup_{\delta[X,Y]} (a_2, b_2) = ((a_1 \cup_X a_2), (b_1 \cup_Y b_2))$$

Example 16 For p_1 and p_2, we have:

$$\mathsf{p}_1 \cup_{\delta[IP_{Spec}, Prt_{Spec}]} \mathsf{p}_2 = (\{[1 \mathinner{..} 4]\}, \{[1 \mathinner{..} 4]\})$$

△

Duplet Subsumption. A duplet (a_1, b_1) covers a duplet (a_2, b_2) in $\delta[X, Y]$, if a_1 covers a_2 in X, and b_1 covers b_2 in Y. Thus, we define duplet subsumption as:

$$(a_2, b_2) \overset{\delta[X,Y]}{\leftarrow} (a_1, b_1) \Leftrightarrow (a_2 \overset{X}{\leftarrow} a_1 \wedge (b_2 \overset{Y}{\leftarrow} b_1)$$

Example 17 For p_1 and p_2, we have:

$$\begin{array}{l} (\{[2 \mathinner{..} 3]\}, \{[2 \mathinner{..} 3]\}) \overset{\delta[IP_{Spec}, Prt_{Spec}]}{\leftarrow} \mathsf{p}_1 \wedge \\ (\{[2 \mathinner{..} 3]\}, \{[2 \mathinner{..} 3]\}) \overset{\delta[IP_{Spec}, Prt_{Spec}]}{\leftarrow} \mathsf{p}_2 \end{array}$$

△

Precedence Subsumption. A precedence subsumption is defined for duplets, whereby we explicitly define subsumption orderings separately in each coordinate.

The relation $(\_ \overset{-}{\rightarrow} \_)$ defines a general format for precedence subsumption over any ordered set. For $a, b \in X$, if $a \overset{X}{\rightarrow} b$, then a covers b by precedence in X. The properties of reflexivity, transitivity and antisymmetry define $\overset{X}{\rightarrow}$ as a non-strict partial order over X [18]. The following schema defines a generic precedence subsumption relation that can be instantiated for precedence subsumption over different datatypes.

$$
\begin{array}{|l}
[X] \\
\hline
\_ \overset{-}{\rightarrow} \_ : \mathbb{P}\, X \rightarrow\!\!\!\!\!\!\!\!\;\mapsto (X \leftrightarrow X) \\
\hline
\forall\, a, b, c : X \bullet \\
\qquad a \overset{X}{\rightarrow} a \land \\
\qquad (a \overset{X}{\rightarrow} b \land b \overset{X}{\rightarrow} c \Rightarrow a \overset{X}{\rightarrow} c) \land \\
\qquad (a \overset{X}{\rightarrow} b \land b \overset{X}{\rightarrow} a \Rightarrow a = b) \\
\hline
\end{array}
$$

A duplet (a_1, b_1) covers a duplet (a_2, b_2) by precedence in $\delta[X, Y]$, if a_1 covers a_2 in X, and b_2 covers b_1 in Y. Thus, we define precedence subsumption as:

$$(a_1, b_1) \overset{\delta[X,Y]}{\rightarrow} (a_2, b_2) \Leftrightarrow (a_2 \overset{X}{\leftarrow} a_1 \land (b_1 \overset{Y}{\leftarrow} b_2)$$

Example 18 For $\mathsf{p}_1, \mathsf{p}_2$, and duplets $(\{[1 \mathinner{.\,.} 4]\}, \{[2 \mathinner{.\,.} 3]\}), (\{[1 \mathinner{.\,.} 3]\}, \{[1 \mathinner{.\,.} 1]\}) \in \delta[\mathit{IP}_{\mathit{Spec}}, \mathit{Prt}_{\mathit{Spec}}]$, then we have:

$$
\begin{array}{l}
(\{[1 \mathinner{.\,.} 3]\}, \{[1 \mathinner{.\,.} 1]\}) \overset{\delta[\mathit{IP}_{\mathit{Spec}}, \mathit{Prt}_{\mathit{Spec}}]}{\rightarrow} \mathsf{p}_1 \land \\
(\{[1 \mathinner{.\,.} 4]\}, \{[2 \mathinner{.\,.} 3]\}) \overset{\delta[\mathit{IP}_{\mathit{Spec}}, \mathit{Prt}_{\mathit{Spec}}]}{\rightarrow} \mathsf{p}_1 \land \\
(\{[1 \mathinner{.\,.} 4]\}, \{[2 \mathinner{.\,.} 3]\}) \overset{\delta[\mathit{IP}_{\mathit{Spec}}, \mathit{Prt}_{\mathit{Spec}}]}{\rightarrow} \mathsf{p}_2
\end{array}
$$

△

Precedence Cover. For a duplet $a \in \delta[X, Y]$ and a set of duplets $S \in \mathbb{P}\, \delta[X, Y]$, then S covers a if the duplet merge of all elements in S that each cover a by precedence subsumption, cover a by duplet subsumption. We define:

$$\begin{array}{l}
[X, Y] \\
\hline
\_ \overset{-}{\leftrightsquigarrow} \_ : \mathbb{P}\ \delta[X, Y] \twoheadrightarrow\!\!\!\!\!\!\!\;\;\; (\delta[X, Y] \leftrightarrow \mathbb{P}\ \delta[X, Y]) \\
\hline
\forall\, a : \delta[X, Y];\ S : \mathbb{P}\ \delta[X, Y] \bullet \\
\qquad a \overset{\delta[X, Y]}{\leftrightsquigarrow} S \Leftrightarrow (\exists\, b, b' : S \mid (b \overset{\delta[X, Y]}{\rightarrow} a \land b' \overset{\delta[X, Y]}{\rightarrow} a) \land \\
\qquad\qquad a \overset{\delta[X, Y]}{\leftarrow} (b \cup_{\delta[X, Y]} b'))
\end{array}$$

Example 19 For p_1, and duplets $(\{[1\mathinner{..}3]\}, \{[2\mathinner{..}3]\}), (\{[1\mathinner{..}3]\}, \{[1\mathinner{..}1]\}) \in \delta[IP_{Spec}, Prt_{Spec}]$, then we have:

$$\mathsf{p}_1 \overset{\delta[IP_{Spec}, Prt_{Spec}]}{\leftrightsquigarrow} \{(\{[1\mathinner{..}3]\}, \{[2\mathinner{..}3]\}), (\{[1\mathinner{..}3]\}, \{[1\mathinner{..}1]\})\}$$

as:

$$(\{[1\mathinner{..}3]\}, \{[1\mathinner{..}3]\}) \overset{\delta[IP_{Spec}, Prt_{Spec}]}{\leftrightsquigarrow} \{(\{[1\mathinner{..}3]\}, \{[1\mathinner{..}3]\})\}$$

△

Intersecting Elements. For $a \in \delta[X, Y]$ and $S \in \mathbb{P}\,\delta[X, Y]$, then $a\lfloor S \rfloor$ is the set of all non-empty intersections of a with each value $b \in S$. We define:

$$\begin{array}{l}
[X, Y] \\
\hline
\_\lfloor\_\rfloor : \delta[X, Y] \times \mathbb{P}\ \delta[X, Y] \rightarrow \mathbb{P}\ \delta[X, Y] \\
\hline
\forall\, a : \delta[X, Y];\ S : \mathbb{P}\ \delta[X, Y] \bullet \\
\qquad a\lfloor S \rfloor = \{b : S \mid \neg\ (a \mid_{\delta[X, Y]} b) \bullet (a \cap_{\delta[X, Y]} b)\}
\end{array}$$

Example 20 For p_1, and duplets $(\{[1\mathinner{..}4]\}, \{[2\mathinner{..}3]\}), (\{[1\mathinner{..}3]\}, \{[1\mathinner{..}1]\}), (\{[2\mathinner{..}4]\}, \{[4\mathinner{..}4]\}) \in \delta[IP_{Spec}, Prt_{Spec}]$, then:

$$\begin{array}{l}
\mathsf{p}_1\lfloor\{(\{[1\mathinner{..}4]\}, \{[2\mathinner{..}3]\}), (\{[1\mathinner{..}3]\}, \{[1\mathinner{..}1]\}), (\{[2\mathinner{..}4]\}, \{[4\mathinner{..}4]\}\}\rfloor = \\
\qquad \{(\{[1\mathinner{..}3]\}, \{[2\mathinner{..}3]\}), (\{[1\mathinner{..}3]\}, \{[1\mathinner{..}1]\})\}
\end{array}$$

△

5.1.4 Duplet Adjacency Ordering

In this section, an ordering is defined for Adjacency-sets of duplets.

Duplet Adjacency Difference. Given $S, T \in \alpha[\delta[X, Y]]$, then $S \setminus_{\alpha[\delta[X,Y]]} T$ is the cover-set for the set of all duplets covered in S by a duplet, or by a collection of duplets, and not covered in T. Thus, we define the difference of Adjacency-sets of duplets as:

$$S \setminus_{\alpha[\delta[X,Y]]} T = \lceil \{ c : \delta[X, Y] \mid c \overset{\delta[X,Y]}{\leftrightsquigarrow} c\lfloor S \rfloor \land \neg\, (c \overset{\delta[X,Y]}{\leftrightsquigarrow} c\lfloor T \rfloor) \} \rceil$$

Example 21 Given $\mathsf{Pol}_1^{\mathsf{p}}, \mathsf{Pol}_2^{\mathsf{p}} \in \alpha[\delta[IP_{Spec}, Prt_{Spec}]]$, where:

$$\begin{aligned}
&\mathsf{Pol}_1^{\mathsf{p}} == \{(\{[1 \mathinner{\ldotp\ldotp} 3]\}, \{[1 \mathinner{\ldotp\ldotp} 3]\})\}\\
&\mathsf{Pol}_2^{\mathsf{p}} == \{(\{[2 \mathinner{\ldotp\ldotp} 4]\}, \{[2 \mathinner{\ldotp\ldotp} 4]\})\}
\end{aligned}$$

then:

$$\mathsf{Pol}_1^{\mathsf{p}} \setminus_{\alpha[\delta[IP_{Spec}, Prt_{Spec}]]} \mathsf{Pol}_2^{\mathsf{p}} = \{(\{[1 \mathinner{\ldotp\ldotp} 3]\}, \{[1 \mathinner{\ldotp\ldotp} 1]\}), (\{[1 \mathinner{\ldotp\ldotp} 1]\}, \{[2 \mathinner{\ldotp\ldotp} 3]\})\}$$

△

The implementation definition for duplet difference is given in Chapter 6 Section 6.1.1.

Duplet Adjacency Ordering. An ordering can be defined over Adjacency-sets of duplets as follows:

$$\begin{array}{|l}
[X, Y] \\
\hline
\bot, \top : \alpha[\delta[X, Y]] \\
\mathbf{not} : \alpha[\delta[X, Y]] \to \alpha[\delta[X, Y]] \\
\_ \leq \_ : \alpha[\delta[X, Y]] \leftrightarrow \alpha[\delta[X, Y]] \\
\_ \otimes \_, \\
\_ \oplus \_ : \alpha[\delta[X, Y]] \times \alpha[\delta[X, Y]] \to \alpha[\delta[X, Y]] \\
\hline
\bot = \emptyset \land \top = \lceil \delta[X, Y] \rceil \\
\forall\, S, T : \alpha[\delta[X, Y]] \bullet \\
\qquad \mathbf{not}\, S = \top \setminus_{\alpha[\delta[X,Y]]} S \land \\
\qquad S \leq T \Leftrightarrow (\forall\, a : S \bullet a \overset{\delta[X,Y]}{\leftrightsquigarrow} a\lfloor T \rfloor) \land \\
\qquad S \oplus T = \lceil \{ a, b : \bigcap \{ U : \mathbb{P}(\delta[X, Y]) \mid (\forall\, c : U \bullet \exists\, a : S;\; b : T \bullet \\
\qquad\qquad\qquad c \overset{\delta[X,Y]}{\rightarrow} a \lor c \overset{\delta[X,Y]}{\rightarrow} b)\} \mid a \wr^{+}_{\delta[X,Y]} b \bullet a \cup_{\delta[X,Y]} b \} \rceil \land \\
\qquad S \otimes T = \lceil \bigcup \{ U : \alpha[\delta[X, Y]] \mid \forall\, c : U \bullet c \overset{\delta[X,Y]}{\leftrightsquigarrow} c\lfloor S \rfloor \land c \overset{\delta[X,Y]}{\leftrightsquigarrow} c\lfloor T \rfloor \} \rceil \\
\hline
\end{array}$$

Figure 5.2 depicts a partial Hasse diagram, for the composition of $\mathsf{Pol}_1^\mathsf{p}$ and $\mathsf{Pol}_2^\mathsf{p}$ under the relative ordering of $\leq$ over $\alpha[\delta[IP_{Spec}, Prt_{Spec}]]$.

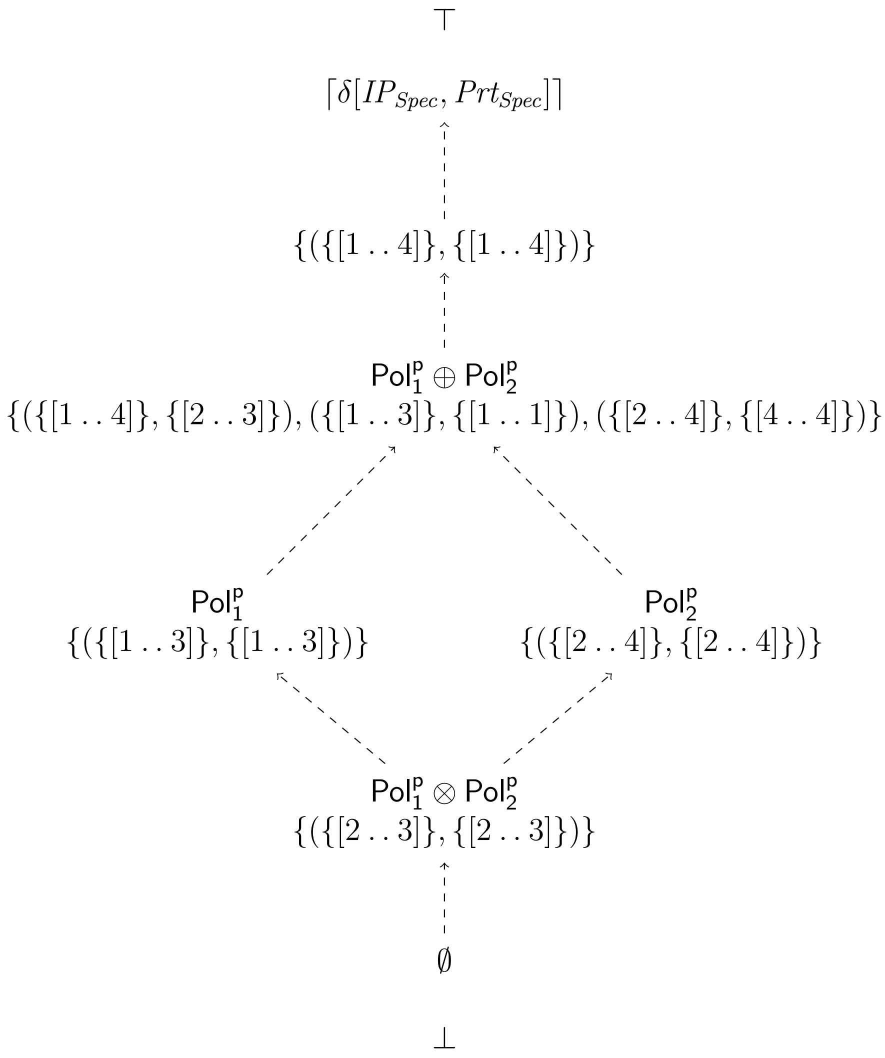

$\top$
$\lceil \delta[IP_{Spec}, Prt_{Spec}] \rceil$
$\{(\{[1\mathinner{..}4]\}, \{[1\mathinner{..}4]\})\}$
$\mathsf{Pol}_1^\mathsf{p} \oplus \mathsf{Pol}_2^\mathsf{p}$
$\{(\{[1\mathinner{..}4]\}, \{[2\mathinner{..}3]\}), (\{[1\mathinner{..}3]\}, \{[1\mathinner{..}1]\}), (\{[2\mathinner{..}4]\}, \{[4\mathinner{..}4]\})\}$
$\mathsf{Pol}_1^\mathsf{p}$
$\{(\{[1\mathinner{..}3]\}, \{[1\mathinner{..}3]\})\}$
$\mathsf{Pol}_2^\mathsf{p}$
$\{(\{[2\mathinner{..}4]\}, \{[2\mathinner{..}4]\})\}$
$\mathsf{Pol}_1^\mathsf{p} \otimes \mathsf{Pol}_2^\mathsf{p}$
$\{(\{[2\mathinner{..}3]\}, \{[2\mathinner{..}3]\})\}$
$\emptyset$
$\perp$

Figure 5.2: Duplet ordering fragment

Lemma 5.1.6 *The ordering relation $\leq$ is a non-strict partial order over $\alpha[\delta[X, Y]]$.*

Proof For $S, T \in \alpha[\delta[X, Y]]$, then $S \leq T$ means that T covers S, that is, every $a \in S$ is covered under duplet subsumption, either by a duplet, or by a collection of duplets in T. The ordering relation $\leq$, is defined as an antisymmetric preorder, where the properties of reflexivity, transitivity and antisymmetry hold for $\leq$ over $\alpha[\delta[X, Y]]$.

Reflexivity. For $S \in \alpha[\delta[X, Y]]$, we have for $a \in S$, then a covers itself under duplet subsumption. Since S is adjacency-free, then every duplet in S covers only itself under a duplet subsumption in S. Since duplet subsumption is reflexive, then $\leq$ is reflexive; that is:

$$\begin{array}{l} \forall\, S : \alpha[\delta[X, Y]] \bullet \\ \qquad (S, S) \in (\_ \leq \_) \end{array}$$

Transitivity. For $S, T \in \alpha[\delta[X, Y]]$, then $S \leq T$ if all $a \in S$ are covered in T under duplet subsumption. Then if we take some $U \in \alpha[\delta[X, Y]]$, such that $T \leq U$, then all $b \in T$ are covered in U under duplet subsumption. Therefore, all $a \in S$ are covered in U under duplet subsumption as duplet subsumption is transitive. Then $\leq$ is transitive; that is:

$$\begin{array}{l} \forall\, S, T, U : \alpha[\delta[X, Y]] \bullet \\ \qquad (S, T) \in (\_ \leq \_) \wedge (T, U) \in (\_ \leq \_) \Rightarrow (S, U) \in (\_ \leq \_) \end{array}$$

Antisymmetry. For $S, T \in \alpha[\delta[X, Y]]$, then $S \leq T$ if all $a \in S$ are covered in T under duplet subsumption, and if $T \leq S$ then all $b \in T$ are covered in S under duplet subsumption. Therefore, from the definition of duplet subsumption, if $S \leq T$ and $T \leq S$ then $T = S$. Then $\leq$ is antisymmetric; that is:

$$\begin{array}{l} \forall\, S, T : \alpha[\delta[X, Y]] \bullet \\ \qquad (S, T) \in (\_ \leq \_) \wedge (T, S) \in (\_ \leq \_) \Rightarrow S = T \end{array}$$

The elements $\bot, \top \in \alpha[\delta[X, Y]]$ define the least and greatest bounds, respectively, on $\alpha[\delta[X, Y]]$, where $\bot$ is the unique minimal element that is covered by all elements, and $\top$ is the unique maximal element that covers all other elements. We have:

$$\begin{array}{l} \forall\, S : \alpha[\delta[X, Y]] \bullet \\ \qquad \bot \leq S \leq \top \end{array}$$

Then we have:

$$\begin{array}{l} \forall\, S, T, U : \alpha[\delta[X, Y]] \bullet \\ \qquad S \leq S \wedge \\ \qquad (S \leq T \wedge T \leq U \Rightarrow S \leq U) \wedge \\ \qquad (S \leq T \wedge T \leq S \Rightarrow S = T) \end{array}$$

Thus, $\leq$ is a non-strict partial order over $\alpha[\delta[X, Y]]$. ▮

Adjacency Duplet Union. The join of $S, T \in \alpha[\delta[X, Y]]$ is defined using subsumption, as the cover-set for the duplet merge of the transitive closure of adjacent duplets, from the generalized intersection of all sets of sets of duplets, whereby each element of the generalized intersection covers an element in *either* S or T by duplet precedence subsumption. The generalized intersection defines the smallest collection of duplets that cover all of the duplets from both S and T by precedence subsumption. Given that all duplets in this set are now disjoint, the cover-set for the duplet merge of the transitive closure of adjacent duplets merges any forward-adjacent duplets from S and T. If we take some $U \in \alpha[\delta[X, Y]]$, such that $U \leq (S \oplus T)$ and $S \leq U \wedge T \leq U$, then $(S \oplus T) = U$. Thus, Adjacency join provides a lowest upper bound operator, we have:

$$\begin{array}{l} \forall\, S, T, U : \alpha[\delta[X, Y]] \bullet \\ \qquad S \leq (S \oplus T) \wedge T \leq (S \oplus T) \wedge \\ \qquad (S \leq U \wedge T \leq U \Rightarrow (S \oplus T) \leq U) \end{array}$$

An efficient implementation definition for duplet join is given in Chapter 6 Section 6.1.1.

Adjacency Duplet Intersection. Under this ordering, the meet $(S \otimes T)$ of $S, T \in \alpha[\delta[X, Y]]$ is defined using subsumption, as the cover-set for the generalized union of all Adjacency-sets, where each element of $(S \otimes T)$ is covered by a duplet, or by a collection of duplets in *both* S and T. The meet is defined as the largest set of adjacency-free duplets that is covered by both S and T. Thus, $\otimes$ provides the glb operator, then for $S, T \in \alpha[\delta[X, Y]]$ we have $S \otimes T$ is covered by both S and T, that is $(S \otimes T) \leq S$ and $(S \otimes T) \leq T$.

Adjacency Duplet Negation. Given $S \in \alpha[\delta[X, Y]]$, then $\mathbf{not}\, S$ defines a complement operator in $\alpha[\delta[X, Y]]$, where $\mathbf{not}\, S$ is the cover-set for all elements of type X that are not covered by some member of S. We have:

$$\begin{array}{l} \forall\, S : \alpha[\delta[X, Y]] \bullet \\ \qquad (S \,\oplus\, \mathbf{not}\, S) = \top \wedge (S \,\otimes\, \mathbf{not}\, S) = \bot \end{array}$$

Theorem 5.1.7 *The poset* $(\alpha[\delta[X, Y]], \leq)$ *forms a lattice with complement operator* **not**.

Proof This follows from the definition of $\leq$ as an antisymmetric preorder, the properties of **not**, and the definitions of $\oplus$ and $\otimes$.

Commutative Laws. We observe that changing the order of the operands/Adjacency-sets does not change the composition result.

$$\begin{array}{l} \forall\, S, T : \alpha[\delta[X, Y]] \bullet \\ \quad S \oplus T = T \oplus S \wedge \\ \quad S \otimes T = T \otimes S \end{array}$$

Associative Laws. We observe that the order in which the operations are performed does not change the outcome of the operation.

$$\begin{array}{l} \forall\, S, T, U : \alpha[\delta[X, Y]] \bullet \\ \quad S \oplus (T \oplus U) = (S \oplus T) \oplus U \wedge \\ \quad S \otimes (T \otimes U) = (S \otimes T) \otimes U \end{array}$$

Absorption Laws. The following identities link $\oplus$ and $\otimes$.

$$\begin{array}{l} \forall\, S, T : \alpha[\delta[X, Y]] \bullet \\ \quad S \oplus (S \otimes T) = S \wedge \\ \quad S \otimes (S \oplus T) = S \end{array}$$

Idempotent Laws. We observe that for all $S \in \alpha[\delta[X, Y]]$, S is idempotent with respect to $\oplus$ and $\otimes$.

$$\begin{array}{l} \forall\, S : \alpha[\delta[X, Y]] \bullet \\ \quad S \oplus S = S \wedge \\ \quad S \otimes S = S \end{array}$$

Identity Laws. We observe that $(\alpha[\delta[X, Y]], \oplus, \otimes, \bot, \top)$ is a bounded lattice/algebraic structure, such that $(\alpha[\delta[X, Y]], \oplus, \otimes)$ is a lattice, $\bot$ is the identity element for the join operation $\oplus$, and $\top$ is the identity element for the meet operation $\otimes$.

$$\begin{array}{l} \forall\, S : \alpha[\delta[X, Y]] \bullet \\ \quad S \oplus \bot = S \wedge \\ \quad S \otimes \top = S \end{array}$$

Distributivity Laws. The join operation distributes over the meet operation and vice-versa.

$$\begin{aligned}&\forall\, S, T, U : \alpha[\delta[X, Y]] \bullet\\ &\qquad S \oplus (T \otimes U) = (S \oplus T) \otimes (T \oplus U) \land\\ &\qquad S \otimes (T \oplus U) = (S \otimes T) \oplus (T \otimes U)\end{aligned}$$

Thus, $(\alpha[\delta[X, Y]], \leq, \oplus, \otimes, \bot, \top, \mathbf{not})$ is a lattice. ▮

5.2 $\mathcal{FW}_1$ Filter Conditions

In this section, the filter condition attribute datatypes for the $\mathcal{FW}_1$ policy model are defined.

OSI Layer 2. Let $\mathcal{L}_2$ define the set of all additional filter condition attributes at the Data-Link Layer, given as the set of all duplets over the set of all sets of packet-types ($\mathbb{P}\, \mathit{PktTpe}$) and the set of all sets of MAC addresses ($\mathbb{P}\, \mathit{MAC}$).

$$\mathcal{L}_2 == \delta[\mathbb{P}\, \mathit{PktTpe}, \mathbb{P}\, \mathit{MAC}]$$

From Lemma 5.1.5, we have that $\mathcal{L}_2$ is a lattice.

OSI Layer 7. Let $\mathcal{L}_7$ define the set of all additional filter condition attributes at the OSI Application Layer, given as the set of all duplets over the set of all sets of Layer 7 protocols ($\mathbb{P}\, \mathit{Proto}_7^L$), the set of all closed subsets for the ranges of all Linux UIDs partitioned by adjacency ($\mathit{UID}_{\mathit{Spec}}$), and the set of all closed subsets for the ranges of all Linux GIDs ($\mathit{GID}_{\mathit{Spec}}$) partitioned by adjacency.

$$\mathit{UID}_{\mathit{Spec}} == \alpha[\mathcal{IV}[0, \mathsf{maxUID}]]$$

$$\mathit{GID}_{\mathit{Spec}} == \alpha[\mathcal{IV}[0, \mathsf{maxGID}]]$$

$$\mathcal{L}_7 == \delta[\mathbb{P}\, \mathit{Proto}_7^L, \delta[\mathit{UID}_{\mathit{Spec}}, \mathit{GID}_{\mathit{Spec}}]]$$

From Lemma 5.1.5, we have that $\mathcal{L}_7$ is a lattice.

The Stateful/Protocol Datatype. Let *Protocol* define the set of all stateful protocols, given as the set of all duplets over the TCP protocol ($\mathbb{P}\, \mathit{Flag}_{\mathit{Spec}}$), the UDP protocol ($\mathit{UDP}$), the ICMP protocol ($\mathbb{P}\, \mathit{TypesCodes}$), and the set of all sets of connection tracking states for a packet/connection ($\mathbb{P}\, \mathit{State}$).

$$\mathit{Protocol} == \delta[\mathbb{P}\,\mathit{Flag}_{\mathit{Spec}}, \delta[\mathit{UDP}, \delta[\mathbb{P}\,\mathit{TypesCodes}, \mathbb{P}\,\mathit{State}]]]$$

A benefit of the modelling approach to state in this dissertation, is that the powerset ordering defined over the set of iptables `state` literals enables a consistent means of comparing and composing the stateful aspect of firewall rules in the proposed model. While the approach in [66] provides a more expressive model of stateful rules using general automata, the approach taken to compare and compose stateful firewall rules is more complex than the proposed model in this dissertation. From Lemma 5.1.5, we have that *Protocol* is a lattice.

Stateful/Protocol Disjointness. A pair of stateful protocols are disjoint if the TCP, UDP and ICMP attributes are disjoint, and/or their state is disjoint. For $\mathsf{t}_1, \mathsf{t}_2 \in \mathbb{P}\,\mathit{Flag}_{\mathit{Spec}}$, $\mathsf{u}_1, \mathsf{u}_2 \in \mathit{UDP}$, $\mathsf{i}_1, \mathsf{i}_2 \in \mathbb{P}\,\mathit{TypesCodes}$ and $\mathsf{s}_1, \mathsf{s}_2 \in \mathbb{P}\,\mathit{State}$, we define:

$$\begin{aligned}&(\mathsf{t}_1, \mathsf{u}_1, \mathsf{i}_1, \mathsf{s}_1) \mid_{\mathit{Protocol}} (\mathsf{t}_2, \mathsf{u}_2, \mathsf{i}_2, \mathsf{s}_2) \Leftrightarrow \\ &\qquad ((\mathsf{t}_1 \mid_{\mathbb{P}\,\mathit{Flag}_{\mathit{Spec}}} \mathsf{t}_2 \wedge \mathsf{u}_1 \mid_{\mathit{UDP}} \mathsf{u}_2 \wedge \mathsf{i}_1 \mid_{\mathbb{P}\,\mathit{TypesCodes}} \mathsf{i}_2) \vee \mathsf{s}_1 \mid_{\mathbb{P}\,\mathit{State}} \mathsf{s}_2)\end{aligned}$$

Additional Filtering Specifications. Let *AdditionalFC* define the set of all additional filter condition attributes of interest, given as the set of all duplets over the set of all closed subsets for the ranges of all Unix timestamps, from $\mathsf{0}$ up to and including $\mathsf{maxTime}$ ($\mathit{Time}_{\mathit{Spec}}$), the set of all sets of all network interfaces on a machine ($\mathbb{P}\,\mathit{IFACE}$), the set of all sets of directions for direction-oriented filtering ($\mathbb{P}\,\mathit{Dir}$) and the set of all sets of iptables chains ($\mathbb{P}\,\mathit{Chain}$).

$$\begin{aligned}&\mathit{Time}_{\mathit{Spec}} == \alpha[\mathcal{IV}[0, \mathsf{maxTime}]] \\ &\mathit{AdditionalFC} == \delta[\mathit{Time}_{\mathit{Spec}}, \delta[\mathbb{P}\,\mathit{IFACE}, \delta[\mathbb{P}\,\mathit{Dir}, \mathbb{P}\,\mathit{Chain}]]]\end{aligned}$$

From Lemma 5.1.5, we have that *AdditionalFC* is a lattice.

Filter Conditions. A filter condition is an eight-tuple (s, sprt, d, dprt, p, l_2, l_7, a), representing network traffic originating from source IP ranges s, with source port ranges sprt, destined for destination IP ranges d, with destination port ranges dprt, using stateful-protocols p, with additional Layer 2 attributes l_2, additional Layer 7 attributes l_7 and additional filtering specifications a. Let *FC* define the set of all filter conditions, where:

$$\mathit{FC} == \delta[\mathit{IP}_{\mathit{Spec}}, \delta[\mathit{Prt}_{\mathit{Spec}}, \delta[\mathit{IP}_{\mathit{Spec}}, \delta[\mathit{Prt}_{\mathit{Spec}}, \delta[\mathit{Protocol}, \delta[\mathcal{L}_2, \delta[\mathcal{L}_7, \mathit{AdditionalFC}]]]]]]]$$

From Lemma 5.1.5, we have that FC is a lattice.

Forward Adjacency. Recall, for $(a_1, b_1), (a_2, b_2) \in \delta[X, Y]$, we define forward adjacency, whereby: $(a_1 = a_2 \land b_1 \wr_Y b_2)$. A pair of filter conditions are forward adjacent if the attributes in the first coordinate are equal, and there is one adjacent attribute in the second coordinate, while all other attributes in the second coordinate are equal.

5.3 The $\mathcal{FW}_1$ Firewall Algebra

In this section, we define an algebra $\mathcal{FW}_1$, for constructing and reasoning about anomaly-free firewall policies. We focus on stateless and stateful firewall policies that are defined in terms of constraints on source/destination IP/port ranges, the TCP, UDP and ICMP protocols, and additional filter condition attributes. A firewall policy defines the filter conditions that may be allowed or denied by a firewall. Let *Policy* define the set of all firewall policies, whereby:

$$Policy == \{A, D : \alpha[FC] \mid \forall\, a : A;\ d : D \bullet a \mid_{FC} d\}$$

A firewall policy $(A, D) \in Policy$ defines a policy as a disjoint pair of adjacency-free sets of filter conditions under the duplet adjacency ordering, whereby a filter condition $f \in A$ should be allowed by the firewall, while a filter condition $f \in D$ should be denied. Given $(A, D) \in Policy$, then A and D are disjoint: this avoids any contradiction in deciding whether a filter condition should be allowed or denied. The policy destructor functions *allow* and *deny* are analogous to functions *first* and *second* for ordered pairs:

$$\begin{array}{|l}
allow,\\
deny : Policy \rightarrow \alpha[FC]\\
\hline
\forall\, A, D : \alpha[FC] \bullet\\
\qquad allow\,(A, D) = A \land deny\,(A, D) = D
\end{array}$$

Thus, we have for all $P \in Policy$ then $P = (allow(P), deny(P))$.

Lemma 5.3.1 *Policy defines the set of anomaly-free policies.*

Proof Given a policy $(A, D) \in Policy$, as A and D are adjacency-free sets, then A has no redundancy and D has no redundancy, as Adjacency-sets have no subsumption. Therefore, all packets matched in filter conditions allowed by the policy are distinct, as are all packets matched in filter conditions that are denied by the policy. Given a policy $P \in \mathit{Policy}$, as $\mathit{allow}(P)$ and $\mathit{deny}(P)$ are disjoint, then P has no shadowing.

$$\forall\, P : \mathit{Policy} \bullet \mathit{allow}(P) \mid_{\alpha[FC]} \mathit{deny}(P)$$

As P has no subsumption and P has no shadowing, then P has no generalised filter conditions and P has no correlated filter conditions. Therefore, as P has no redundancy/shadowing/generalisation/correlation, then *Policy* defines the set of *anomaly-free* policies. $\blacksquare$

Note that $(A, D) \in \mathit{Policy}$ need not partition $\lceil FC \rceil$: the allow and deny sets define the filter conditions to which the policy *explicitly* applies, and an *implicit* default decision is applied for those filter conditions in $\lceil FC \rceil \setminus_{\alpha[FC]} (A \oplus D)$. For the purposes of modelling iptables firewalls it is sufficient to assume *default deny*, though we observe that $\mathcal{FW}_1$ can also be used to reason about *default allow* firewall policies.

Policy Refinement. An ordering can be defined over firewall policies, whereby given $P, Q \in \mathit{Policy}$ then $P \sqsubseteq Q$ means that P is no less restrictive than Q, that is, any filter condition that is denied by Q is denied by P. Intuitively, policy P is considered to be a *safe replacement* for policy Q, in the sense of [56, 57, 77] and any firewall that enforces policy Q can be reconfigured to enforce policy P without any loss of security. The set *Policy* forms a lattice under the safe replacement ordering and is defined as follows.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\mathcal{FW}_1} \\
\hline
\bot, \top : \mathit{Policy} \\
\_ \sqsubseteq \_ : \mathit{Policy} \leftrightarrow \mathit{Policy} \\
\_ \sqcap \_, \\
\_ \sqcup \_ : \mathit{Policy} \times \mathit{Policy} \rightarrow \mathit{Policy} \\
\hline
\bot = (\emptyset, \lceil FC \rceil) \wedge \top = (\lceil FC \rceil, \emptyset) \\
\forall\, P, Q : \mathit{Policy} \bullet \\
\qquad P \sqsubseteq Q \Leftrightarrow ((\mathit{allow}\, P \leq \mathit{allow}\, Q) \wedge \\
\qquad\qquad\qquad (\mathit{deny}\, Q \leq \mathit{deny}\, P)) \wedge \\
\qquad P \sqcap Q = (\mathit{allow}\, P \otimes \mathit{allow}\, Q, \\
\qquad\qquad\qquad \mathit{deny}\, P \oplus \mathit{deny}\, Q) \wedge \\
\qquad P \sqcup Q = (\mathit{allow}\, P \oplus \mathit{allow}\, Q, \\
\qquad\qquad\qquad \mathit{deny}\, P \otimes \mathit{deny}\, Q) \\
\hline
\end{array}
$$

Formally, $P \sqsubseteq Q$ iff every filter condition allowed by P is allowed by Q and that any filter conditions explicitly denied by Q are also explicitly denied by P. Note that in this definition we distinguish between filter conditions *explicitly* denied in the policy versus those *implicitly* denied by default. This means that, everything else being equal, a policy that explicitly denies a filter condition is considered more restrictive than a policy that relies on the implicit default-deny for the same network traffic pattern. Safe replacement is defined as the Cartesian product of Adjacency orderings over allow and deny sets and it therefore follows that $(\mathit{Policy}, \sqsubseteq)$ is a poset. $\bot$ and $\top$ define the most restrictive and least restrictive policies, that is, for any $P \in \mathit{Policy}$ we have $\bot \sqsubseteq P \sqsubseteq \top$. Thus, for example, any firewall enforcing a policy P can be safely reconfigured to enforce the (not very useful) firewall policy $\bot$.

Theorem 5.3.2 *The set of all policies Policy forms a lattice under safe replacement, with greatest lower bound ($\sqcap$) and lowest upper bound ($\sqcup$) operators in $\mathcal{FW}_1$.*

Proof The ordering of adjacency-free filter condition/duplets is a lattice under subsumption, the Cartesian product is a lattice under the definitions of glb and lub, therefore, $\mathcal{FW}_1$ is a lattice. ∎

Note, the lattice properties of the $\mathcal{FW}_0$ policy algebra, described in Chapter 4 Section 4.2.1, also apply to *Policy* policies in the $\mathcal{FW}_1$ algebra.

Policy Intersection. Under this ordering, the meet $P \sqcap Q$, of two firewall policies P and Q is defined as the policy that denies any filter condition that is explicitly denied by *either* P or Q, but allows filter conditions that are allowed by *both* P and Q. Intuitively, this means that if a firewall is required to enforce both policies P and Q, it can be configured to enforce the policy $(P \sqcap Q)$ since $P \sqcap Q$ is a safe replacement for both P and Q, that is; $(P \sqcap Q) \sqsubseteq P$ and $(P \sqcap Q) \sqsubseteq Q$. Given the definition of safe replacement as a product of two Adjacency lattices, it follows that the policy meet provides the glb operator. Thus, $P \sqcap Q$ provides the 'best'/least restrictive safe replacement (under $\sqsubseteq$) for both P and Q.

Policy Union. The join of two firewall policies P and Q is defined as the policy that allows any filter condition allowed by *either* P or Q, but denies filter conditions that are explicitly denied by *both* P and Q. Intuitively, this means that a firewall that is required to enforce either policy P or Q can be safely configured to enforce the policy $(P \sqcup Q)$. Since $\sqcup$ provides a lub operator we have $P \sqsubseteq (P \sqcup Q)$ and $Q \sqsubseteq (P \sqcup Q)$.

5.3.1 Constructing Firewall Policies

The lattice of policies $\mathcal{FW}_1$ provides us with an algebra for constructing and interpreting firewall polices. The following constructor functions are used to build primitive policies, and are analogous to the $\mathcal{FW}_0$ constructors defined in Section 4.2.2. Given $A \in \alpha[FC]$, then $(\mathsf{Allow}\, A)$ is a policy that allows filter conditions in A, and $(\mathsf{Deny}\, D)$ is a policy that explicitly denies filter conditions in D. This provides a *weak* interpretation of allow and deny.

$$\begin{array}{|l}
\mathsf{Allow},\\
\mathsf{Deny} : \alpha[FC] \rightarrow \mathit{Policy}\\
\hline
\forall\, S : \alpha[FC] \bullet\\
\qquad \mathsf{Allow}\, S = (S, \emptyset) \wedge\\
\qquad \mathsf{Deny}\, S = (\emptyset, S)
\end{array}$$

The following provides us with a *strong* interpretation for these constructors:

$$\begin{array}{|l}
\mathsf{Allow}^+,\\
\mathsf{Deny}^+ : \alpha[FC] \rightarrow \mathit{Policy}\\
\hline
\forall\, S : \alpha[FC] \bullet\\
\qquad \mathsf{Allow}^+\, S = (S, \mathbf{not}\, S) \wedge\\
\qquad \mathsf{Deny}^+\, S = (\mathbf{not}\, S, S)
\end{array}$$

whereby $(\mathsf{Allow}^+\, A)$ allows filter conditions specified in A, while explicitly denying all other filter conditions, and $(\mathsf{Deny}^+\, D)$ denies filter conditions specified in D while allowing all other filter conditions.

A firewall policy $P \in \mathit{Policy}$ can be decomposed into its corresponding allow and deny sets, and re-constructed using the algebra; for any $(A, D) \in \mathit{Policy}$, since A and D are disjoint then:

Lemma 5.3.3 *Given $A, D \in \alpha[FC]$, then:*

$$(\mathsf{Allow}^+\, A) \sqcup (\mathsf{Deny}\, D) = (A, \lceil FC \rceil \setminus_{\alpha[FC]} A) \sqcup (\emptyset, D)$$

Proof

$$\begin{aligned}(\mathsf{Allow}^{+}\,A) \sqcup (\mathsf{Deny}\,D) &= (\mathsf{Allow}\,A) \sqcap (\mathsf{Deny}^{+}\,D)\\ &= (A, D)\end{aligned}$$

∎

Corollary 5.3.4 *Given $A, D \in \alpha[FC]$, then:*

$$(\mathsf{Deny}^{+}\,D) \sqcap (\mathsf{Allow}\,A) = (A, \emptyset) \sqcap (\lceil FC \rceil \setminus_{\alpha[FC]} D, D)$$

Proof It follows from Lemma 5.3.3 that Corollary 5.3.4 holds. ∎

5.4 Reasoning About Policies in Practice

Sequential Composition. The algebra $\mathcal{FW}_1$ can be extended to include a form of sequential composition of policies. The definition for $\mathcal{FW}_1$ sequential policy composition is analogous to that given in Chapter 4 Section 4.3 for the $\mathcal{FW}_0$ algebra. The policy constructions in Section 5.3.1 can be regarded as representing the individual rules of a conventional firewall policy.

Let $(\mathsf{Allow}\,A) \mathbin{;} Q$ denote a sequential composition of an allow rule $(\mathsf{Allow}\,A)$ with policy Q with the interpretation that a given network packet matched in A is allowed; if it does not match in A then policy Q is enforced. The resulting policy either: allows filter conditions in A (and denies all other filter conditions), or allows/denies filter conditions in accordance with policy Q. We define:

$$\begin{aligned}(\mathsf{Allow}\,A) \mathbin{;} Q &= (\mathsf{Allow}^{+}\,A) \sqcup Q\\ &= ((A \oplus \mathit{allow}(Q)), ((\lceil FC \rceil \setminus_{\alpha[FC]} A) \otimes \mathit{deny}(Q)))\\ &= ((A \oplus \mathit{allow}(Q)), (\mathit{deny}(Q) \setminus_{\alpha[FC]} A))\end{aligned}$$

which is as expected. A similar definition can be provided for the sequential composition $(\mathsf{Deny}\,D) \mathbin{;} Q$, whereby a given network packet that is matched in D is denied; if it does not match in D then policy Q is enforced. We define:

$$\begin{aligned}(\mathsf{Deny}\,D) \mathbin{;} Q &= (\mathsf{Deny}^{+}\,D) \sqcap Q\\ &= (((\lceil FC \rceil \setminus_{\alpha[FC]} D) \otimes \mathit{allow}(Q)), \mathit{deny}(Q) \oplus D)\\ &= (\mathit{allow}(Q) \setminus_{\alpha[FC]} D, \mathit{deny}(Q) \oplus D)\end{aligned}$$

While in practice its usual to write a firewall policy in terms of many constructions of allow and deny rules, in principle, any firewall policy $P \in \mathit{Policy}$ can be defined in terms of one allow policy $(\mathsf{Allow}\, \mathit{allow}(P))$ and one deny policy $(\mathsf{Deny}\, \mathit{deny}(P))$ and since the allow and deny sets of P are disjoint we have $P \fatsemi Q = (\mathsf{Deny}\, \mathit{deny}(P)) \fatsemi (\mathsf{Allow}\, \mathit{allow}(P)) \fatsemi Q$. We define this as:

$$
\begin{array}{|l}
\_ \fatsemi \_ : \mathit{Policy} \times \mathit{Policy} \rightarrow \mathit{Policy} \\
\hline
\forall\, \mathcal{FW}_1;\ P, Q : \mathit{Policy} \bullet \\
\qquad P \fatsemi Q = (\mathsf{Deny}^{+}\ (\mathit{deny}(P))) \sqcap \\
\qquad\qquad (Q \sqcup (\mathsf{Allow}^{+}\ (\mathit{allow}(P))))
\end{array}
$$

Let *Rule* define the set of all firewall rules, whereby:

$$
\begin{array}{l}
\mathit{Rule} ::= \mathsf{allow}\ \langle\!\langle FC \rangle\!\rangle\ | \\
\qquad \mathsf{deny}\ \langle\!\langle FC \rangle\!\rangle
\end{array}
$$

In Chapter 9 Section 9.2, we consider an additional target action of *log* for firewall rules. We define a rule interpretation function as:

$$
\begin{array}{|l}
\mathcal{I} : \mathit{Rule} \rightarrow \mathit{Policy} \\
\hline
\forall\, f : FC \bullet \\
\qquad \mathcal{I}(\mathsf{allow}\, f) = \mathsf{Allow}(\{f\}) \wedge \\
\qquad \mathcal{I}(\mathsf{deny}\, f) = \mathsf{Deny}(\{f\})
\end{array}
$$

A firewall policy is defined as a sequence of rules $\langle r_1, r_2, .., r_n \rangle$, for $r_i \in \mathit{Rule}$, and is encoded in the policy algebra as $\mathcal{I}(r_1) \fatsemi \mathcal{I}(r_2) \fatsemi .. \fatsemi \mathcal{I}(r_n)$.

Policy Negation. The policy negation of $P \in \mathit{Policy}$ allows filter conditions explicitly denied by P and explicitly denies filter conditions allowed by P. We define:

$$
\begin{array}{|l}
\mathbf{not} : \mathit{Policy} \rightarrow \mathit{Policy} \\
\hline
\forall\, \mathcal{FW}_1;\ P : \mathit{Policy} \bullet \\
\qquad \mathbf{not}\, P = (\mathsf{Allow}^{+}\ (\mathit{deny}\,(P))) \sqcup \\
\qquad\qquad (\mathsf{Deny}\ (\mathit{allow}\,(P)))
\end{array}
$$

From this definition it follows that $(\mathbf{not}\ P)$ is $(\mathit{deny}\,(P), \mathit{allow}\,(P))$ and thus $\mathbf{not}\,(\mathsf{Deny}\, D) = (\mathsf{Allow}\, D)$ and $\mathbf{not}\,(\mathsf{Allow}\, A) = (\mathsf{Deny}\, A)$. Given that policy negation is defined similarly to the policy negation operation in the $\mathcal{FW}_0$ algebra, we observe that in general, policy negation does not define a complement operator in the algebra $\mathcal{FW}_1$, that is, it not necessarily the case that $(P \sqcup \textbf{not } P) = \top$ and $(P \sqcap \textbf{not } P) = \bot$. However, the sub-lattice of policies with allow and deny sets that exactly partition the same set $S \leq \lceil FC \rceil$ has policy negation as complement $(\mathit{allow}(P) \oplus \mathit{deny}(P) = S$ for all P in the sub-lattice).

5.4.1 Standards Compliance

RFC 5735 [33], details fifteen IPv4 address blocks that have been assigned by IANA for specialized/global purposes. The special-use IPv4 addresses are detailed in Table 5.3.

Address Block	Present Use	Range Start	Range End	Num IPs
0.0.0.0/8	"This" Network	0.0.0.0	0.255.255.255	2^{24}
10.0.0.0/8	Private-Use Networks	10.0.0.0	10.255.255.255	2^{24}
127.0.0.0/8	Loopback	127.0.0.0	127.255.255.255	2^{24}
169.254.0.0/16	Link Local	169.254.0.0	169.254.255.255	2^{16}
172.16.0.0/12	Private-Use Networks	172.16.0.0	172.31.255.255	2^{20}
192.0.0.0/24	IETF Protocol Assignments	192.0.0.0	192.0.0.255	2^{8}
192.0.2.0/24	TEST-NET-1	192.0.2.0	192.0.2.255	2^{8}
192.88.99.0/24	6to4 Relay Anycast	192.88.99.0	192.88.99.255	2^{8}
192.168.0.0/16	Private-Use Networks	192.168.0.0	192.168.255.255	2^{16}
198.18.0.0/15	Network Interconnect Device Benchmark Testing	198.18.0.0	198.19.255.255	2^{17}
198.51.100.0/24	TEST-NET-2	198.51.100.0	198.51.100.255	2^{8}
203.0.113.0/24	TEST-NET-3	203.0.113.0	203.0.113.255	2^{8}
224.0.0.0/4	Multicast	224.0.0.0	239.255.255.255	2^{28}
240.0.0.0/4	Reserved for Future Use	240.0.0.0	255.255.255.255	2^{28}
255.255.255.255/32	Limited Broadcast	255.255.255.255	255.255.255.255	1

Table 5.3: IANA special-use IPv4 addresses

Some of these address spaces may appear on the Internet, and may be used legitimately outside a single administrative domain, however, while the assigned values of the address blocks do not directly raise security issues; unexpected use may indicate an attack [33]. For example, packets with a source IP address from the private address space 172.16.0.0/12, arriving on the Wide Area Network interface of a network router, may be considered spoofed, and may be part of a Denial of Service (DoS), or Distributed DoS attack.

RFC 5735 Compliance. Best practice recommendations are implemented for each of the fifteen specialized IP address block ranges in [33], resulting in one hundred and twenty iptables *deny* rules. In [55], we defined this *deny* ruleset for a firewall management tool. We define IP spoof-mitigation policies for each iptables chain separately. For the INPUT chain, a compliance policy $\mathsf{RFC5735}^{\mathsf{I}}$ is defined, whereby for each of the IP address block ranges, the following iptables rules are enforced.

```
iptables -A INPUT -i $in \
     -m iprange --src-range $min:$max -j DROP

iptables -A INPUT -i $in \
     -m iprange --dst-range $min:$max -j DROP
```

Similarly, for the OUTPUT chain, an IP spoof-mitigation compliance policy $\mathsf{RFC5735}^{\mathsf{O}}$ is defined, whereby for each of the specialized IP address block ranges we have:

```
iptables -A OUTPUT -o $out \
     -m iprange --src-range $min:$max -j DROP

iptables -A OUTPUT -o $out \
     -m iprange --dst-range $min:$max -j DROP
```

For the FORWARD chain, then $\mathsf{RFC5735}^{\mathsf{F}}$ enforces the following iptables rules for each of the IP address block ranges.

```
iptables -A FORWARD -i $in \
     -m iprange --src-range $min:$max -j DROP

iptables -A FORWARD -i $in \
     -m iprange --dst-range $min:$max -j DROP

iptables -A FORWARD -o $out \
     -m iprange --src-range $min:$max -j DROP

iptables -A FORWARD -o $out \
     -m iprange --dst-range $min:$max -j DROP
```

Each policy, $\mathsf{RFC5735}^{\mathsf{I}}, \mathsf{RFC5735}^{\mathsf{O}}, \mathsf{RFC5735}^{\mathsf{F}}$, terminates with a final iptables rule that specifies all other traffic be permitted on the given iptables chain.

A Redefined Firewall Policy. We model these iptables rules in the algebra by redefining some policy-model attributes, and provide more formal definitions of

$\mathsf{RFC5735}^{\mathsf{I}}, \mathsf{RFC5735}^{\mathsf{O}}$ and $\mathsf{RFC5735}^{\mathsf{F}}$. Let $AdditionalFC_I$ be the set of all duplets for additional filter condition attributes of interest, whereby:

$$AdditionalFC_I == \delta[\mathbb{P}\, Chain, \delta[\mathbb{P}\, Dir, \mathbb{P}\, IFACE]]$$

A revised definition for the set of all filter conditions $FC_{\mathcal{I}}$ is given as:

$$FC_{\mathcal{I}} == \delta[IP_{Spec}, \delta[Prt_{Spec}, \delta[IP_{Spec}, \delta[Prt_{Spec}, \delta[Protocol, AdditionalFC_I]]]]]$$

A revised definition for the set of all policies $Policy_{\mathcal{I}}$ is given as:

$$Policy_{\mathcal{I}} == \{A, D : \alpha[FC_{\mathcal{I}}] \mid \forall\, a : A;\; d : D \bullet a \mid_{FC_{\mathcal{I}}} d\}$$

The compliance policies $\mathsf{RFC5735}^{\mathsf{I}}, \mathsf{RFC5735}^{\mathsf{O}}, \mathsf{RFC5735}^{\mathsf{F}} \in Policy_{\mathcal{I}}$, define the minimum requirements for what it means for some perimeter network firewall policy to mitigate the threat of IP spoofing for all traffic, in accordance with RFC 5735. Thus, we have for all $P \in Policy_{\mathcal{I}}$ if:

$$P \sqsubseteq (\mathsf{RFC5735}^{\mathsf{I}} \sqcap \mathsf{RFC5735}^{\mathsf{O}} \sqcap \mathsf{RFC5735}^{\mathsf{F}})$$

then P complies with the best practice recommendations outlined in [33] for IP address spoof-mitigation. In Chapter 6, a prototype policy management toolkit that implements $Policy_{\mathcal{I}}$ firewall policies is described.

5.4.2 Anomaly Detection

We have by definition, the adjacency-free allow and deny sets of some $P \in Policy$ are disjoint, therefore P is anomaly-free by construction. We can however define anomalies using the algebra; by considering how a policy changes when composed with other policies. The definitions given in this section are analogous to those given in Chapter 4 Section 4.3.1 for the $\mathcal{FW}_0$ algebra.

Redundancy. A policy P is redundant given policy Q if their composition results in no difference between the resulting policy and Q, in particular, if:

$$P \mathbin{⨟} Q = Q$$

Further definitions may be given for redundancy. For example, there are redundant packets with a target action of allow in filter conditions between policy

P and policy Q, if:

$$\mathsf{Allow}(allow\,(P)) \sqcap \mathsf{Allow}(allow\,(Q)) \neq (\emptyset, \emptyset)$$

as:

$$\mathsf{Allow}(allow\,(P)) \sqcap \mathsf{Allow}(allow\,(Q)) = (allow\,(P) \otimes allow\,(Q), \emptyset)$$

A similar interpretation follows for redundant packets with a target action of deny between filter conditions in a policy P and filter conditions in a policy Q. In particular, we have redundant denies if:

$$\mathsf{Deny}(deny\,(P)) \sqcup \mathsf{Deny}(deny\,(Q)) \neq (\emptyset, \emptyset)$$

as:

$$\mathsf{Deny}(deny\,(P)) \sqcup \mathsf{Deny}(deny\,(Q)) = (\emptyset, deny\,(P) \otimes deny\,(Q))$$

Shadowing. Some part of policy Q is shadowed by the entire policy P in the composition $P \fatsemi Q$ if the filter condition constraints that are specified by P contradict the constraints that are specified by Q, in particular, if:

$$(\textbf{not}\ P) \fatsemi Q = Q$$

This is a very general definition for shadowing, as discussed in Chapter 4 Section 4.3.1 when reasoning about shadowing anomalies using the $\mathcal{FW}_0$ algebra.

Further definitions may also be given for shadowing. For example, we have that some of the packets denied by filter conditions in a policy P shadow some of the packets allowed by filter conditions in a policy Q if:

$$\mathsf{Deny}(deny\,(P)) \sqcup \mathsf{Deny}(allow\,(Q)) \neq (\emptyset, \emptyset)$$

as:

$$\mathsf{Deny}(deny\,(P)) \sqcup \mathsf{Deny}(allow\,(Q)) = (\emptyset, deny\,(P) \otimes allow\,(Q))$$

Similarly, some of the packets allowed by filter conditions in a policy P shadow some of the packets denied by filter conditions in a policy Q if:

$$\mathsf{Allow}(allow\,(P)) \sqcap \mathsf{Allow}(deny\,(Q)) \neq (\emptyset, \emptyset)$$

as:

$$\mathsf{Allow}(allow\,(P)) \sqcap \mathsf{Allow}(deny\,(Q)) = (allow\,(P) \otimes deny\,(Q), \emptyset)$$

Generalisation. A generalisation anomaly exists between P and Q if some of the packets allowed by filter conditions in P shadow some of the packets denied by filter conditions in Q, in particular, if:

$$\mathsf{Allow}(allow\,(P)) \sqcap \mathsf{Allow}(deny\,(Q)) \neq (\emptyset, \emptyset)$$

and, those packets shadowed by filter conditions in Q are subsumed by Q's denies:

$$\textbf{not}\,(\mathsf{Allow}(allow\,(P)) \sqcap \mathsf{Allow}(deny\,(Q))) \neq \mathsf{Deny}(deny\,(Q))$$

whereby:

$$\textbf{not}\,(\mathsf{Allow}(allow\,(P)) \sqcap \mathsf{Allow}(deny\,(Q))) = (\emptyset, allow\,(P) \otimes deny\,(Q))$$

Similarly, a generalisation anomaly exists between P and Q if some of the packets denied by filter conditions in P shadow some of the packets allowed by filter conditions in Q, in particular, if:

$$\mathsf{Deny}(deny\,(P)) \sqcup \mathsf{Deny}(allow\,(Q)) \neq (\emptyset, \emptyset)$$

and, those packets shadowed by filter conditions in Q are subsumed by Q's allows:

$$\textbf{not}\,(\mathsf{Deny}(deny\,(P)) \sqcup \mathsf{Deny}(allow\,(Q))) \neq \mathsf{Allow}(allow\,(Q))$$

as:

$$\textbf{not}\,(\mathsf{Deny}(deny\,(P)) \sqcup \mathsf{Deny}(allow\,(Q))) = (deny\,(P) \otimes allow\,(Q), \emptyset)$$

Inter-policy Anomalies. We can also use the $\mathcal{FW}_1$ algebra to reason about anomalies between the different policies of distributed firewall configurations. In the following, assume that P is a policy on an *upstream* firewall and Q is a policy on a *downstream* firewall.

Redundancy. An inter-redundancy anomaly exists between policies P and Q if some part of Q is redundant to some part of P, whereby the target action of the redundant filter conditions is *deny*. Given some set of filter conditions A

denied by P, and some set of filter conditions B denied by Q, then there exists an inter-redundancy between P and Q, if:

$$(\mathsf{Deny}\, A) \fatsemi (\mathsf{Deny}\, B) = (\mathsf{Deny}\, A)$$

Further definitions may be given for inter-redundancy. For example, there are redundant packets with a target action of deny in filter conditions between upstream policy P and downstream policy Q, if:

$$\mathsf{Deny}(\mathit{deny}\,(P)) \sqcup \mathsf{Deny}(\mathit{deny}\,(Q)) \neq (\emptyset, \emptyset)$$

Shadowing. An inter-shadowing anomaly exists between policies P and Q if some part of Q's allows are shadowed by some part of P's denies. Given some set of filter conditions A denied by P, and some set of filter conditions B allowed by Q, then there is an inter-shadowing anomaly between P and Q, if:

$$(\mathsf{Deny}\, A) \fatsemi (\mathsf{Allow}\, B) = (\mathsf{Deny}\, A)$$

Further definitions may also be given for inter-shadowing. For example, we have that some of the packets denied by filter conditions in a policy P shadow some of the packets allowed by filter conditions in a policy Q if:

$$\mathsf{Deny}(\mathit{deny}\,(P)) \sqcup \mathsf{Deny}(\mathit{allow}\,(Q)) \neq (\emptyset, \emptyset)$$

Spuriousness. An inter-spuriousness anomaly exists between policies P and Q if some part of Q's denies are shadowed by some part of P's allows. Given some set of filter conditions A allowed by P, and some set of filter conditions B denied by Q, then there exists an inter-spuriousness anomaly between P and Q, if:

$$(\mathsf{Allow}\, A) \fatsemi (\mathsf{Deny}\, B) = (\mathsf{Allow}\, A)$$

Spuriousness may also be defined as follows, whereby some of the packets allowed by filter conditions in a policy P shadow some of the packets denied by filter conditions in a policy Q. We have an inter-spuriousness anomaly from an upstream policy P to a downstream policy Q, if:

$$\mathsf{Allow}(\mathit{allow}\,(P)) \sqcap \mathsf{Allow}(\mathit{deny}\,(Q)) \neq (\emptyset, \emptyset)$$

5.5 Discussion

In this chapter, a policy algebra $\mathcal{FW}_1$ is defined in which firewall policies can be specified and reasoned about. At the heart of this algebra is the notion of safe replacement, that is, whether it is secure to replace one firewall policy by another. The set of policies form a lattice under safe replacement and this enables consistent operators for safe composition to be defined. Policies in this lattice are anomaly-free by construction, and thus, composition under glb and lub operators preserves anomaly-freedom. A policy sequential composition operator is also proposed that can be used to interpret firewall policies defined more conventionally as sequences of rules.

The algebra can be used to characterize anomalies, such as redundancy and shadowing, that arise from policy composition. To consider the types of stateful anomalies from [66] in the proposed model $\mathcal{FW}_1$, then it would be necessary to apply additional constraints when constructing and composing anomaly-free firewall policies. For example, a policy that specifies a firewall rule that enables the establishment of a TCP connection from host X to host Y, should also include rules that allow for the various other permissible transitions of the TCP protocol. A similar approach would be required when considering more complex time-based rule anomalies. Suppose, for example, a policy specifies a rule that allows outbound SSH traffic from host X to host Y between 6 a.m. and 8 p.m. on Tuesdays, but is missing firewall rule/s permitting inbound SSH traffic during the same time period between X and Y. Similar to the notion of more complex stateful anomalies, it would also be necessary to apply additional constraints when constructing and composing anomaly-free firewall policies to consider more complex time-based anomalies. Best practice policy compliance may be defined using $\sqsubseteq$. The algebra $\mathcal{FW}_1$ provides a formal interpretation of the network access controls for a partial mapping of the iptables filter table. $\mathcal{FW}_1$ is a generic algebra and can also be used to model other firewall systems.

In [5], a firewall policy is modelled as a single rooted tree, relations between rules are defined on a pairwise basis, and definitions for firewall configuration anomalies are provided. In [6], the work is extended to distributed firewall policies. In [35], a firewall policy is modelled as a linked-list, and in [73] rule relations within a policy are modelled in a directed graph. In [159] Binary Decision Diagrams are used to model firewall rulesets. In [22], a theorem-proving approach is used to reason about firewall policies. We model a firewall policy as an ordered pair of disjoint adjacency-free sets, where the set of policies *Policy* forms a lattice under $\sqsubseteq$, and each $P \in \mathit{Policy}$ is anomaly-free by construction.

In [5, 6, 35, 73, 159] an algorithmic approach is taken to detect/resolve anomalies. We follow an algebraic (as opposed to algorithmic) approach towards modelling anomalies in a single policy, and across a distributed policy configuration through policy composition. In [161], a firewall policy algebra is proposed. However, the authors note that an anomaly-free composition is not guaranteed as a result of using their algebraic operators. Our work differs, in that policy composition under the $\sqcap, \sqcup$ and $\fatsemi$ operators defined in this chapter all result in anomaly-free policies. The proposed algebra $\mathcal{FW}_1$ is used to reason about and compose anomaly-free policies and therefore we do not have to worry about dealing with conflicts that may arise. Anomaly conflicts are dealt with in composition by computing anomaly-free policies, rather than using techniques such as [78] to resolve conflicts in policy decisions.

Yang and Lam [158] propose the Atomic Predicate (AP) Verifier tool. AP Verifier reduces the set of predicates representing network packet filters to a set of atomic predicates, with the aim of speeding up network reachability testing. A predicate is represented as a set of integers that define the atomic predicates. The performance analysis of AP Verifier is compared with other tools, and demonstrates the proposed approach is significantly more time- and space-efficient. Belhaouane et al. [17] propose a formal method to verify when two Access Control Lists are functionally equivalent/isofunctional. In this dissertation, we argue that there is an isomorphic mapping between policies in the $\mathcal{FW}_0$ and $\mathcal{FW}_1$ firewall algebras, that is, every firewall policy $P \in \mathit{Policy}$ has a corresponding unique representation $\mathcal{P} \in \mathit{Policy}_0$ and vice-versa.

In [62], we developed the algebra $\mathcal{FW}_0$ (extended in Chapter 4), and used it to reason over host-based and network access controls in OpenStack. In [62], we focused on stateless firewall policies that are defined in terms of constraints on individual IPs, ports and protocols. The $\mathcal{FW}_1$ algebra is defined over stateful firewall policies constructed in terms of constraints on source/destination IP/-port ranges, the TCP, UDP and ICMP protocols, and additional filter condition attributes. We argue that $\mathcal{FW}_1$ gives a more expressive means for reasoning over OpenStack security group and perimeter firewall configurations. In Chapter 7, the $\mathcal{FW}_1$ algebra is used to construct a policy model for OpenStack host-based and network access controls.

Part III

Firewall Policy Algebras in Practice

Chapter 6

Implementing The $\mathcal{FW}_1$ Policy Algebra

This chapter describes a prototype policy management toolkit that implements the $\mathcal{FW}_1$ $Policy_{\mathcal{I}}$ firewall policies defined in Chapter 5 Section 5.4.1, for iptables. Experiments for policy operators are conducted and the results are presented. This chapter is an extended version of the results reported for the prototype in [101], and is organised as follows. In Section 6.1 the prototype construction is described, we give efficient implementations of firewall rule (duplet) join and firewall rule (duplet) difference, and the experimentation methodology is presented. In Section 6.2, the sequential composition operator for $Policy_{\mathcal{I}}$ policies is evaluated. Policy union is evaluated in Section 6.3, as is policy intersection in Section 6.4. In Section 6.5 we evaluate policy compliance.

6.1 Implementing iptables Policies

A prototype policy management toolkit has been implemented in Python for iptables. We reason over an implementation of the $Policy_{\mathcal{I}}$ policies described in Chapter 5 Section 5.4.1, using the $\fatsemi$, $\sqcup, \sqcap$ and $\sqsubseteq$ policy operators. The test-bed for the experiments is a 64-Bit Ubuntu 14.04 LTS OS, running on a Dell Latitude E6430, with a quad-core Intel i5-3320M processor and 4GB of RAM. Every experiment was conducted three times; the median result chosen for inclusion. In this section, the construction of the firewall datatypes for the prototype and the methodology for generating the test-data is described.

Firewall Rules. An iptables rule is modelled as a list of generic filter conditions. The current implementation defines firewall rules with filter condition attributes for source/destination IP/port ranges, the ICMP, UDP and TCP protocols, and additional filter condition attributes. The relationships of adjacency, disjointness and subsumption have been encoded, as have composition operators for rule intersection and rule join/combination.

Range-based Attributes. Filter condition attributes defined as ranges in the $\mathcal{FW}_1$ framework, for example, source/destination IP/port ranges, are implemented as interval sets, using the Pyinter Python package [104]. The package was modified to include definitions for relative compliment operators and adjacency over intervals and interval sets.

ICMP and UDP. The ICMP protocol is modelled as the set of all valid ICMP Type/Code pairs, given in Chapter 3 Section 3.2.2. UDP has been defined as a binary attribute. Boolean operators apply for the UDP filter condition attributes.

TCP. The TCP protocol is encoded as a 2^{12} bit array, whereby the position of each bit is mapped to an index value in a table. This table is the implementation of the $Flag_{Spec}$ object defined in Chapter 3 Section 3.2.3, and is encoded as the list of TCP (mask, comp) pairs, as pairs of six-bit arrays; as depicted in Figure 6.1.

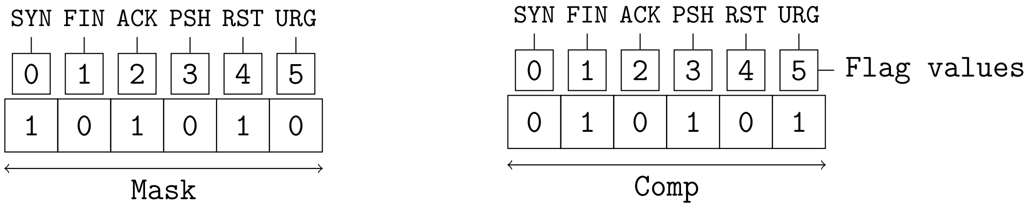


Figure 6.1: A TCP $Flag_{Spec}$ element

A value of 1 at a given position in the 2^{12} bit array indicates that this particular arrangement of TCP flags are matched in the packets specified by the firewall rule. Table 6.1 gives an overview of the *FlagSpec* lookup.

Index	Mask	Comp
1	‘000000’	‘000000’
2	‘000000’	‘000001’
. .	. .	. .
8	‘000000’	‘000111’
. .	. .	. .
4096	‘111111’	‘111111’

Table 6.1: Partial TCP *FlagSpec* Lookup Table

Additional Attributes. Attributes for direction-oriented filtering, network interface and iptables chains have been encoded as sets for firewall rules.

Firewall Policies. A policy is implemented as a disjoint pair of adjacency-free sets of firewall rules. The adjacency-free sets of rules have been modelled following the approach taken to model the interval-sets in the Pyinter package [104].

Transitive Closure of Adjacent Rules. The transitive closure for the adjacency relation over rules in firewall policies has been implemented recursively, following the approach used in the Pyinter package to implement the transitive closure over adjacent intervals [104]. A set of firewall rules is adjacency-free by construction. When a new rule is added to the ruleset, a check is made firstly to determine if there are any rules in the set that are adjacent to the new rule. If there are none, the new rule is added. Otherwise, the adjacent rules are removed from the ruleset, and rules resulting from the combination of the new rule with the adjacent rules are added to the ruleset, starting again with a check to determine if there are any rules in the set that are adjacent to the new rule.

6.1.1 Implementing Duplet Join and Difference

In this section, the implementations of duplet/rule join and duplet/rule difference are defined.

Duplet Combination. For $f, g \in \delta[X, Y]$, then the combination operation $(f \uplus_{\delta[X,Y]} g)$ defines a set of adjacency-free duplets that exactly cover f and g.

$$
\begin{array}{l}
[X] \\
\hline
\_ \uplus_{\_} \_ : \mathbb{P}\, X \nrightarrow (X \times X) \rightarrow \mathbb{P}\, X \\
\hline
\forall\, a, b : X \bullet \\
\quad \forall\, c : (a \uplus_X b) \bullet \\
\qquad a \overset{X}{\leftarrow} c \vee b \overset{X}{\leftarrow} c \wedge \\
\qquad\quad \nexists\, d : (a \uplus_X b) \bullet \\
\qquad\qquad c \wr_X d \mid c \neq d \\
\hline
\end{array}
$$

The operation is described using three recursive functions; *center* $\mathcal{C}(f, g)$, *left* $\mathcal{L}(f, g)$ and *right* $\mathcal{R}(f, g)$, and is defined as the set union of the duplets resulting from $\mathcal{C}(f, g)$, $\mathcal{L}(f, g)$ and $\mathcal{R}(f, g)$. For ease of exposition, a duplet is given as sequence of filter condition attributes. We assume f and g always have the same number of attributes. The functions are defined as follows.

Center. For $f, g \in \delta[X, Y]$, then $\mathcal{C}(f, g)$ defines the join of the adjacent and common attributes in f and g. For duplets comprising two attributes, we define:

$$\mathcal{C}(\langle a_1, b_1\rangle, \langle a_2, b_2\rangle) = \langle a_1 \cup_X a_2, b_1 \cap_Y b_2\rangle$$

Table 6.2 specifies the operations and duplet resulting from $\mathcal{C}(f, g)$ for two-attribute duplets. The label D signifies duplet, while A means the attribute.

A D	$\boxed{1}$	$\boxed{2}$
$\boxed{1}$	$\boxed{a_1 \cup_X a_1}$	$\boxed{b_1 \cap_Y b_2}$

Table 6.2: *Center* function for two-attribute duplets

For f and g of length greater than two, we define for each additional attribute:

$$\mathcal{C}(f\,\hat{}\,\langle c_1\rangle, g\,\hat{}\,\langle c_2\rangle) = \mathcal{C}(f, g) \frown \langle c_1 \cap_Y c_2\rangle$$

Table 6.3 specifies the operations and duplet resulting from $\mathcal{C}(f, g)$ for duplets with three attributes.

A D	$\boxed{1}$	$\boxed{2}$	$\boxed{3}$
$\boxed{1}$	$\boxed{a_1 \cup_X a_1}$	$\boxed{b_1 \cap_Y b_2}$	$\boxed{c_1 \cap_Y c_2}$

Table 6.3: *Center* function for three-attribute duplets

Left. For $f, g \in \delta[X, Y]$, then $\mathcal{L}(f, g)$ defines the remaining attribute constraints covered in f, that are not covered in $\mathcal{C}(f, g)$. For duplets comprising two attributes, we define:

$$\mathcal{L}(\langle a_1, b_1\rangle, \langle a_2, b_2\rangle) = \{\langle a_1, b_1 \setminus_Y b_2\rangle\}$$

Table 6.4 specifies the operations and duplet resulting from $\mathcal{L}(f, g)$ for two-attribute duplets.

$D \diagdown A$	$\boxed{1}$	$\boxed{2}$
$\boxed{1}$	$\boxed{a_1}$	$\boxed{b_1 \setminus_Y b_2}$

Table 6.4: *Left* function for two-attribute duplets

For f and g of length greater than two, we define for each additional attribute:

$$\begin{aligned}\mathcal{L}(f\,\hat{}\,\langle c_1 \rangle, g\,\hat{}\,\langle c_2 \rangle) = \{&\langle \mathit{head}\, f \rangle \frown \mathit{tail}\,(\mathcal{C}(f,g)) \frown \langle c_1 \setminus_Y c_2 \rangle\} \cup \\ &\{t : \mathcal{L}(f,g) \bullet t \frown \langle c_1 \rangle\}\end{aligned}$$

Table 6.5 specifies the operations and duplets resulting from $\mathcal{L}(f,g)$ for duplets comprising three attributes.

$D \diagdown A$	$\boxed{1}$	$\boxed{2}$	$\boxed{3}$
$\boxed{1}$	$\boxed{a_1}$	$\boxed{b_1 \setminus_Y b_2}$	$\boxed{c_1}$
$\boxed{2}$	$\boxed{a_1}$	$\boxed{b_1 \cap_Y b_2}$	$\boxed{c_1 \setminus_Y c_2}$

Table 6.5: *Left* function for three-attribute duplets

Right. For $f, g \in \delta[X, Y]$, then $\mathcal{R}(f,g)$ defines the remaining attribute constraints covered in g, that are not covered in $\mathcal{C}(f,g)$. For duplets comprising two attributes, we define:

$$\mathcal{R}(\langle a_1, b_1 \rangle, \langle a_2, b_2 \rangle) = \{\langle a_2, b_2 \setminus_Y b_1 \rangle\}$$

Table 6.6 specifies the operations and duplet resulting from $\mathcal{R}(f,g)$ for two-attribute duplets.

$D \diagdown A$	$\boxed{1}$	$\boxed{2}$
$\boxed{1}$	$\boxed{a_2}$	$\boxed{b_2 \setminus_Y b_1}$

Table 6.6: *Right* function for two-attribute duplets

For f and g of length greater than two, we define for each additional attribute:

$$\begin{aligned}\mathcal{R}(f\,\hat{}\,\langle c_1 \rangle, g\,\hat{}\,\langle c_2 \rangle) = \{&\langle \mathit{head}\, g \rangle \frown \mathit{tail}\,(\mathcal{C}(f,g)) \frown \langle c_2 \setminus_Y c_1 \rangle\} \cup \\ &\{t : \mathcal{R}(f,g) \bullet t \frown \langle c_2 \rangle\}\end{aligned}$$

Table 6.7 specifies the operations and duplets resulting from $\mathcal{R}(f, g)$ for duplets comprising three attributes.

A D	$\boxed{1}$	$\boxed{2}$	$\boxed{3}$
$\boxed{1}$	$\boxed{a_2}$	$\boxed{b_2 \setminus_Y b_1}$	$\boxed{c_2}$
$\boxed{2}$	$\boxed{a_2}$	$\boxed{b_2 \cap_Y b_1}$	$\boxed{c_2 \setminus_Y c_1}$

Table 6.7: *Right* function for three-attribute duplets

Thus, we define the combination operation for f and g as:

$$f \uplus_{\delta[X,Y]} g = \{\mathcal{C}(f, g)\} \cup \mathcal{L}(f, g) \cup \mathcal{R}(f, g)$$

Theorem 6.1.1 *For $f, g \in \delta[X, Y]$, then $f \uplus_{\delta[X,Y]} g$ defines the adjacency-free combination for all $n \in \mathbb{N}$, where $(n = \# f = \# g) \geq 2$.*

Proof We will show that for $f, g \in \delta[X, Y]$, then $f \uplus_{\delta[X,Y]} g$ defines the adjacency-free combination for all $n \in \mathbb{N}$, where $(n = \# f = \# g) \geq 2$, using induction on n.

Base Case. For $n = 2$, then for $f \uplus_{\delta[X,Y]} g$, the resulting operations and duplets for f and g as two attribute duplets are given in Table 6.8:

A D	$\boxed{1}$	$\boxed{2}$
$\boxed{1}$	$\boxed{a_1 \cup_X a_1}$	$\boxed{b_1 \cap_Y b_2}$
$\boxed{2}$	$\boxed{a_1}$	$\boxed{b_1 \setminus_Y b_2}$
$\boxed{3}$	$\boxed{a_2}$	$\boxed{b_2 \setminus_Y b_1}$

Table 6.8: A two-attribute duplet join

Therefore, Theorem 6.1.1 holds when $n = 2$.

Inductive Hypothesis. Suppose Theorem 6.1.1 holds for $k \in \mathbb{N}$, where $k > n$, and $k = \# f = \# g$. Then for $f \uplus_{\delta[X,Y]} g$, the resulting operations and duplets for f and g as k-attribute duplets are given in Table 6.9:

A D	$\boxed{1}$	$\boxed{2}$	$\boxed{\cdot\,\cdot}$	$\boxed{k}$
$\boxed{1}$	$a_1 \cup_X a_1$	$b_1 \cap_Y b_2$	$\cdot\,\cdot$	$x_1 \cap_Y x_2$
$\boxed{2}$	a_1	$b_1 \setminus_Y b_2$	$\cdot\,\cdot$	x_1
$\boxed{3}$	a_2	$b_2 \setminus_Y b_1$	$\cdot\,\cdot$	x_2
$\cdot\,\cdot$	$\cdot\,\cdot$	$\cdot\,\cdot$	$\cdot\,\cdot$	x_1
$\cdot\,\cdot$	$\cdot\,\cdot$	$\cdot\,\cdot$	$\cdot\,\cdot$	x_2
$(k+k-2)$	a_1	$b_1 \cap_Y b_2$	$\cdot\,\cdot$	$x_1 \setminus_Y x_2$
$(k+k-1)$	a_2	$b_2 \cap_Y b_1$	$\cdot\,\cdot$	$x_2 \setminus_Y x_1$

Table 6.9: A k-attribute duplet join

Inductive Step. Let $n = k + 1$. Then by the recursive definitions of $\mathcal{C}(f, g)$, $\mathcal{L}(f, g)$ and $\mathcal{R}(f, g)$, the resulting operations and duplets for f and g as $(k + 1)$-attribute duplets in $(f \uplus_{\delta[X,Y]} g)$ are given in Table 6.10, whereby:

A D	$\boxed{1}$	$\boxed{2}$	$\boxed{\cdot\,\cdot}$	$\boxed{k}$	$\boxed{k+1}$
$\boxed{1}$	$a_1 \cup_X a_1$	$b_1 \cap_Y b_2$	$\cdot\,\cdot$	$x_1 \cap_Y x_2$	$y_1 \cap_Y y_2$
$\boxed{2}$	a_1	$b_1 \setminus_Y b_2$	$\cdot\,\cdot$	x_1	y_1
$\boxed{3}$	a_2	$b_2 \setminus_Y b_1$	$\cdot\,\cdot$	x_2	y_2
$\cdot\,\cdot$	$\cdot\,\cdot$	$\cdot\,\cdot$	$\cdot\,\cdot$	x_1	y_1
$\cdot\,\cdot$	$\cdot\,\cdot$	$\cdot\,\cdot$	$\cdot\,\cdot$	x_2	y_2
$(k+k-2)$	a_1	$b_1 \cap_Y b_2$	$\cdot\,\cdot$	$x_1 \setminus_Y x_2$	y_1
$(k+k-1)$	a_2	$b_2 \cap_Y b_1$	$\cdot\,\cdot$	$x_2 \setminus_Y x_1$	y_2
$(k+k)$	a_1	$b_1 \cap_Y b_2$	$\cdot\,\cdot$	$x_1 \cap_Y x_2$	$y_1 \setminus_Y y_2$
$(k+k+1)$	a_2	$b_2 \cap_Y b_1$	$\cdot\,\cdot$	$x_2 \cap_Y x_1$	$y_2 \setminus_Y y_1$

Table 6.10: A $(k + 1)$-attribute duplet join

Therefore, Theorem 6.1.1 holds for $n = k + 1$. By the principal of mathematical induction, the theorem holds for all $n \in \mathbb{N}$, where $n \geq 2$. ∎

Algorithm 1 summarises the duplet combination operation.

```
Combine(f, g, len, ruleSet) /* combines duplets f and g. */
    input : a pair of duplets $(f, g)$, the number of attributes ($len$), an
            empty list to hold the result ($ruleSet$)
    output: the list ($ruleSet$), where $ruleSet = \langle \mathcal{C}(f, g), \mathcal{L}(f, g), \mathcal{R}(f, g) \rangle$
    if ($len == 2$) then
        $ruleSet(0) \leftarrow \langle a_1 \cup_X a_2, b_1 \cap_Y b_2 \rangle$
        $ruleSet(1) \leftarrow \{\langle a_1, b_1 \setminus_Y b_2 \rangle\}$
        $ruleSet(2) \leftarrow \{\langle a_2, b_2 \setminus_Y b_1 \rangle\}$
        return $ruleSet$
    else
        $len \leftarrow len - 1$
        $i \leftarrow \# ruleSet(0)$
        $l \leftarrow {\frown}/\langle \langle head\, f \rangle, tail\,(ruleSet(0)), \langle f(i) \setminus_Y g(i) \rangle \rangle$
        $r \leftarrow {\frown}/\langle \langle head\, g \rangle, tail\,(ruleSet(0)), \langle g(i) \setminus_Y f(i) \rangle \rangle$
        $ruleSet(0) \leftarrow ruleSet(0) \frown \langle f(i) \cap_Y g(i) \rangle$
        $ruleSet(1) \leftarrow \{t : ruleSet(1) \bullet t \frown \langle f(i) \rangle\} \cup \{l\}$
        $ruleSet(2) \leftarrow \{t : ruleSet(2) \bullet t \frown \langle g(i) \rangle\} \cup \{r\}$
        return Combine(f, g, len, ruleSet)
```

Algorithm 1: Duplet combination operation

A Bottom-up Approach to Duplet Join. Recall, the definition for the join operation of $S, T \in \alpha[\delta[X, Y]]$ given in Chapter 5 Section 5.1.3 is constructed following a top-down approach with respect to the ordering relation $\leq$, whereby:

$$S \oplus T = \lceil \{a, b : \bigcap \{U : \mathbb{P}(\delta[X, Y]) \mid (\forall\, c : U \bullet \exists\, a : S;\ b : T \bullet c \overset{\delta[X,Y]}{\rightarrow} a \vee c \overset{\delta[X,Y]}{\rightarrow} b)\} \mid a \wr^{+}_{\delta[X,Y]} b \bullet a \cup_{\delta[X,Y]} b\} \rceil$$

For all $S, T \in \alpha[\delta[X, Y]]$, we define the implementation definition for sets of adjacency-free duplets as:

$$S \oplus T = \lceil \{a, b : \lceil \bigcup \{a, b : (S \cup T) \mid a \wr^{+}_{(S \cup T)} b \bullet (a \uplus_{\delta[X,Y]} b)\} \rceil \mid a \wr^{+}_{\delta[X,Y]} b \bullet a \cup_{\delta[X,Y]} b\} \rceil$$

Adjacency Duplet Union Implementation. The implementation definition for the join of $S, T \in \alpha[\delta[X, Y]]$ is defined as the cover-set for the duplet merge of the transitive closure of adjacent duplets, from the coverset for the generalized union of sets from the duplet combination operation ($\_ \uplus_{\_} \_$), for all transitively adjacent duplets in S and T. The coverset for the the generalized union defines the smallest collection of duplets that cover all of the duplets from both S and T by precedence subsumption. Given that all duplets in this set are now disjoint, the cover-set for the duplet merge of the transitive closure of adjacent duplets merges any forward-adjacent duplets from S and T. If we take some $U \in \alpha[\delta[X, Y]]$, such that $U \leq (S \oplus T)$ and $S \leq U \wedge T \leq U$, then $(S \oplus T) = U$. Thus, the implementation definition for duplet Adjacency-set join provides a lowest upper bound operator.

Theorem 6.1.2 *The two given definitions for joining sets of adjacency-free duplets are equivalent.*

$$
\begin{aligned}
&\forall\, S, T : \alpha[\delta[X, Y]] \bullet \\
&\qquad S \oplus T = \lceil \{a, b : \bigcap \{ U : \mathbb{P}(\delta[X, Y]) \mid (\forall\, c : U \bullet \exists\, a : S;\ b : T \bullet \\
&\qquad\qquad c \overset{\delta[X,Y]}{\rightarrow} a \vee c \overset{\delta[X,Y]}{\rightarrow} b)\} \mid a \wr^{+}_{\delta[X,Y]} b \bullet a \cup_{\delta[X,Y]} b\} \rceil \\
&\qquad\quad = \lceil \{a, b : \lceil \bigcup \{a, b : (S \cup T) \mid a \wr^{+}_{(S \cup T)} b \bullet (a \uplus_{\delta[X,Y]} b)\} \rceil \mid \\
&\qquad\qquad a \wr^{+}_{\delta[X,Y]} b \bullet a \cup_{\delta[X,Y]} b\} \rceil
\end{aligned}
$$

Proof Given that both definitions define the cover-set for the duplet merge of the transitive closure of forward adjacent duplets from the smallest collection of disjoint adjacency-free duplets that cover all of the duplets from both S and T by precedence subsumption, then Theorem 6.1.2 holds. ▮

Duplet Difference. For $f, g \in \delta[X, Y]$, the operation $(f \setminus_{\delta[X,Y]} g)$ defines a set of adjacency-free duplets that are covered by f but not by g. The operation is described using two recursive functions; *center* $\mathcal{C}^{\mathit{diff}}(f, g)$, and the *left* $\mathcal{L}(f, g)$ function given previously. The function $(f \setminus_{\delta[X,Y]} g)$ is defined as the set union of the duplets resulting from $\mathcal{C}^{\mathit{diff}}(f, g)$ and $\mathcal{L}(f, g)$.

Center. For $f, g \in \delta[X, Y]$, then for duplets comprising two attributes; $\mathcal{C}^{\mathit{diff}}(f, g)$ is defined as follows:

$$\mathcal{C}^{\mathit{diff}}(\langle a_1, b_1 \rangle, \langle a_2, b_2 \rangle) = \langle a_1 \setminus_X a_2, b_1 \cap_Y b_2 \rangle$$

The operations and duplet resulting from $\mathcal{C}^{\mathit{diff}}(f, g)$ for two-attribute duplets are given in Table 6.11.

$D \diagdown A$	$\boxed{1}$	$\boxed{2}$
$\boxed{1}$	$\boxed{a_1 \setminus_X a_1}$	$\boxed{b_1 \cap_Y b_2}$

Table 6.11: *Center* difference function for two-attribute duplets

For f and g of length greater than two, we define for each additional attribute:

$$\mathcal{C}^{\textit{diff}}(f\,\hat{}\,\langle c_1\rangle, g\,\hat{}\,\langle c_2\rangle) = \mathcal{C}^{\textit{diff}}(f, g) \frown \langle c_1 \cap_Y c_2\rangle$$

The operations and duplet resulting from $\mathcal{C}^{\textit{diff}}(f, g)$ for duplets with three attributes are given in Table 6.12.

$D \diagdown A$	$\boxed{1}$	$\boxed{2}$	$\boxed{3}$
$\boxed{1}$	$\boxed{a_1 \setminus_X a_1}$	$\boxed{b_1 \cap_Y b_2}$	$\boxed{c_1 \cap_Y c_2}$

Table 6.12: *Center* difference function for three-attribute duplets

Thus, we define the difference operation for f and g as:

$$f \setminus_{\delta[X,Y]} g = \{\mathcal{C}^{\textit{diff}}(f, g)\} \cup \mathcal{L}(f, g)$$

Theorem 6.1.3 *For $f, g \in \delta[X, Y]$, then $f \setminus_{\delta[X,Y]} g$ defines the adjacency-free duplet difference for all $n \in \mathbb{N}$, where $(n = \# f = \# g) \geq 2$.*

Proof We will show that for $f, g \in \delta[X, Y]$, then $f \setminus_{\delta[X,Y]} g$ defines the adjacency-free duplet difference for all $n \in \mathbb{N}$, where $(n = \# f = \# g) \geq 2$, using induction on n.

Base Case. For $n = 2$, then for $f \setminus_{\delta[X,Y]} g$, the resulting operations and duplets for f and g as two-attribute duplets are given in Table 6.13.

$D \diagdown A$	$\boxed{1}$	$\boxed{2}$
$\boxed{1}$	$\boxed{a_1 \setminus_X a_1}$	$\boxed{b_1 \cap_Y b_2}$
$\boxed{2}$	$\boxed{a_1}$	$\boxed{b_1 \setminus_Y b_2}$

Table 6.13: A two-attribute duplet difference

Therefore, Theorem 6.1.3 holds when $n = 2$.

Inductive Hypothesis. Suppose Theorem 6.1.3 holds for $k \in \mathbb{N}$, where $k > n$, and $k = \# f = \# g$. Then for $f \setminus_{\delta[X,Y]} g$, the resulting operations and duplets for f and g as k-attribute duplets are given in Table 6.14.

$D \backslash A$	1	2	$\cdot\cdot$	k
1	$a_1 \setminus_X a_1$	$b_1 \cap_Y b_2$	$\cdot\cdot$	$x_1 \cap_Y x_2$
2	a_1	$b_1 \setminus_Y b_2$	$\cdot\cdot$	x_1
$\cdot\cdot$	$\cdot\cdot$	$\cdot\cdot$	$\cdot\cdot$	x_1
k	a_1	$b_1 \cap_Y b_2$	$\cdot\cdot$	$x_1 \setminus_Y x_2$

Table 6.14: A k-attribute duplet difference

Inductive Step. Let $n = k + 1$. Then by the recursive definitions of $\mathcal{C}^{diff}(f, g)$ and $\mathcal{L}(f, g)$, the resulting operations and duplets for f and g as $(k+1)$-attribute duplets in $(f \setminus_{\delta[X,Y]} g)$ are given in Table 6.15, whereby:

$D \backslash A$	1	2	$\cdot\cdot$	k	$k+1$
1	$a_1 \setminus_X a_1$	$b_1 \cap_Y b_2$	$\cdot\cdot$	$x_1 \cap_Y x_2$	$y_1 \cap_Y y_2$
2	a_1	$b_1 \setminus_Y b_2$	$\cdot\cdot$	x_1	y_1
$\cdot\cdot$	$\cdot\cdot$	$\cdot\cdot$	$\cdot\cdot$	x_1	y_1
(k)	a_1	$b_1 \cap_Y b_2$	$\cdot\cdot$	$x_1 \setminus_Y x_2$	y_1
$(k+1)$	a_1	$b_1 \cap_Y b_2$	$\cdot\cdot$	$x_1 \cap_Y x_2$	$y_1 \setminus_Y y_2$

Table 6.15: A $(k + 1)$-attribute duplet difference

Therefore, Theorem 6.1.3 holds for $n = k + 1$. By the principal of mathematical induction, the theorem holds for all $n \in \mathbb{N}$, where $n \geq 2$. ∎

Algorithm 2 summarises the duplet difference operation.

```
Difference(f, g, len, ruleSet) /* duplet difference operation. */
    input  : a pair of duplets $(f, g)$, the number of attributes ($len$), an
             empty list to hold the result ($ruleSet$)
    output: the list ($ruleSet$), where $ruleSet = \langle \mathcal{C}^{diff}(f, g), \mathcal{L}(f, g)\rangle$
    if ($len == 2$) then
        $ruleSet \leftarrow \langle\langle a_1 \setminus_X a_2, b_1 \cap_Y b_2\rangle, \{\langle a_1, b_1 \setminus_Y b_2\rangle\}\rangle$
        return $ruleSet$
    else
        $len \leftarrow len - 1$
        $i \leftarrow \#ruleSet(0)$
        $l \leftarrow {}^\frown/\langle\langle head\, f\rangle, tail\,(ruleSet(0)), \langle f(i) \setminus_Y g(i)\rangle\rangle$
        $ruleSet(0) \leftarrow ruleSet(0) {}^\frown \langle f(i) \cap_Y g(i)\rangle$
        $ruleSet(1) \leftarrow \{t : ruleSet(1) \bullet t {}^\frown \langle f(i)\rangle\} \cup \{l\}$
        return Difference(f, g, len, ruleSet)
```

Algorithm 2: Duplet difference operation

6.1.2 Generating the Datasets

Datasets have been constructed to evaluate the policy operators. The policy composition datasets were constructed in such a way as to test possible best/worst-case scenarios. A dataset has also been constructed to test policy compliance.

Datasets for Sequential Composition Experiments. Two datasets were generated for experimentation. Each dataset consists of iptables policies of size $2^4, 2^5 \,..\, 2^{11}$. This allows us to formulate a *doubling hypothesis* [128] for each dataset, whereby we determine the effect on the experiment's running time by doubling the number of rules. All dataset rules have a target action of *allow*.

Non-adjacent Dataset. This dataset contains policies where no rule is adjacent to any other rule (other than itself). The source/destination IP/port ranges for each rule in a policy are incremented, such that each new rule is not adjacent to the previous rule. The first two rules of each policy in this dataset are:

```
iptables -A INPUT -m iprange --src-range 0.0.0.1-0.0.0.50 \
    --dst-range 0.0.0.26-0.0.0.75 -p udp \
    --sport 100:110 --dport 100:110 -j ACCEPT
```

```
iptables -A INPUT -m iprange --src-range 0.0.0.51-0.0.0.100 \
    --dst-range 0.0.0.76-0.0.0.125 -p udp \
    --sport 120:130 --dport 120:130 -j ACCEPT
```

Adjacent Dataset. This dataset consists of policies where every new rule is adjacent to the previous rule, to ensure the maximum number of possible rules are generated as a result of composition. The policies have been generated by incrementing the values of the source/destination IP/port ranges for each new rule, such that the new rule is adjacent to the previous rule. The first two rules in each of the adjacent dataset policies are:

```
iptables -A INPUT -m iprange --src-range 0.0.0.1-0.0.0.50 \
    --dst-range 0.0.0.45-0.0.0.100 -p tcp \
    --sport 100:110 --dport 100:110 -j ACCEPT

iptables -A INPUT -m iprange --src-range 0.0.0.51-0.0.0.100 \
    --dst-range 0.0.0.95-0.0.0.150 -p tcp \
    --sport 105:115 --dport 105:115 -j ACCEPT
```

The datasets have been constructed in this way, as evaluating randomly generated rules is not an effective testing technique. The adjacent and non-adjacent datasets allow for a *boundary value analysis* [114] of rule composition.

Example 1 Sequentially composing the first two rules in an adjacent dataset policy results in the following re-computed adjacency-free iptables ruleset.

```
iptables -A INPUT -m iprange --src-range 0.0.0.51-0.0.0.100 \
    --dst-range 0.0.0.101-0.0.0.150 -p tcp \
    --sport 105:110 --dport 105:115 -j ACCEPT

iptables -A INPUT -m iprange --src-range 0.0.0.1-0.0.0.100 \
    --dst-range 0.0.0.95-0.0.0.100 -p tcp \
    --sport 105:110 --dport 105:110 -j ACCEPT

iptables -A INPUT -m iprange --src-range 0.0.0.1-0.0.0.50 \
    --dst-range 0.0.0.45-0.0.0.100 -p tcp \
    --sport 100:104 --dport 100:110 -j ACCEPT

iptables -A INPUT -m iprange --src-range 0.0.0.51-0.0.0.100 \
    --dst-range 0.0.0.95-0.0.0.150 -p tcp \
    --sport 111:115 --dport 105:115 -j ACCEPT
```

```
iptables -A INPUT -m iprange --src-range 0.0.0.1-0.0.0.50 \
    --dst-range 0.0.0.95-0.0.0.100 -p tcp \
    --sport 105:110 --dport 100:104 -j ACCEPT

iptables -A INPUT -m iprange --src-range 0.0.0.51-0.0.0.100 \
    --dst-range 0.0.0.95-0.0.0.100 -p tcp \
    --sport 105:110 --dport 111:115 -j ACCEPT

iptables -A INPUT -m iprange --src-range 0.0.0.1-0.0.0.50 \
    --dst-range 0.0.0.45-0.0.0.94 -p tcp \
    --sport 105:110 --dport 100:110 -j ACCEPT
```

However, sequentially composing the first two rules from a non-adjacent dataset policy results in those two rules for a re-computed adjacency-free iptables ruleset/policy. △

Dataset for Policy Union and Intersection Experiments. Each policy in the previously defined adjacent dataset, is split into two policies to construct the dataset for the lub/glb experiments. The first policy contains the odd (index) rules from the original policy. All rules in an odd-index policy are adjacency-free. The second policy contains the even (index) rules from the original policy. All rules in an even-index policy are adjacency-free. Constructing the dataset from the odd-index and even-index policies allows us to evaluate the cost, in terms of time, of composing policies of different sizes, whereby for the policy union experiments, the maximum number of rules are generated as a result of composition.

Dataset for Policy Compliance Experiments. A dataset consisting of iptables policies of size $2^5, 2^6 \mathinner{..} 2^{11}$ is generated to test policy compliance. Each policy in this dataset is RFC 5735 compliant by construction, for TCP traffic arriving on the iptables INPUT chain to/from each of the fifteen special-use IPv4 addresses [33]. Recall, the compliance policy $\mathsf{RFC5735}^{\mathsf{I}}$ defined in Chapter 5 Section 5.4.1 for $Policy_{\mathcal{I}}$ policies. The UDP rules from the previously defined non-adjacent dataset have been re-written with a target action of *deny*, and are used to construct the remaining rules for each policy in the compliance dataset.

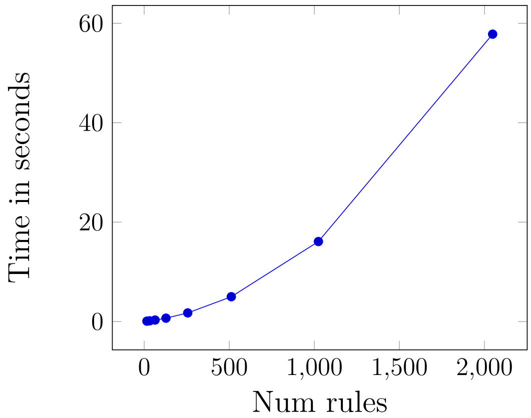
The iptables policies in the datasets described in this section are used to evaluate the time complexity of the Python prototype implementation of the $\mathcal{FW}_1$ $Policy_{\mathcal{I}}$ policy operators. In this section, when we refer to the *cost* of the evaluation of a policy in an experiment, we are referring to cost in terms of time.

6.2 Evaluating Sequential Policy Composition

In this section, the sequential composition operator for $Policy_{\mathcal{I}}$ policies is evaluated. The prototype parses the system's currently enforced iptables ruleset $\langle r_1, r_2 \mathinner{..} r_n \rangle$ by chain, using the Python-iptables package [100], and then normalizes each rule to a primitive/singleton policy. The overall policy for the chain is evaluated as $\mathcal{I}(r_1) \fatsemi \mathcal{I}(r_2) \fatsemi \mathinner{..} \fatsemi \mathcal{I}(r_n)$. Once the sequential composition of the system's currently enforced iptables policy is computed, the prototype generates a semantically-equivalent adjacency-free set of iptables rules and re-writes this new ruleset to the system. Experimentation was conducted over the adjacent and non-adjacent rulesets. Three different approaches have been implemented for evaluating sequential composition. The different approaches and results of the experiments are described in this section.

A Linear Approach. This approach is the evaluation of policies in a linear sequence, as $\mathcal{I}(r_1) \fatsemi \mathcal{I}(r_2) \fatsemi \mathinner{..} \fatsemi \mathcal{I}(r_n)$.

Non-adjacent Dataset Experiments. The first set of experiments were conducted on the non-adjacent datasets. Figure 6.2 details the results.

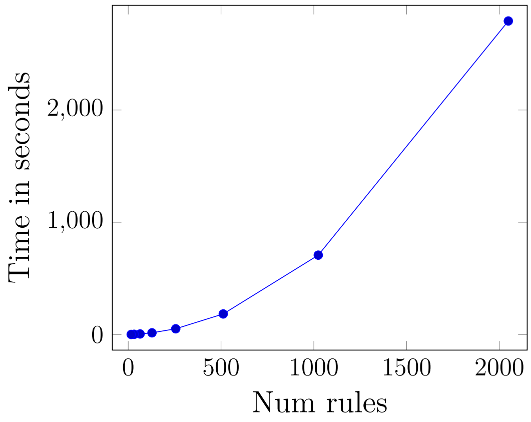


Num rules	**Time**	**Ratio**
2^4	0.07	-
2^5	0.13	1.85
2^6	0.29	2.23
2^7	0.67	2.31
2^8	1.73	2.58
2^9	4.98	2.87
2^{10}	16.09	3.23
2^{11}	57.81	3.59

Figure 6.2: Sequential composition with no adjacent rules (in seconds)

As the number of rules increase, the cost of computing the sequential composition of non-adjacent rules is relatively cheap, and is proportional to the number of rules. For the largest ruleset 2^{11}, the time taken for the evaluation of the sequential composition of the rules is approximately one minute.

Adjacent Dataset Experiments. The second set of experiments was conducted on the adjacent rulesets. Figure 6.3 gives the results.



Num rules	Time	Ratio
2^4	0.80	-
2^5	2.02	2.53
2^6	5.13	2.98
2^7	15.32	3.34
2^8	51.18	3.58
2^9	183.42	3.85
2^{10}	707.15	3.85
2^{11}	2792.81	3.94

Figure 6.3: Sequential composition with adjacent rules (in seconds)

We observe that as the number of rules increase, the cost of computing the sequential composition of adjacent rules is expensive, but is also proportional to the number of rules used in the experiment. However, the cost is by orders of magnitude more expensive than the cost for evaluating the sequential composition of non-adjacent policies of the same size. For 2^9 rules, the time taken for the evaluation of sequential composition is around three minutes, and the time taken for 2^{11} rules is approximately forty six minutes.

A Divide-and-Conquer Approach. This approach has been implemented in order to improve on the results from previous experiments using the linear approach. For this approach, we take a sequence of primitive policies $\langle \mathcal{I}(r_1), \mathcal{I}(r_2) \mathinner{\ldotp\ldotp} \mathcal{I}(r_n) \rangle$, and perform a binary-chop on the sequence; whereby a binary expression tree is constructed to compute the order of the sequential compositions. Policies are always leaves, and sequential composition operators always have a left and a right sub-tree. Figure 6.4 depicts the expression tree for 2^3 rules/primitive policies.

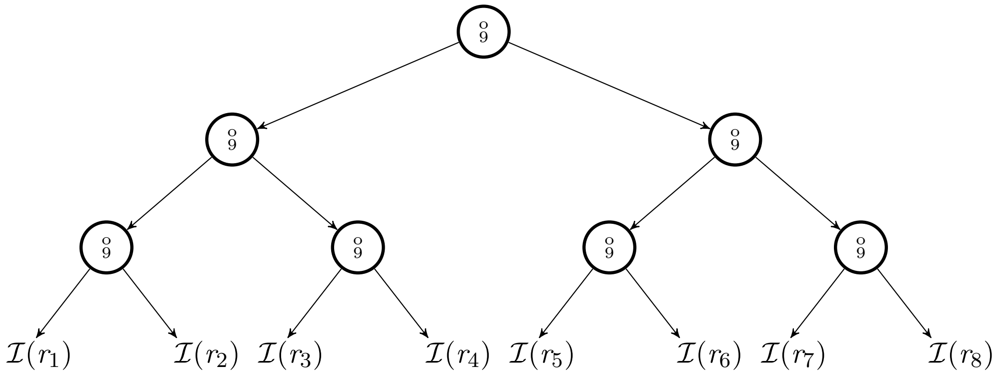


Figure 6.4: A 2^3 rule binary-chop

We have that the sequential composition for the 2^3 rules in the binary-chop is evaluated as:

$$(((\mathcal{I}(r_1) \fatsemi \mathcal{I}(r_2)) \fatsemi (\mathcal{I}(r_3) \fatsemi \mathcal{I}(r_4))) \fatsemi ((\mathcal{I}(r_5) \fatsemi \mathcal{I}(r_6)) \fatsemi (\mathcal{I}(r_7) \fatsemi \mathcal{I}(r_8))))$$

The result of this evaluation is equivalent to the policy resulting from the linear sequential composition of policies, that is, the evaluation of $\mathcal{I}(r_1) \fatsemi \mathcal{I}(r_2) \fatsemi .. \fatsemi \mathcal{I}(r_8)$.

If the number of rules/primitive policies is odd, the expression tree is balanced from left to right, as depicted in Figure 6.5.

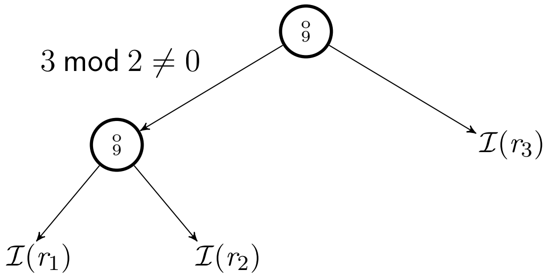
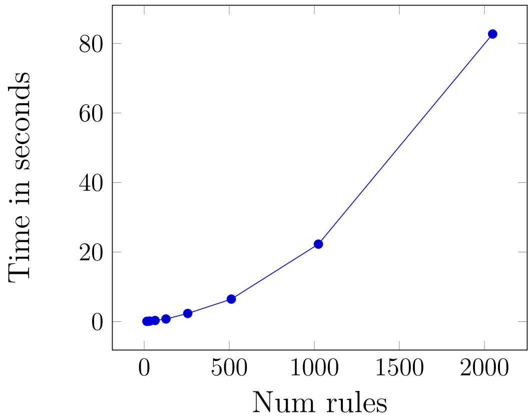


Figure 6.5: Balancing an unbalanced expression tree with three rules

Binary-chop Non-adjacent Dataset Experiments. The first set of experiments were conducted on the non-adjacent datasets. Figure 6.6 details the results.

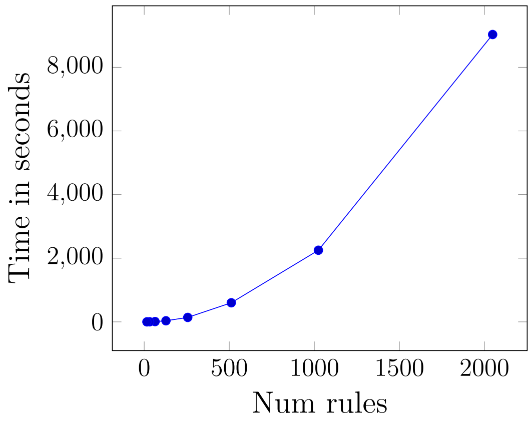


Num rules	Time	Ratio
2^4	0.07	-
2^5	0.14	2
2^6	0.31	2.21
2^7	0.75	2.41
2^8	2.33	3.10
2^9	6.46	2.77
2^{10}	22.28	3.44
2^{11}	82.73	3.71

Figure 6.6: Binary-chop evaluation with no adjacent rules (in seconds)

As the number of rules increase, the cost of computing the sequential composition of non-adjacent rules using the binary-chop is relatively cheap, and is proportional to the number of rules. We observe however, that the cost is slightly more expensive than the cost for evaluating policies of the same size following the linear approach for the non-adjacent dataset. For the largest ruleset 2^{11}, the time taken for the evaluation of the sequential composition of 2^{11} rules is approximately one minute and twenty seconds.

Binary-chop Adjacent Dataset Experiments. The second set of experiments was conducted on the adjacent rulesets. Figure 6.7 details the results.



Num rules	Time	Ratio
2^4	0.64	-
2^5	2.23	3.48
2^6	8.71	3.90
2^7	34.51	3.96
2^8	139.55	4.04
2^9	560.00	4.01
2^{10}	2250.89	4.01
2^{11}	9030.17	4.01

Figure 6.7: Binary-chop evaluation with adjacent rules (in seconds)

As the number of rules increase, the cost, in terms of time, to compute the sequential composition of adjacent rules using the binary-chop is expensive, but is also proportional to the number of rules. However, we observe that the cost is much more expensive than the cost for evaluating policies of the same size following the linear approach for the adjacent dataset. For 2^9 rules, the time taken for the evaluation of sequential composition is around nine and a half minutes, and the time taken for 2^{11} rules is approximately two and a half hours.

A Parallel Divide-and-Conquer Approach. This approach has been implemented in order to improve on the results from previous experiments using the binary-chop approach. For this approach, we again perform a binary-chop on the sequence of rules, however, we incorporate the standard Python multiprocessing package [65]. The left and right sub-expressions are evaluated in parallel, and then sequentially composed, as depicted in Figure 6.8.

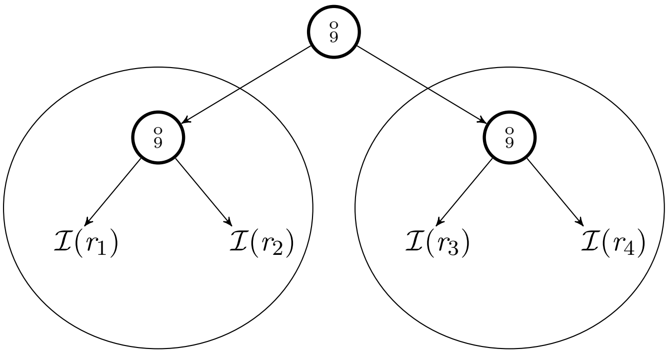
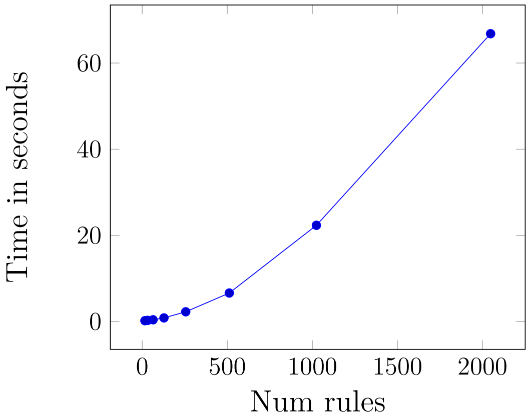


Figure 6.8: A 2^2 rule binary-chop (parallel evaluation)

Parallel Binary-chop Non-adjacent Dataset Experiments. The first set of experiments were conducted on the non-adjacent datasets. Figure 6.9 details the results.

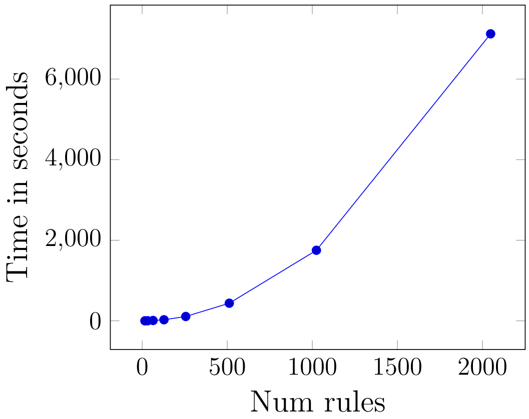


Num rules	Time	Ratio
2^4	0.18	-
2^5	0.24	1.33
2^6	0.39	1.62
2^7	0.82	2.10
2^8	2.24	2.73
2^9	6.60	2.94
2^{10}	22.32	3.38
2^{11}	66.80	2.99

Figure 6.9: Binary-chop parallel evaluation with no adjacent rules (in seconds)

As the number of rules increase, the cost of computing the sequential composition of non-adjacent rules using the parallel binary-chop is relatively cheap, and is proportional to the number of rules used in the experiment. We observe that the cost is an improvement to the cost for evaluating policies of the same size following the binary-chop approach without multiprocessing, for the non-adjacent dataset. For the largest ruleset 2^{11}, the time taken for the evaluation of the sequential composition of 2^{11} rules is approximately one minute.

Parallel Binary-chop Adjacent Dataset Experiments. The second set of experiments was conducted on the adjacent rulesets. Figure 6.10 gives the results.



Num rules	Time	Ratio
2^4	0.72	-
2^5	1.83	2.54
2^6	8.81	4.81
2^7	26.69	3.02
2^8	108.04	4.04
2^9	437.29	4.04
2^{10}	1751.13	4
2^{11}	7122.47	4.06

Figure 6.10: Binary-chop parallel evaluation with adjacent rules (in seconds)

We observe that as the number of rules increase, the cost, in terms of time, to compute the sequential composition of adjacent rules using the parallel binary-chop is expensive, and is proportional to the number of rules. We observe that for larger policies, the parallel approach is an improvement to the cost for evaluating policies of the same size following the binary-chop approach without multiprocessing, for the adjacent dataset. The evaluation time for the sequential composition of 2^9 rules is around seven and a half minutes, and the time taken for the largest dataset, 2^{11}, is approximately two hours.

6.3 Evaluating Policy Union

In this section, the join operation for the $\mathcal{FW}_1$ $Policy_{\mathcal{I}}$ policies is evaluated. Experiments are conducted, whereby each policy in the adjacent dataset described in Section 6.1, is split into two policies to construct the dataset for the lub experiments. The first policy contains the odd (index) rules from the original policy, and the second policy contains the even (index) rules from the original policy. For each $P, Q \in Policy_{\mathcal{I}}$ in this dataset, the time taken for the operation $P \sqcup Q$ is given by Table 6.16.

Q P	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
2^3	$\boxed{0.65}$	0.79	0.81	0.99	1.40	2.51	5.73	16.93
2^4	0.79	$\boxed{1.86}$	2.09	2.32	2.91	4.50	8.83	22.19
2^5	0.81	2.09	$\boxed{4.97}$	5.45	6.78	9.17	15.50	32.89
2^6	0.99	2.32	5.45	$\boxed{14.70}$	17.01	21.93	32.29	57.47
2^7	1.40	2.91	6.78	17.01	$\boxed{48.85}$	58.44	76.94	119.28
2^8	2.51	4.50	9.17	21.93	58.44	$\boxed{179.87}$	217.34	294.56
2^9	5.73	8.83	15.50	32.29	76.94	217.34	$\boxed{699.11}$	839.49
2^{10}	16.93	22.19	32.89	57.47	119.28	294.56	839.49	$\boxed{2722.63}$

Table 6.16: Time taken to compute $P \sqcup Q$ (in seconds)

A benefit of conducting the policy join experiments in this way, is that in practice, we may want to update a policy P, comprising a large number of rules, with a smaller policy Q that permits some new accesses. The time taken for composition of policies of equal size is approximately the same as (slightly less than) the time necessary to sequentially compose the rules from both policies. That is; for example, the time taken for the sequential composition of 2^9 rules is around three minutes, as is the join of the two policies of size 2^8. This is highlighted through the diagonal in the matrix, and is as expected; given that we used all *allow* rules, and the sequential composition of the rules used in these experiments results in the eventual join of the rules. Figure 6.11 gives the 3D plot for the data in Table 6.16.

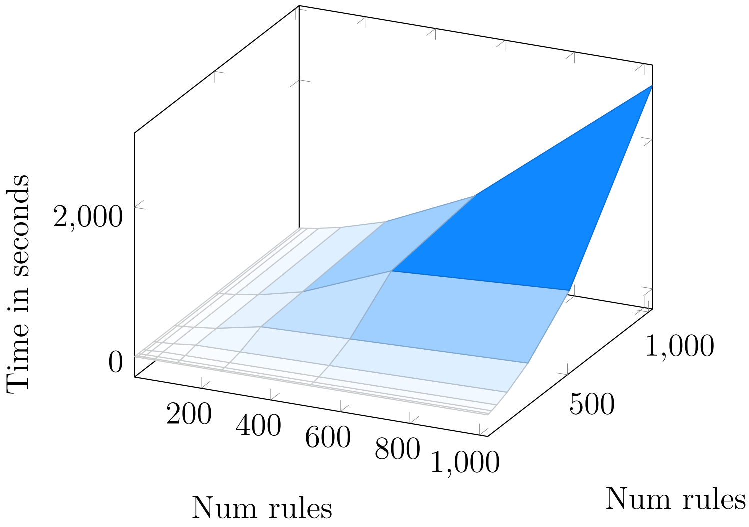


Figure 6.11: Time taken to compute $P \sqcup Q$ (in seconds)

6.4 Evaluating Policy Intersection

In this section, the meet operation for the $\mathcal{FW}_1$ $Policy_{\mathcal{I}}$ policies is evaluated. Experiments are conducted to test policy glb using the odd-index policies from the policy join experiments. For each $P, Q \in Policy_{\mathcal{I}}$ in this dataset, the time taken for the operation $P \sqcap Q$ is given in Table 6.17.

P \ Q	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
2^3	1.0×10^{-3}	2.0×10^{-3}	3.0×10^{-3}	4.0×10^{-3}	9.0×10^{-3}	1.0×10^{-2}	3.0×10^{-2}	6.0×10^{-2}
2^4	2.0×10^{-3}	4.0×10^{-3}	6.0×10^{-3}	1.0×10^{-2}	1.7×10^{-2}	3.0×10^{-2}	6.0×10^{-2}	1.2×10^{-1}
2^5	3.0×10^{-3}	6.0×10^{-3}	0.01	0.02	0.03	0.06	0.12	0.25
2^6	4.0×10^{-3}	1.0×10^{-2}	0.02	0.05	0.08	0.14	0.26	0.50
2^7	9.0×10^{-3}	1.7×10^{-2}	0.03	0.08	0.18	0.31	0.55	1.03
2^8	1.0×10^{-2}	3.0×10^{-2}	0.06	0.14	0.31	0.73	1.21	2.19
2^9	3.0×10^{-2}	6.0×10^{-2}	0.12	0.26	0.55	1.21	2.93	4.90
2^{10}	6.0×10^{-2}	1.2×10^{-1}	0.25	0.50	1.03	2.19	4.90	11.99

Table 6.17: Time taken to compute $P \sqcap Q$ (in seconds)

We observe that as the number of rules increase, the cost in terms of time to compute the intersection of the rulesets is cheap. For the largest ruleset composition of 2^{10} and 2^{10} rules, the time taken for evaluation was approximately twelve seconds. Figure 6.12 gives the 3D plot for the data in Table 6.17.

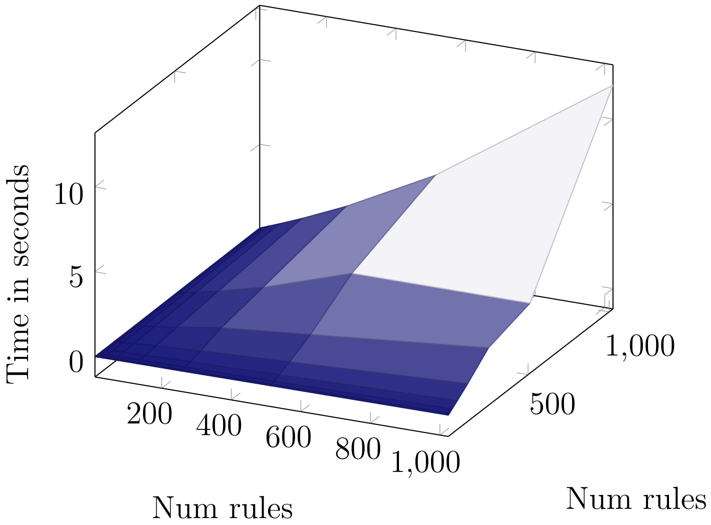
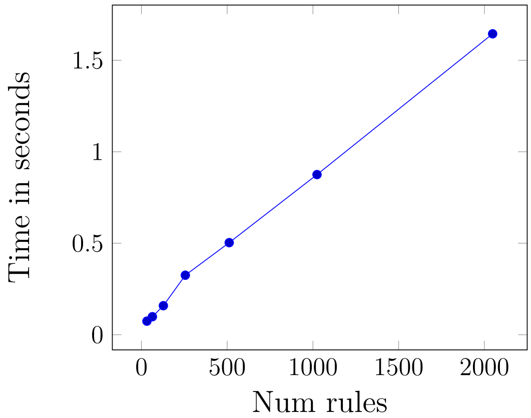


Figure 6.12: Time taken to compute $P \sqcap Q$ (in seconds)

6.5 Evaluating Policy Compliance

In this section, we evaluate the ordering relation $\sqsubseteq$ for $\mathit{Policy}_{\mathcal{I}}$ policies. For each $\mathcal{P}$ in the compliance dataset described in Section 6.1, the time taken in seconds for the evaluation of $\mathcal{P} \sqsubseteq \mathsf{RFC5735}^{\mathsf{I}}$ is given in Figure 6.13.



Num rules	Time	Ratio
2^5	0.07	-
2^6	0.09	1.28
2^7	0.15	1.66
2^8	0.32	2.13
2^9	0.50	1.56
2^{10}	0.87	1.74
2^{11}	1.64	1.88

Figure 6.13: Time taken to compute $\mathcal{P} \sqsubseteq \mathsf{RFC5735}^{\mathsf{I}}$ (in seconds)

We observe that the compliance test has a cheap cost in terms time, and all evaluation times for $\mathcal{P} \sqsubseteq \mathsf{RFC5735}^{\mathsf{I}}$ are negligible.

6.6 Discussion

In this chapter, a prototype policy management toolkit that implements $\mathcal{FW}_1$ $Policy_{\mathcal{I}}$ firewall policies in Python for iptables is described. The results in this chapter are described in terms of the algebra $\mathcal{FW}_1$, for stateful and stateless firewall policies that are defined in terms of constraints on source/destination IP/port ranges, the TCP, UDP and ICMP protocols, and additional filter condition attributes. Experiments were conducted for $\fatsemi$, $\sqcup, \sqcap$ and $\sqsubseteq$ policy operators.

Two datasets were generated to evaluate sequential policy composition, one dataset contains policies that are adjacency-free, while the other dataset consists of policies where every new rule in a policy is adjacent to the previous rule. Datasets were constructed in this way to test possible best- and worst-case scenarios for policy composition. Three different approaches have been implemented for evaluation of the sequential composition operator. For the non-adjacent dataset, the cost, of computing the sequential composition of the rules is cheap. In the worst case, evaluation time for 2^{11} rules is approximately one minute and twenty seconds using the binary-chop approach, and in the best case, using the linear approach, it is approximately one minute. For the adjacent rulesets, we observe a significant difference in the evaluation times across approaches. In the best case, evaluation time for 2^{11} rules is approximately forty six minutes using the linear approach, and in the worst case, evaluation time for 2^{11} rules is approximately two and a half hours using the divide-and-conquer/binary-chop approach. We conjecture that this is due to the computation of unnecessary transitive closures over adjacent rules, as a result of how transitive closure has been implemented, and constructing the expression tree to compute the order of the sequential compositions for the binary-chop. For example, in a 2^2 rule policy expression, as depicted in Figure 6.14, rules $\mathcal{I}(r_1)$ and $\mathcal{I}(r_2)$ are sequentially composed, the result of this is sequentially composed with the result of $(\mathcal{I}(r_3) \fatsemi \mathcal{I}(r_4))$.

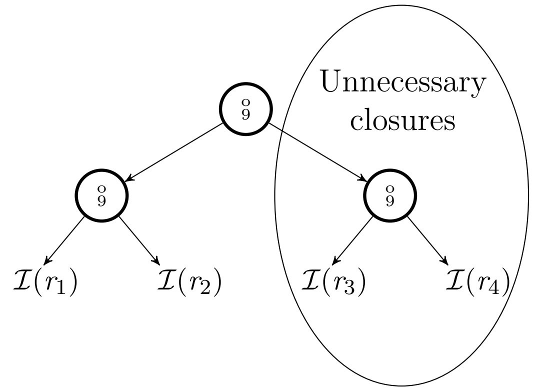


Figure 6.14: An unnecessary closure computation

The unnecessary closures in this case are computed during the sequential composition of rules $\mathcal{I}(r_3)$ and $\mathcal{I}(r_4)$, as $\mathcal{I}(r_3)$ is adjacent to $(\mathcal{I}(r_1) \fatsemi \mathcal{I}(r_2))$ and the result of the composition of $\mathcal{I}(r_3)$ and $\mathcal{I}(r_4)$ will result in a set of rules that are adjacent to $(\mathcal{I}(r_1) \fatsemi \mathcal{I}(r_2))$, rather than just the single rule $\mathcal{I}(r_3)$.

In evaluating $\sqcup$ and $\sqcap$, a dataset was constructed from the rules of the adjacent dataset used in the sequential composition experiments, and a dataset specifying best practice firewall rules was constructed to test policy compliance. Overall, the results are promising and demonstrate that the approach is practical for large policies.

Chapter 7

A Firewall Policy Algebra For OpenStack

This chapter describes a firewall policy algebra $\mathcal{FW}_{OpenStack}$ for OpenStack. $\mathcal{FW}_{OpenStack}$ is a derivation of the $\mathcal{FW}_1$ algebra defined in Chapter 5. We use the algebra to provide a uniform way to specify and reason about OpenStack host-based and network access controls. This chapter builds upon earlier research [62, 101] and is organised as follows. Section 7.1 gives an overview of the OpenStack cloud operating system. OpenStack host-based and network access control is examined in Section 7.2. In Section 7.3, OpenStack perimeter firewalls and security groups are encoded, and a firewall policy algebra $\mathcal{FW}_{OpenStack}$ is derived from the $\mathcal{FW}_1$ model. Section 7.4 presents a case study OpenStack deployment that illustrates practical use of the algebra.

7.1 OpenStack

OpenStack [63] is an open-source cloud operating system for public and private clouds. It provides Infrastructure-as-a-Service (IaaS) through a collection of interrelated projects/services. These services are used to manage pools of compute, storage, and networking resources throughout a datacenter [63]. For example, the *Nova* compute service manages the life-cycle of compute instances in an OpenStack deployment. The *Glance* image service stores and retrieves virtual machine disk images, and is used by Nova during instance provisioning. The *Neutron* networking service enables Network-Connectivity-as-a-Service for Nova [64]. In this chapter, we focus on the OpenStack networking service, Neutron.

7.1.1 Motivation

The cloud computing paradigm has become widely adopted, with applications ranging from research to enterprise. However, for end-users, administrators and providers alike, there are security challenges. Managing the host-based and network access controls within and across cloud deployments is complex and error-prone. Cross-tenant accesses are often necessary for service-to-service communication, and the environment is highly dynamic due to platform and service migration. Multiple access control policies of varying types are required for a deployment, and a misconfigured policy may permit accesses that were intended to be denied or vice-versa. We regard the specification of an OpenStack access control policy as a process that evolves. Threats to, and access requirements for, resources within a cloud do not usually remain static, and over time, a policy or distributed policy configuration may be updated on an ad-hoc basis possibly by multiple specifiers/administrators. This can be problematic and may introduce anomalies, whereby the intended semantics of the specified access controls become ambiguous.

7.2 OpenStack Firewall Policies

The OpenStack Networking service, Neutron, is a standalone API-centric networking service. In general, the OpenStack networking configuration for a deployment will be segmented into four physical data center networks, as part of three distinct security domains. The *Management* network is used for inter-communication between OpenStack services, and is considered the *Management Security Domain*. The *API* network is used by tenants to access OpenStack APIs, and is considered the *Public Security Domain*. The *External* network, also in the Public Security Domain, is used by virtual machines (VMs) for Internet access, while the *Guest* network, used for VM instance-to-instance communication, is considered the *Guest Security Domain*. In this chapter, we focus on host-based and network access controls within the Guest Security Domain. These controls consist of perimeter firewall policies and Neutron security groups.

7.2.1 Perimeter Firewall Policies

Firewall-as-a-Service (FWaaS) adds perimeter firewall management to an OpenStack project by filtering traffic at the Neutron router. It is implemented as a sequence of iptables rules, where a default-deny policy is enforced. One firewall policy is supported per project, whereby the policy is applied to all networking routers within the project [64]. FWaaS is currently considered an experimental feature of OpenStack Networking [15]. A FWaaS rule can be constructed using the OpenStack Neutron command-line client:

```
neutron firewall-rule-create
    --source-ip-address $s
    --source-port $sprt
    --destination-ip-address $d
    --destination-port $dprt
    --protocol $p
    --action $act
```

The source ($\$s$) and destination ($\$d$) IP fields may be given as a single IP address/an IP address block (CIDR), the source ($\$sprt$) and destination ($\$dprt$) ports may be specified as single port values or ranges. The protocol ($\$p$) field may be given as TCP/UDP/ICMP/Any, and the action field ($\$act$) specifying the access decision, may be given as allow/deny.

7.2.2 Security Group Policies

A security group policy is a container for IP filter rules. Traditionally, security group capabilities were managed as part of the OpenStack compute service, Nova, and were instance-based. In Neutron, security groups are virtual interface port based. When utilizing Neutron as part of an OpenStack deployment, best practice [15] stipulates that security group capabilities be disabled in Nova, due to both possible conflicting policies, and also the more powerful capabilities of Neutron security groups. Security group rules allow administrators/tenants the ability to specify the type and direction of traffic that is allowed to pass through a virtual interface port. When a port is created in Neutron it is associated with a security group. If no security group is specified, a 'default' security group is assigned. This default group will drop all ingress traffic except that traffic originating from the default group, and allow all egress [15]. Rules may be added/removed to/from any security group by a tenant/administrator to change the default behaviour. A security group rule can be constructed using the Open-

Stack Neutron command-line client:

```
neutron security-group-rule-create
    --direction $dir
    --port-range-min $min
    --port-range-max $max
    --remote-ip-prefix $rsrc
    --remote-group-id $rsrc
    --protocol $p
    SECURITY_GROUP
```

The direction field ($\$dir$) is specified as ingress/egress. The remote source ($\$rsrc$) may be given as an IP address/an IP address block (CIDR) using the remote IP prefix, or as a Neutron security group using the remote group id. Selecting a security group as the remote source will allow access to/from any instance in that security group, depending on the value specified in the direction field. The destination port ($\$min, \$max$) may be given as a single port/port range. The protocol field ($\$p$) may be specified as TCP/UDP/ICMP/Any. Additionally, when specifying a rule with an ICMP protocol, given that ICMP does not support ports, the specific ICMP Type/Code may be given in place of the destination port. The `SECURITY_GROUP` attribute specifies the security group that this rule applies to. Note that there are additional parameters that may be provided as command-line arguments, however the above are sufficient for our purposes.

7.3 The OpenStack Policy Model

In this section, a derivation of the $\mathcal{FW}_1$ algebra is used to encode the network and host-based access controls available in OpenStack.

FWaaS Filter Conditions. A FWaaS filter condition is a five-tuple, $(s, sprt, d, dprt, p) \in FC_{FWaaS}$, representing network traffic originating from source IP range s, with source port range $sprt$, destined for destination IP range d, with destination port range $dprt$, using protocols p. Let FC_{FWaaS} define the set of all FWaaS filter conditions:

$$FC_{FWaaS} == \delta[IP_{Spec}, \delta[Prt_{Spec}, \delta[IP_{Spec}, \delta[Prt_{Spec}, Protocol]]]]$$

From this definition it follows that FC_{FWaaS} is a five-tuple iptables filter condition

(FC). From Lemma 5.1.5, we have that FC_{FWaaS} is a lattice.

Security Group Filter Conditions. A security group filter condition defines the packets to be accepted relative to the members of the security group. Let FC_{SG} define the set of all security group filter conditions:

$$FC_{SG} == \delta[IP_{Spec}, \delta[Dir, \delta[IP_{Spec}, \delta[Prt_{Spec}, Protocol]]]]$$

The filter condition attribute Dir, specifies direction-oriented filtering as $\mathsf{ingress}$ or egress. A security group filter condition $(sgm, dir, rsrc, dprt, p) \in FC_{SG}$ specifies that for all members sgm of the security group to which the rule belongs, network traffic is permitted in direction dir to/from remote-source $rsrc$ (depending on direction dir), to destination ports $dprt$, using protocols p.

Filter Condition Mapping. A security group filter condition $f \in FC_{SG}$ can be mapped to the FWaaS filter condition $\mathcal{F}_{sg}(f)$ that it matches, whereby:

$$\begin{array}{|l}
\mathcal{F}_{sg} : FC_{SG} \to FC_{FWaaS} \\
\hline
\forall\, sgm, rsrc : IP_{Spec};\ prt : Prt_{Spec};\ p : Protocol \bullet \\
\qquad \mathcal{F}_{sg}(sgm, \mathsf{egress}, rsrc, prt, p) \\
\qquad\qquad = (sgm, \{\{[0 \mathrel{..} \mathsf{maxPrt}]\}\}, rsrc, prt, p) \wedge \\
\qquad \mathcal{F}_{sg}(sgm, \mathsf{ingress}, rsrc, prt, p) \\
\qquad\qquad = (rsrc, \{\{[0 \mathrel{..} \mathsf{maxPrt}]\}\}, sgm, prt, p)
\end{array}$$

If the direction attribute is $\mathsf{ingress}$ then the filter condition constrains packets coming from the remote source and destined to the members of the security group; if direction attribute is egress then the filter condition constrains packets coming from members of the security group (source) and destined to the remote resource.

7.3.1 The OpenStack Policy Algebra

A firewall policy defines the filter conditions that may be allowed or denied by a firewall. Let $Policy_{FWaaS}$ define the set of all OpenStack firewall policies, whereby:

$$Policy_{FWaaS} == \{A, D : \alpha[FC_{FWaaS}] \mid \forall\, a : A;\ d : D \bullet a \mid_{FC_{FWaaS}} d\}$$

A firewall policy $(A, D) \in Policy_{FWaaS}$ defines a policy as a pair of adjacency-free sets of filter conditions under the duplet adjacency ordering, whereby a filter condition $f \in A$ should be allowed by the OpenStack firewall, while a filter condition $f \in D$ should be denied.

OpenStack Policy Ordering. The safe replacement [56, 57, 77] ordering for OpenStack firewall policies is defined as follows.

$$
\begin{array}{|l}
\hline
\mathcal{FW}_{\mathit{OpenStack}} \\
\bot, \top : \mathit{Policy}_{FWaaS} \\
\_ \sqsubseteq \_ : \mathit{Policy}_{FWaaS} \leftrightarrow \mathit{Policy}_{FWaaS} \\
\_ \sqcap \_, \\
\_ \sqcup \_ : \mathit{Policy}_{FWaaS} \times \mathit{Policy}_{FWaaS} \rightarrow \mathit{Policy}_{FWaaS} \\
\hline
\bot = (\emptyset, \lceil FC_{FWaaS} \rceil) \land \top = (\lceil FC_{FWaaS} \rceil, \emptyset) \\
\forall P, Q : \mathit{Policy}_{FWaaS} \bullet \\
\quad P \sqsubseteq Q \Leftrightarrow ((\mathit{allow}\, P \leq \mathit{allow}\, Q) \land \\
\qquad\qquad (\mathit{deny}\, Q \leq \mathit{deny}\, P)) \land \\
\quad P \sqcap Q = (\mathit{allow}\, P \otimes \mathit{allow}\, Q, \\
\qquad\qquad \mathit{deny}\, P \oplus \mathit{deny}\, Q) \land \\
\quad P \sqcup Q = (\mathit{allow}\, P \oplus \mathit{allow}\, Q, \\
\qquad\qquad \mathit{deny}\, P \otimes \mathit{deny}\, Q) \\
\hline
\end{array}
$$

The definitions for ordering and composition operations in the $\mathcal{FW}_{\mathit{OpenStack}}$ algebra are the same as those defined for the $\mathcal{FW}_1$ policy algebra in Chapter 5, Section 5.3, and therefore, $\mathcal{FW}_{\mathit{OpenStack}}$ is a lattice.

The lattice of policies Policy_{FWaaS} provides us with an algebra for constructing and interpreting OpenStack firewall polices. Definitions for policy construction, policy negation, policy sequential composition and firewall rule interpretation are assumed for Policy_{FWaaS} policies. These definitions are similar to those defined in Chapter 5 for the $\mathcal{FW}_1$ policy algebra.

FWaaS Rules. A FWaaS rule defines an action (allow or deny) for a given filter condition. Let *Rule* define the set of all FWaaS rules whereby:

$$
\begin{array}{l}
\mathit{Rule} ::= \mathsf{allow}\, \langle\!\langle FC_{FWaaS} \rangle\!\rangle \mid \\
\qquad \mathsf{deny}\, \langle\!\langle FC_{FWaaS} \rangle\!\rangle
\end{array}
$$

A FWaaS firewall policy is defined as a sequence of rules $\langle r_1, r_2, \ldots, r_n \rangle$ for $r_i \in$ *Rule*, and is encoded in the $\mathcal{FW}_{\mathit{OpenStack}}$ policy algebra as $\mathcal{I}(r_1) \mathbin{⨾} \mathcal{I}(r_2) \mathbin{⨾} \ldots \mathbin{⨾} \mathcal{I}(r_n)$.

Security Group Rules. A security group rule is simply an allow action on its filter condition:

$$
\begin{array}{|l}
\mathcal{I}^s : FC_{SG} \rightarrow Policy_{FWaaS} \\
\hline
\forall f : FC_{SG} \bullet \\
\qquad \mathcal{I}^s\,(f) = (\mathsf{Allow}\,(\mathcal{F}_{sg}\,(f)))
\end{array}
$$

A security group policy is written as a sequence of security group rules $\langle r_1, r_2, \ldots, r_n \rangle$ where each $r_i \in FC_{SG}$ and is encoded in the policy algebra as $\mathcal{I}^s\,(r_1) \mathbin{⨟} \mathcal{I}^s\,(r_2) \mathbin{⨟} \ldots \mathbin{⨟} \mathcal{I}^s\,(r_n)$. Note that in this encoding it is assumed that each rule in the original policy has the same membership, that is, $group(r_i) = group(r_j)$ for all rules r_i and r_j in the policy where $group(r)$ gives the group in the rule/filter condition r.

Policy Projection. The projection operators $@^u$ and $@^d$ filter a policy by a set of IP ranges. Firstly, let $\mathcal{S}(S)$ give the cover-set for all filter conditions that have a source IP in some member $s \in S$, and similarly $\mathcal{D}(D)$ gives the cover-set for all filter conditions with a destination IP address in some member $d \in D$.

$$
\begin{array}{|l}
\mathcal{S}, \mathcal{D} : IP_{Spec} \rightarrow \alpha[FC_{FWaaS}] \\
\hline
\forall\, S, D : IP_{Spec} \bullet \\
\qquad \mathcal{S}(S) = \lceil \delta[S, \delta[Prt_{Spec}, \delta[IP_{Spec}, \delta[Prt_{Spec}, \delta[Protocol]]]]] \rceil \wedge \\
\qquad \mathcal{D}(D) = \lceil \delta[IP_{Spec}, \delta[Prt_{Spec}, \delta[D, \delta[Prt_{Spec}, \delta[Protocol]]]]] \rceil
\end{array}
$$

For a policy P and a set of adjacency-free IP ranges S, $P @^u S$ is the upstream projection of P, and consists of the allow and deny filter conditions from P where each filter condition has a source IP in some member of S. Similarly, $P @^d S$ is the downstream projection of P, it consists of the allow and deny filter conditions from P whereby each filter conditions has as a destination IP in some member of S.

$$
\begin{array}{|l}
\_ @^u \_, \\
\_ @^d \_ : Policy_{FWaaS} \times IP_{Spec} \rightarrow Policy_{FWaaS} \\
\hline
\forall\, P : Policy_{FWaaS};\ S : IP_{Spec} \bullet \\
\qquad P \mathbin{@^u} S = (allow(P) \otimes \mathcal{S}(S), deny(P) \otimes \mathcal{S}(S)) \wedge \\
\qquad P \mathbin{@^d} S = (allow(P) \otimes \mathcal{D}(S), deny(P) \otimes \mathcal{D}(S))
\end{array}
$$

7.4 Reasoning About OpenStack Firewall Policies

In this section, a case study OpenStack deployment is presented that illustrates the practical use of the algebra $\mathcal{FW}_{OpenStack}$. Consider, a company that has migrated platforms and services to an on-premises OpenStack deployment. The deployment hosts both a development and a production cloud for the Web-service provided by the company and the code revision control systems for the Web-service. These two private clouds are defined as independent OpenStack projects/tenants as depicted in Figure 7.1[1].

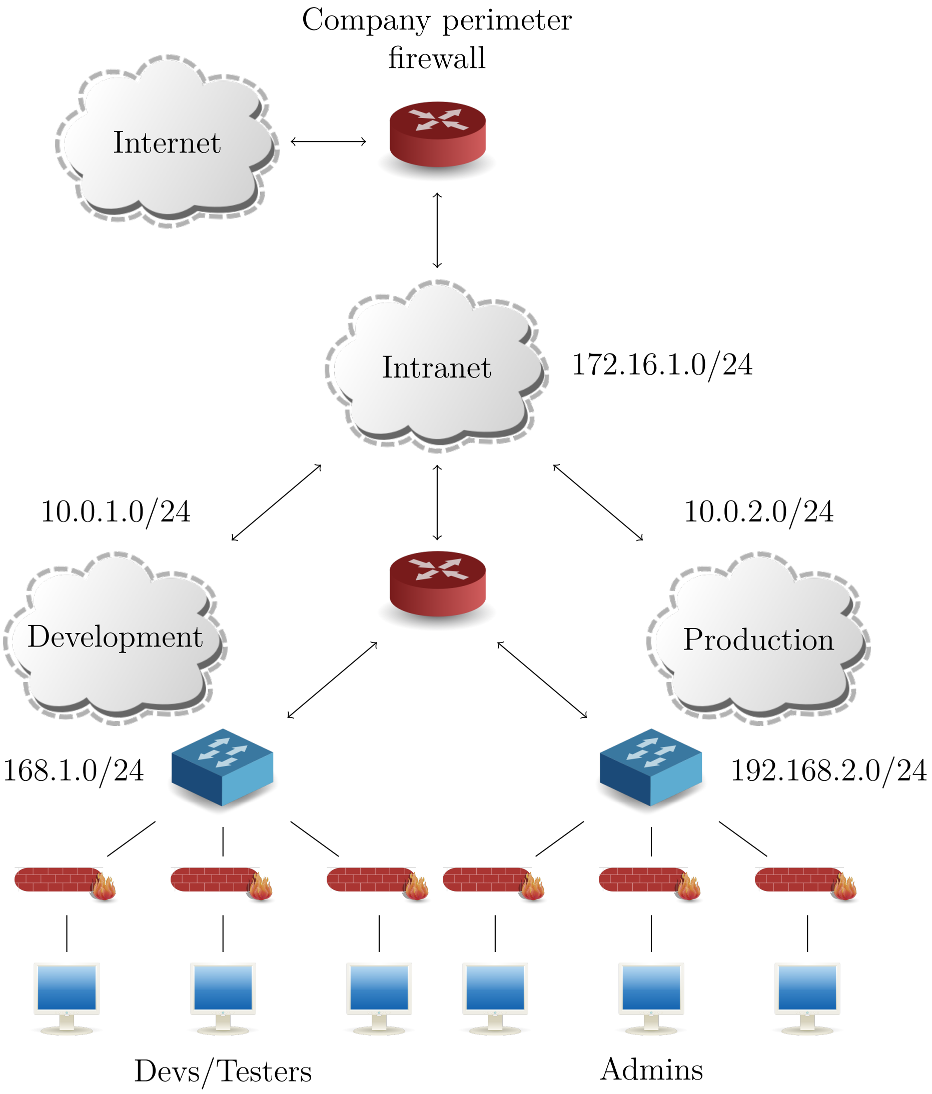


Figure 7.1: External network architecture

[1] All clip art used in this dissertation has been sourced from [106].

System Administrator Bob manages the network access controls for both the development and production clouds. The architecture for each cloud is depicted in Figure 7.2.

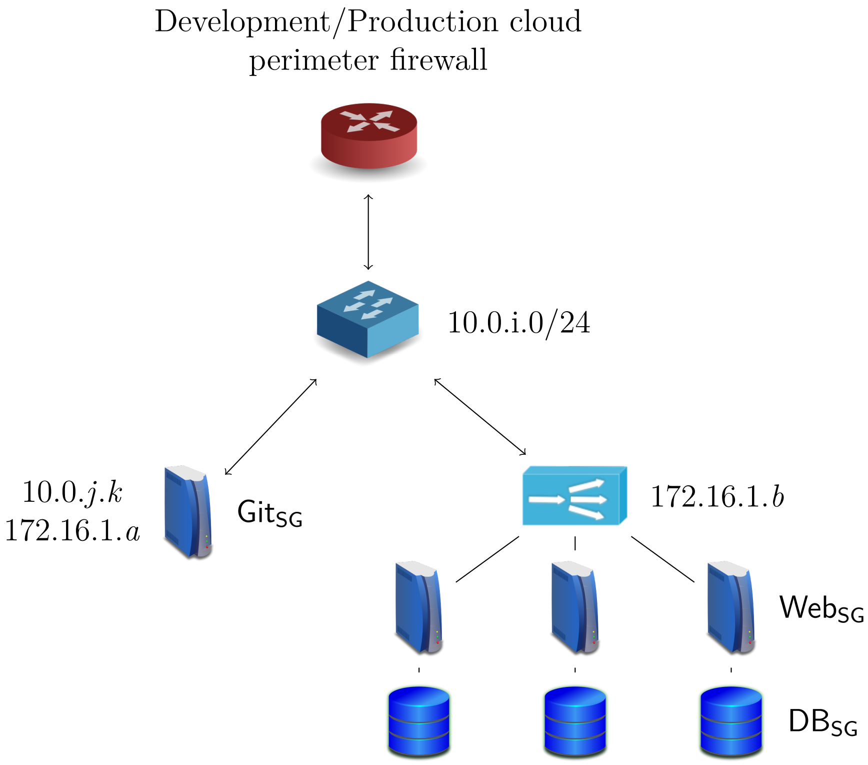

Development/Production cloud
perimeter firewall
10.0.i.0/24
10.0.j.k
172.16.1.a
$\mathsf{Git}_{\mathsf{SG}}$
172.16.1.b
$\mathsf{Web}_{\mathsf{SG}}$
$\mathsf{DB}_{\mathsf{SG}}$

Figure 7.2: Guest network architecture

7.4.1 Reasoning About Security Groups

Bob creates a security group policy $\mathsf{Git}_{\mathsf{SG}}$ within the development cloud to manage the type of traffic permitted to/from the code revision control server. $\mathsf{sgm}_1 \in IP_{Spec}$ denotes the set of adjacency-free IP ranges for the members of this security group. Bob begins to add rules $\mathsf{git}_1, \mathsf{git}_2$, for ICMP ping for each member of the dev/tester subnet to allow developers and testers to ping the Git server in the development cloud. Recall that when specifying a security group rule using the OpenStack Neutron CLI, that the ICMP Type/Code is given in place of a port range, when required. The function $\mathcal{M}(\mathsf{rule})$ maps a rule written in OpenStack firewall rule syntax to $\mathcal{FW}_1$ algebra syntax.

$$\begin{aligned} \mathsf{git}_1 == {} & \texttt{neutron security-group-rule-create --direction}\ \mathsf{ingress}\ \backslash \\ & \quad \texttt{--port-range-min}\ 8\ \texttt{--port-range-max}\ 0\ \backslash \\ & \quad \texttt{--remote-ip-prefix}\ 192.168.1.3\ \texttt{--protocol}\ \mathsf{icmp}\ \mathsf{Git}_{\mathsf{SG1}} \end{aligned}$$

$$\begin{aligned} \mathsf{git}_2 == {} & \texttt{neutron security-group-rule-create --direction}\ \mathsf{ingress}\ \backslash \\ & \quad \texttt{--port-range-min}\ 8\ \texttt{--port-range-max}\ 0\ \backslash \\ & \quad \texttt{--remote-ip-prefix}\ 192.168.1.4\ \texttt{--protocol}\ \mathsf{icmp}\ \mathsf{Git}_{\mathsf{SG1}} \end{aligned}$$

$$\mathsf{Git}_{\mathsf{SG1}} == \mathcal{I}^{\mathsf{s}}\,(\mathcal{M}(\mathsf{git}_1)) \fatsemi \mathcal{I}^{\mathsf{s}}\,(\mathcal{M}(\mathsf{git}_2))$$

Bob finds this tedious and decides to simply add a rule git_3 that allows all inbound ICMP traffic from the dev/tester subnet.

$$\begin{aligned} \mathsf{git}_3 == {} & \texttt{neutron security-group-rule-create --direction}\ \mathsf{ingress}\ \backslash \\ & \quad \texttt{--remote-ip-prefix}\ 192.168.1.0/24\ \texttt{--protocol}\ \mathsf{icmp}\ \mathsf{Git}_{\mathsf{SG2}} \end{aligned}$$

$$\mathsf{Git}_{\mathsf{SG2}} == \mathsf{Git}_{\mathsf{SG1}} \fatsemi \mathcal{I}^{\mathsf{s}}\,(\mathcal{M}(\mathsf{git}_3))$$

In doing so, however, git_1 and git_2 are now redundant to git_3, $(\mathcal{I}^{s}\,(\mathcal{M}(\mathsf{git}_1)) \fatsemi \mathcal{I}^{\mathsf{s}}\,(\mathcal{M}(\mathsf{git}_2))) \fatsemi \mathcal{I}^{\mathsf{s}}\,(\mathcal{M}(\mathsf{git}_3)) = \mathcal{I}^{\mathsf{s}}\,(\mathcal{M}(\mathsf{git}_3))$.

Rule git_4 is introduced to allow all developers and testers access to the code revision control system (Git) in the development cloud, where:

$$\begin{aligned} \mathsf{git}_4 == {} & \texttt{neutron security-group-rule-create --direction}\ \mathsf{ingress}\ \backslash \\ & \quad \texttt{--port-range-min}\ 9418\ \texttt{--port-range-max}\ 9418\ \backslash \\ & \quad \texttt{--remote-ip-prefix}\ 192.168.1.0/24\ \texttt{--protocol}\ \mathsf{tcp}\ \mathsf{Git}_{\mathsf{SG3}} \end{aligned}$$

$$\mathsf{Git}_{\mathsf{SG3}} == \mathsf{Git}_{\mathsf{SG2}} \fatsemi \mathcal{I}^{\mathsf{s}}\,(\mathcal{M}(\mathsf{git}_4))$$

Cross-tenant access is required for source code replication, therefore Bob must ensure that rsync via SSH is permitted from the Git server in the development cloud to the Git server in the production cloud. To do so, he introduces rule git_5, where:

$$\begin{aligned} \mathsf{git}_5 == {} & \texttt{neutron security-group-rule-create --direction}\ \mathsf{egress}\ \backslash \\ & \quad \texttt{--port-range-min}\ 22\ \texttt{--port-range-max}\ 22\ \backslash \\ & \quad \texttt{--remote-ip-prefix}\ 172.16.1.7\ \texttt{--protocol}\ \mathsf{tcp}\ \mathsf{Git}_{\mathsf{SG}} \end{aligned}$$

$$\mathsf{Git}_{\mathsf{SG}} == \mathsf{Git}_{\mathsf{SG3}} \fatsemi \mathcal{I}^{\mathsf{s}}\,(\mathcal{M}(\mathsf{git}_5))$$

Bob creates the security group policy $\mathsf{Web}_{\mathsf{SG}}$ to manage the accesses to/from the Web-service load balancer in the development cloud. Let $\mathsf{sgm}_2 \in \mathit{IP}_{\mathit{Spec}}$ denote the set of adjacency-free IP ranges for the members of this security group. Bob adds rules to allow HTTP traffic from the developers and testers (web_1), and from the administrators (web_2), where:

$$\begin{aligned}
\mathsf{web}_1 == {} & \texttt{neutron security-group-rule-create --direction } \mathsf{ingress} \backslash \\
& \texttt{--port-range-min } \mathsf{80} \texttt{ --port-range-max } \mathsf{80} \backslash \\
& \texttt{--remote-ip-prefix } \mathsf{192.168.1.0/24} \texttt{ --protocol } \mathsf{tcp}\ \mathsf{Web}_{\mathsf{SG}}
\end{aligned}$$

$$\begin{aligned}
\mathsf{web}_2 == {} & \texttt{neutron security-group-rule-create --direction } \mathsf{ingress} \backslash \\
& \texttt{--port-range-min } \mathsf{80} \texttt{ --port-range-max } \mathsf{80} \backslash \\
& \texttt{--remote-ip-prefix } \mathsf{192.168.2.0/24} \texttt{ --protocol } \mathsf{tcp}\ \mathsf{Web}_{\mathsf{SG}}
\end{aligned}$$

$$\mathsf{Web}_{\mathsf{SG}} == \mathcal{I}^{\mathsf{s}}\left(\mathcal{M}(\mathsf{web}_1)\right) \fatsemi \mathcal{I}^{\mathsf{s}}\left(\mathcal{M}(\mathsf{web}_2)\right)$$

The security group policy $\mathsf{DB}_{\mathsf{SG}}$ is created by Bob to manage accesses to/from the Web-service data tier in the development cloud. The literal $\mathsf{sgm}_3 \in \mathit{IP}_{\mathit{Spec}}$ denotes the set of adjacency-free IP ranges for the members of this security group. He adds the rule db_1 to allow all inbound traffic from members of the $\mathsf{Web}_{\mathsf{SG}}$ group to MySQL port 3306.

$$\begin{aligned}
\mathsf{db}_1 == {} & \texttt{neutron security-group-rule-create --direction } \mathsf{ingress} \backslash \\
& \texttt{--port-range-min } \mathsf{3306} \texttt{ --port-range-max } \mathsf{3306} \backslash \\
& \texttt{--remote-group-id } \mathsf{sgm}_2 \texttt{ --protocol } \mathsf{tcp}\ \mathsf{DB}_{\mathsf{SG}}
\end{aligned}$$

$$\mathsf{DB}_{\mathsf{SG}} == \mathcal{I}^{\mathsf{s}}\left(\mathcal{M}(\mathsf{db}_1)\right)$$

The development cloud enforces the security group policies $\mathsf{Git}_{\mathsf{SG}}, \mathsf{Web}_{\mathsf{SG}}$ and $\mathsf{DB}_{\mathsf{SG}}$. Recall that a security group policy is a container for allow rules managing the access to/from the security group, and that each security group policy in a cloud deployment is enforced independent of the other security group policies. Thus, the overall security group policy is the union of the individual policies:

$$\mathsf{Dev}_{\mathsf{SG}} == \mathsf{Git}_{\mathsf{SG}} \sqcup \mathsf{Web}_{\mathsf{SG}} \sqcup \mathsf{DB}_{\mathsf{SG}}$$

7.4.2 Reasoning About FWaaS Firewalls

As part of the configuration, Bob must also ensure the appropriate traffic traverses the perimeter firewall at the edge router of the development cloud. He therefore enforces FWaaS policy $\mathsf{Dev_{FW1}}$ and begins to add some rules.

$$\begin{aligned}\mathsf{dev}_1 == {} & \texttt{neutron firewall-rule-create --source-ip-address 172.16.1.5} \ \backslash \\ & \texttt{--destination-ip-address 172.16.1.7 --destination-port 22} \ \backslash \\ & \texttt{--protocol tcp --action deny}\end{aligned}$$

$$\begin{aligned}\mathsf{dev}_2 == {} & \texttt{neutron firewall-rule-create --source-ip-address 172.16.1.5} \ \backslash \\ & \texttt{--destination-ip-address 172.16.1.7 --destination-port 22} \ \backslash \\ & \texttt{--protocol tcp --action allow}\end{aligned}$$

$$\mathsf{Dev_{FW1}} == \mathcal{I}\left(\mathcal{M}(\mathsf{dev}_1)\right) \fatsemi \mathcal{I}\left(\mathcal{M}(\mathsf{dev}_2)\right)$$

Bob mistakenly introduces rule dev_1, creating a shadowing anomaly of rule dev_2, that is, $\textbf{not}\,(\mathcal{I}\,(\mathsf{dev}_1)) \fatsemi \mathcal{I}\,(\mathsf{dev}_2) = \mathcal{I}\,(\mathsf{dev}_2)$, whereby the logical traffic flow is broken between the code revision control systems in the development and production clouds.

Rule dev_3 ensures the developers and testers are permitted to ping the Git server in the development cloud.

$$\begin{aligned}\mathsf{dev}_3 == {} & \texttt{neutron firewall-rule-create --source-ip-address} \ \ \backslash \\ & \texttt{192.168.1.0/24 --destination-ip-address 172.16.1.5} \ \backslash \\ & \texttt{--protocol icmp --action allow}\end{aligned}$$

$$\mathsf{Dev_{FW2}} == \mathsf{Dev_{FW1}} \fatsemi \mathcal{I}\left(\mathcal{M}(\mathsf{dev}_3)\right)$$

The rule dev_4 permits unwanted Telnet traffic to the Git server, thereby allowing spurious traffic into the development cloud, where:

$$\begin{aligned}\mathsf{dev}_4 == {} & \texttt{neutron firewall-rule-create --source-ip-address} \ \ \backslash \\ & \texttt{192.168.1.0/24 --destination-ip-address 172.16.1.5} \ \backslash \\ & \texttt{--destination-port 21 --protocol tcp --action allow}\end{aligned}$$

$$\mathsf{Dev_{FW}} == \mathsf{Dev_{FW2}} \fatsemi \mathcal{I}\left(\mathcal{M}(\mathsf{dev}_4)\right)$$

Recall that all traffic entering the development cloud must traverse the development cloud perimeter firewall, and that the policy defining the complete set of internal accesses for the cloud is given as $\mathsf{Dev_{SG}}$. Thus, the policy constraining accesses for traffic from upstream firewall $\mathsf{Dev_{FW}}$ to downstream composite security groups $\mathsf{Dev_{SG}}$ is calculated as:

$$\begin{aligned}\mathsf{Pol^{IN}_{DEV}} == {} & \mathsf{Dev_{SG}} \mathbin{@^d} (\mathsf{sgm}_1 \oplus \mathsf{sgm}_2 \oplus \mathsf{sgm}_3 \oplus \mathsf{floatIP}) \fatsemi \\ & \qquad \mathsf{Dev_{FW}} \mathbin{@^d} (\mathsf{sgm}_1 \oplus \mathsf{sgm}_2 \oplus \mathsf{sgm}_3 \oplus \mathsf{floatIP})\end{aligned}$$

where $\oplus$ is the join operator for sets of adjacency-free IP ranges, and $\mathsf{floatIP} \in IP_{Spec}$ is the set of floating IP addresses, given as the set of adjacency-free IP ranges used by the tenant. Note that all security group members are included since (at the time of writing) FWaaS applies to all routers within a tenant, not just to the perimeter.

The Production Cloud Firewall. Bob must also ensure the configuration is correct for the production cloud perimeter firewall $\mathsf{Prod_{FW1}}$.

$$\mathsf{prod}_1 == \texttt{neutron firewall-rule-create --source-ip-address } \mathsf{\$malIP} \ \backslash \\ \texttt{--destination-port } \mathsf{80}\texttt{--protocol } \mathsf{tcp}\ \texttt{--action } \mathsf{deny}$$

$$\mathsf{Prod_{FW1}} == \mathcal{I}\,(\mathcal{M}(\mathsf{prod}_1))$$

Rule prod_1 denies HTTP traffic to the production Web servers from the known malicious IP range $\mathsf{\$malIP}$.

$$\mathsf{prod}_2 == \texttt{neutron firewall-rule-create --source-ip-address} \ \backslash \\ \mathsf{172.16.1.5}\ \texttt{--destination-ip-address } \mathsf{172.16.1.7} \ \backslash \\ \texttt{--destination-port } \mathsf{22}\ \texttt{--protocol } \mathsf{tcp}\ \texttt{--action } \mathsf{allow}$$

$$\mathsf{Prod_{FW2}} == \mathsf{Prod_{FW1}} \fatsemi \mathcal{I}\,(\mathcal{M}(\mathsf{prod}_2))$$

Rule prod_2 is introduced by Bob to ensure the rsync via SSH between the code revision control systems in the development and production clouds is permitted. However, the problematic rule dev_1 in the development cloud firewall, has also caused a shadowing anomaly between the two perimeter firewalls ($\mathbf{not}\,(\mathcal{I}\,(\mathsf{dev}_1)) \fatsemi \mathcal{I}\,(\mathsf{prod}_2) = \mathcal{I}\,(\mathsf{prod}_2)$), whereby the the development cloud firewall is denying traffic from the development Git server to the Git server in the production cloud, while the rule prod_2 of the production cloud firewall is permitting the traffic.

$$\mathsf{prod}_3 == \texttt{neutron firewall-rule-create --destination-port } \mathsf{80} \ \backslash \\ \texttt{--protocol } \mathsf{tcp}\ \texttt{--action } \mathsf{allow}$$

$$\mathsf{Prod_{FW}} == \mathsf{Prod_{FW2}} \fatsemi \mathcal{I}\,(\mathcal{M}(\mathsf{prod}_3))$$

Bob introduces prod_3 to allow all other HTTP traffic to the Web-service load balancer. The rule prod_3 is generalised by the rule at prod_1, as prod_1 partially shadows prod_3.

Production Cloud Git Security Group. In the production cloud, Bob creates security group $\mathsf{Git2}_{\mathsf{SG1}}$ to manage the type of traffic permitted to/from the production code revision control server. The literal sgm_4 denotes the set of IP addresses for the members of this security group. Rule $\mathsf{git}^{\mathsf{p}}_1$ is introduced to allow all developers and testers access to the code revision control system (Git) in the production cloud, where:

$$
\begin{aligned}
\mathsf{git}^{\mathsf{p}}_1 == \; & \texttt{neutron security-group-rule-create --direction } \mathsf{ingress} \; \backslash \\
& \quad \texttt{--port-range-min } \mathsf{9418} \; \texttt{--port-range-max } \mathsf{9418} \; \backslash \\
& \quad \texttt{--remote-ip-prefix } \mathsf{192.168.1.0/24} \; \texttt{--protocol } \mathsf{tcp} \; \mathsf{Git2}_{\mathsf{SG1}} \\
\mathsf{Git2}_{\mathsf{SG1}} == \; & \mathcal{I}^{\mathsf{s}}\,(\mathcal{M}(\mathsf{git}^{\mathsf{p}}_1))
\end{aligned}
$$

The rule $\mathsf{git}^{\mathsf{p}}_2$, intended to ensure rsync via SSH between the code revision control systems in the development and production clouds is permitted, is inter-shadowed by the upstream rule dev_1 of the development cloud perimeter firewall, where:

$$
\begin{aligned}
\mathsf{git}^{\mathsf{p}}_2 == \; & \texttt{neutron security-group-rule-create --direction } \mathsf{ingress} \; \backslash \\
& \quad \texttt{--port-range-min } \mathsf{22} \; \texttt{--port-range-max } \mathsf{22} \; \backslash \\
& \quad \texttt{--remote-ip-prefix } \mathsf{172.16.1.5} \; \texttt{--protocol } \mathsf{tcp} \; \mathsf{Git2}_{\mathsf{SG}} \\
\mathsf{Git2}_{\mathsf{SG}} == \; & \mathsf{Git2}_{\mathsf{SG1}} \mathbin{;} \mathcal{I}^{\mathsf{s}}\,(\mathcal{M}(\mathsf{git}^{\mathsf{p}}_2))
\end{aligned}
$$

Recall, $\mathsf{Pol}^{\mathsf{IN}}_{\mathsf{DEV}}$ provides a policy about traffic traversing the perimeter firewall for the development cloud to the composite security group. A similar policy can be given for the traffic traversing the perimeter firewall of the production cloud destined to the security group within the production cloud. Further definitions can be given about policies on traffic leaving the respective clouds. These in turn can be composed to give a policy that is effectively about the rsync via SSH for the code revision systems in the development cloud to the production cloud.

7.4.3 OpenStack Firewall Policy Anomalies

In this section, we use the definitions of [5, 6] to describe potential intra- and inter-anomalies between rules/policies in Neutron security group and FWaaS policy configurations.

Security Group Intra-anomalies. An intra-redundancy anomaly may occur in a security group, given that we may encounter two or more rules that are equivalent, or one or more rules that are filtering a subset of the network traffic filtered by another rule in the policy. We may also encounter rules that cannot match any traffic based on either the source/destination IP addresses, resulting in an intra-irrelevance anomaly.

Security Group (to Security Group) Inter-anomalies. We observe the possibility of spurious network traffic between different security groups; regarded as an implicit inter-spuriousness anomaly, from an upstream security group P permitting traffic to a downstream security group Q, whereby Q is missing a rule permitting the same traffic (and is therefore implicitly denying it). An implicit inter-shadowing anomaly may occur between security groups; depending on, for example, if upstream security group P was acting as a proxy for traffic for a downstream security group Q, and P is missing a rule (implicitly denying) traffic intended for Q, whereby Q is explicitly permitting the same traffic.

FWaaS Policy Anomalies. We observe that all the intra-anomalies described in [5] may occur between rules in a single FWaaS policy, and that all the inter-anomalies given in [6] may occur between rules across distributed FWaaS policies.

Inter-anomalies Between Security Group and FWaaS Policies. We consider an upstream FWaaS policy P and a downstream security group policy Q. We observe the possibility of inter-spuriousness; given that P may be configured in such a way to allow unwanted traffic directed towards members of the security group constrained by Q. We note also the possibility of inter-shadowing, whereby P may be explicitly denying traffic that Q has been configured to accept.

When considering an upstream security group policy P and a downstream FWaaS policy Q, we note the possibility of inter-spuriousness, and an implicit inter-shadowing anomaly; depending on, again for example, if P was acting as a proxy for traffic for Q, and P is missing a rule (implicitly denying) traffic intended for the downstream FWaaS firewall Q, whereby Q is explicitly permitting the same traffic. We observe the possibility of an inter-correlation anomaly between rules in an upstream FWaaS policy P and a downstream security group Q, or also between an upstream security group policy P and a downstream FWaaS policy Q. This anomaly occurs independent of target action/access decision, where a rule in the upstream policy is filtering some of the packets filtered by a rule in the downstream policy, and the rule under question in the downstream policy is filtering some of the packets filtered by the rule in the upstream policy. The inter-correlation anomaly may cause further anomalies such as inter-spuriousness and inter-shadowing, depending on the target actions of the correlated rules.

7.5 Discussion

In this chapter, a policy algebra $\mathcal{FW}_{OpenStack}$ is defined in which OpenStack firewall policies can be specified and reasoned about. The set of policies form a lattice under safe replacement and this enables consistent operators for safe composition to be defined. Policies in this lattice are anomaly-free by construction, and thus, composition under glb and lub operators preserves anomaly-freedom. The algebra $\mathcal{FW}_{OpenStack}$ is a derivation of the $\mathcal{FW}_1$ policy algebra developed in Chapter 5, and provides a formal interpretation of the host-based and network access controls in OpenStack. In particular, it gives a meaning for OpenStack security group policies and perimeter firewalls. This provides us with a uniform notation to define and reason about different kinds policies in OpenStack. For example, reasoning over combinations of perimeter firewall and security group policies to ensure that modifications are safe (replacements) and checking for heterogeneous inter-policy anomalies.

In [80], *cloud calculus* is used to capture the topology of cloud computing systems and the global firewall policy for a given configuration. The work in this chapter could extend the work in [80], given that $\mathcal{FW}_{OpenStack}$ may be used in conjunction with cloud calculus to guarantee anomaly-free dynamic firewall policy reconfiguration, whereby the ordering relation $\sqsubseteq$ may be used to provide a viable alternative for the given equivalence relation defined over 'cloud' terms for the formal verification of firewall policy preservation after a live migration.

Chapter 8

A Policy Management Framework For Android

This chapter describes a system policy algebra $Android_{Sys}$ for the Android OS, that uses the $\mathcal{FW}_1$ firewall policy algebra, in conjunction with an algebra for managing Android permissions, to construct a model of security control reconfiguration for Android firewall policies and Android permission polices. This chapter builds upon earlier research [54, 55, 101], and is organised as follows. Section 8.1 gives an overview of the Android mobile operating system. Android firewall configuration management is considered in Section 8.2. In Section 8.3, we examine the Android Permission Model, and a simple algebra $Android_{Perm}$ is developed for managing Android permissions. In Section 8.4, a threat-based model that represents catalogues of best practice for Android systems is described. In Section 8.5, a policy algebra $Android_{Sys}$ is defined, whereby Android system policies; comprising firewall and permission policies, can be specified and reasoned about. We also consider a future iteration of the MASON prototype incorporating the proposed framework.

8.1 Android

Android [141], is an open-source software framework for mobile devices based on an optimized version of the Linux kernel. It is a privilege-separated mobile operating system. The Android OS has a layered architecture, with user applications (apps) residing at the upper-most layer. The next layer hosts the application framework, containing the Java APIs accessible to app developers. The following layer hosts the native C/C++ libraries of the Android OS; these can be used by any component of the previous two layers. This layer also hosts the Android runtime, which contains an optimized variant of a Java virtual machine (VM), the

Dalvik VM. The next layer hosts the hardware abstraction layer (HAL) subsystem; this layer provides an interface for vendors to create software hooks between Android and the device hardware [143]. At the lowest layer of the hierarchy is the Linux kernel, the kernel acts as a layer of abstraction between software and hardware, and is responsible for core system services such as process management, and management of the network stack. Android uses Linux iptables as its firewall mechanism.

8.1.1 Motivation

Android runs on a variety of mobile devices, such as smartphones (for example, Google Pixel), tablet PCs (for example, Sony Xperia Z4) and embedded devices (for example, Neo-ITX and Raspberry Pi 3). In this section, we consider the Android software framework from the context of smartphones.

Modern smartphones with their processing power and the wide variety of applications (“apps”) are on a par with modern desktop environments [129]. This has resulted in smartphones being used in a variety of domains from a personal device (such as for voice, Web browsing, Email and social media) to enterprise, medical and military domains [154]. The technological advances and the usage of smartphones in a variety of domains is not without its security implications. In addition to traditional mobile phone threats, threats to desktop environments are also applicable to smartphones [29, 81, 129]. For example, Malware threats such as DroidDream [10], an Android Market Trojan used to maliciously root Android smartphones, and apps that steal user’s banking information [82, 139, 162] are on the increase [129, 139].

Smartphones may host a variety of security mechanisms such as anti-virus, app monitoring and firewalls. In practice, security mechanisms are either disabled or configured with an open-access policy [119]. Configuration of smartphone security mechanisms, for example a firewall, is typically performed by non-technical end-users. As a consequence, an effective security configuration may be hampered by a poor understanding and/or management of smartphone application requirements. Misconfiguration, may result in the failure to adequately provide smartphone app services. For example, an overly-restrictive firewall configuration may prevent normal interaction of network-based apps. An overly-permissive firewall configuration, while permitting normal operation of the app, may leave the smartphone vulnerable to attack, for example, across open ports or through malicious payloads.

Smartphones operate in mobile network environments and deploying a fixed security configuration for a global set of threats is not practical. For example, a smartphone may in one scenario be connected to an enterprise WiFi network while in another be connected to an open-access WiFi network or a 3g operator network. What may be considered a threat in one scenario may not be a threat in another. For example, a security configuration that permits a set of apps (such as gaming and social media apps) within a home network environment may not be permitted within an enterprise or teleworking environment. In a teleworking scenario, it is considered best practice to permit the use of "*a different brand of Web browser for telework*" and prohibit the use of the everyday Web browser [125]. Thus, the deployment of smartphone security configurations must be dynamic in order mitigate the relevant threats within a given scenario. Note, while smartphone apps may provide their own end-to-end security, in accordance with for example [48], it is considered best practice to also restrict access at the smartphone firewall [79, 125, 133, 152].

In this chapter, a threat-based model that represents catalogues of best practice standards for smartphones/Android is described. The firewall catalogues are smartphone/Android-centric and extend the work in [61]. New catalogues of best practice for example NIST 800-114 [125] are developed, as is a prototype firewall app called MASON, to automatically manage firewall configurations on behalf of the end-user [55]. A case study based on firewall access control demonstrates how automated firewall configuration recommendations can be made based on catalogues of countermeasures. These countermeasures are drawn from best-practice standards such as NIST 800-124, a guideline on cell phone and PDA security and NIST 800-41-rev1, a guideline on firewall security configuration. The case study also demonstrates how MASON can be extended with the $\mathcal{FW}_1$ algebra to ensure anomaly-free firewall policy reconfiguration.

8.2 Smartphone Firewall Configuration Management

The *smartphone firewall* is a security mechanism that controls traffic flow to and from network-based applications that are hosted by the smartphone itself and/or are hosted by a network of systems tethered to the smartphone in accordance with a security policy. Management of a smartphone firewall configuration involves either writing low-level command syntax via a CLI or the use of a graphical management console (for example DroidWall [163], WhipserMonitor [92] and No-Root [147]). However, smartphone firewall policy management is complex and error-prone [25, 156, 157]. Typical errors range from invalid syntax and incorrect rule ordering, to a failure to uphold a security policy due to lack of GUI-based firewall rule granularity, to errors resulting from the poor comprehension of a firewall configuration [94, 152]. An effective smartphone firewall configuration may be further hampered by the poor understanding and/or management of the overall high-level smartphone security requirements.

8.2.1 Threat Mitigation Using A Smartphone Firewall

In this section, we consider known network-based threats that may be mitigated by the smartphone firewall.

Port-based Attack Surface Mitigation. A smartphone *port-based attack surface* is the number of network accessible apps, hosted on the smartphone or on its tethered devices, in terms of ports that are available for a potential attacker to exploit. A smartphone may have a number of network accessible apps, for example RDP port 3389, VNC port 5900, SSH port 22, FTP ports 20 and 21. It is considered best practice to uninstall or disable unnecessary network apps: "*Removing or disabling unnecessary services enhances the security*" [124]. For example, a smartphone may host server-based apps such as Telnet or FTP intended for occasional use. A smartphone user may not wish to install and uninstall these kinds of apps before or after each use. As a consequence, this increases the smartphone's attack surface.

By explicitly configuring the firewall to permit access to intended app ports only, one can significantly reduce the attack surface. Consider the scenario of a remote desktop server app used to manage a smartphones files and photos. Configuring the firewall to permit only RDP traffic destined for port 3389 will reduce the attack surface from a possible 2^{16} ports to just one intended port.

IP-based Attack Surface Mitigation. A smartphone *IP-based attack surface* is the number of network-accessible apps, hosted on the smartphone or on its tethered devices, in terms of client IP address reachability that are available as a potential attacker threat vector. For example, with respect to smartphone remote management it is recommended to "*Restrict which hosts can be used to remotely administer*" on the smartphone where the restriction is "*by IP address (not hostname)*" [124].

Configuration of a smartphone's firewall to comply with best practice recommendations of this kind ensures that the IP-based attack surface is significantly reduced. Note, while a smartphone's remote management server apps, such as a VPN or SSH, may provide their own protection in terms of authentication and authorisation, it is considered best practice to also restrict access at the smartphone firewall as part of a defence in depth strategy [152].

IP-based Spoof Mitigation. An IP packet's source address may be spoofed (forged) by an attacker in an attempt to trick the smartphone into processing the packet as if it had originated from the smartphone itself or from devices tethered to it. An external attacker may forge IP packets with a set of source IP addresses, for example `192.168.0.0/16`, that are associated with an internal private IP network range [33], but are inbound on the 3g or WiFi external interface.

A smartphone firewall configured in accordance with standards of best practice will mitigate against the threat of IP spoofing. For example, NIST 800-41rev1 recommendation FBPr1-2 in Table B.4 recommends that (spoofed) packets arriving on an external interface claiming to have originated from either of the three RFC 1918 [115] reserved internal IP address ranges should be dropped. This type of attack typically forms part of a Denial of Service (DoS) attack.

Port Scan Mitigation. Port Scanning is a reconnaissance technique that attackers use to determine the network resources of the smartphone and of its tethered devices. Typical TCP-based port scanning involves exploiting the intended use of the TCP protocol by forging TCP header flags.

Firewalls provide an effective way to mitigate against invalid TCP packets. For example, the XMAS TCP port scan where TCP flags FIN, PSH and URG are simultaneously set [91]. In addition to mitigating invalid TCP packets, a firewall that manages TCP communication state is an effective way to mitigate against valid TCP packets that are forged. For example, TCP packets forged to mimic the expected return packets for outbound TCP traffic requests.

Tunnel Bypass Mitigation. From the point of view of the firewall, the term *tunnelling* refers to the practice of encapsulating data from one protocol inside another protocol in order to evade the firewall [27]. For example, a Skype client typically listens on TCP and UDP port 33033 [14]. However, should Skype fail to establish communication over that port, it has the ability to operate on the port required by HTTP (port 80) [14, 21, 116]. As a consequence, despite denying traffic for TCP and UDP port 33033, Skype packets may still traverse the firewall unhindered by exploiting the intended purpose of HTTP-based firewall rules.

A smartphone firewall that can perform Deep Packet Inspection (DPI) mitigates the threat of tunnelling. The following is one of many possible Skype signatures used in a *Skype-to-Skype* communication that a firewall may be configured to filter [116].

```
^..\x02.............
```

Malware Traffic Mitigation. A smartphone firewall can be used to mitigate or reduce the flow of Malware communication even in infected phones. Well known Remote Access Trojans (RATs), such as Android's Geinimi Trojan [137], can be blocked in terms of protocol (`TCP`) and ports (`5432`, `4501` and `6543`) from indiscriminately making outbound connections to an external C&C. RATs may also communicate with their C&C over HTTP-based ports. For example, DroidDream [89] transmits IMEI, IMSI and device model information to its C&C server using the following URL:

```
http://184.105.XXX.XX:8080/GMServer/GMServlet
```

In this scenario, as with the *Tunnel Bypass Mitigation* Skype example, a firewall performing DPI with a deny action on outbound HTTP-based packet payloads that contain "`GMServer/GMServlet`" will prevent an infected smartphone from communicating with DroidDream's C&C. Note, best practice stipulates the avoidance of once-off firefighting rules where possible and to adopt a default deny rule on outbound traffic [152].

8.2.2 Reasoning About Smartphone Firewall Policies

In this section, a case study is presented that illustrates practical use of the $\mathcal{FW}_1$ algebra on Android systems. Consider, some end-user *Alice*, who over the course of her day encounters various scenarios while using her new smartphone.

7:30 a.m. As part of Alice's morning routine, she likes to check her email, read the news on her favourite Website and watch videos on YouTube. However, a problem on Android systems is that the firewall policy is open-access by default.

The following iptables rules implement a firewall policy $\mathsf{Pol}_{\mathsf{Home}}$ for Alice, whereby only her three favourite apps are allowed to access the Internet (rules $\mathsf{r}_1 \mathinner{..} \mathsf{r}_3$), and a default deny policy applies to all other network traffic.

$$\mathsf{r}_1 == \texttt{iptables -A OUTPUT -p tcp -m owner --uid-owner } \mathsf{\$email} \setminus$$
$$\texttt{-m state --state } \mathsf{NEW,ESTABLISHED,RELATED} \texttt{ -j ACCEPT}$$

$$\begin{aligned}\mathsf{r}_2 == {}& \texttt{iptables -A OUTPUT -p tcp -m owner --uid-owner } \mathsf{\$firefox} \setminus \\ & \texttt{-m state --state } \mathsf{NEW,ESTABLISHED,RELATED} \texttt{ -j ACCEPT}\end{aligned}$$

$$\begin{aligned}\mathsf{r}_3 == {}& \texttt{iptables -A OUTPUT -p tcp -m owner --uid-owner } \mathsf{\$youtube} \setminus \\ & \texttt{-m state --state } \mathsf{NEW,ESTABLISHED,RELATED} \texttt{ -j ACCEPT}\end{aligned}$$

The function $\mathcal{M}(\mathsf{rule})$ maps a rule written in iptables rule syntax to $\mathcal{FW}_1$ algebra syntax. The policy $\mathsf{Pol}_{\mathsf{Home}}$ is given as:

$$\mathsf{Pol}_{\mathsf{Home}} == \mathcal{I}(\mathcal{M}(\mathsf{r}_1)) \fatsemi \mathcal{I}(\mathcal{M}(\mathsf{r}_2)) \fatsemi \mathcal{I}(\mathcal{M}(\mathsf{r}_3))$$

The policy $\mathsf{Pol}_{\mathsf{Home}}$ addresses Alice's current requirements and she now has a working firewall implementation.

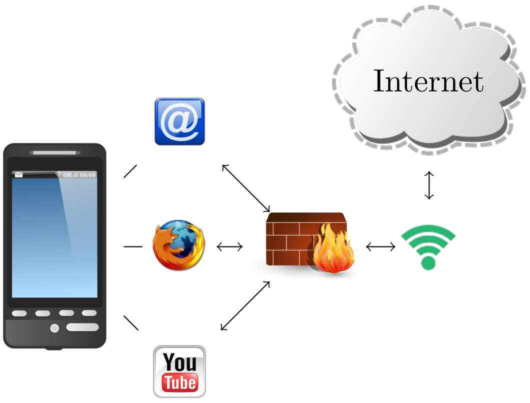


Figure 8.1: 7:30 a.m. ($\mathsf{Pol}_{\mathsf{Home}}$)

9:30 a.m. The company Alice works for has incorporated a BYOD policy. All employees have a VNC server running on their mobile devices to allow for screen sharing with local company systems. Company employees must use a non-standard Web browser for work-related activities, and non-authorised apps are restricted from accessing the company's internal network to help mitigate potential privacy violations.

The following iptables rules implement a simple BYOD policy $\mathsf{Pol}_{\mathsf{Work}}$ for Alice; whereby rule r_1 allows inbound network access to the VNC server port on Alice's device from the company's internal network, and rules $\mathsf{r}_2 \mathinner{\ldotp\ldotp} \mathsf{r}_5$ permit/deny network access to other applications running on the device.

$$\begin{aligned}\mathsf{r}_1 == {}& \texttt{iptables -A INPUT -p tcp -s } \mathsf{192.168.1.0/24} \texttt{ --dport } \mathsf{5900} \setminus \\ & \texttt{-m state --state } \mathsf{NEW,ESTABLISHED,RELATED} \texttt{ -j ACCEPT}\end{aligned}$$

```
r2 == iptables -A OUTPUT -p tcp -m owner --uid-owner $fFoxSec \
    -m state --state NEW,ESTABLISHED,RELATED -j ACCEPT
r3 == iptables -A OUTPUT -p tcp -m owner --uid-owner $email \
    -m state --state NEW,ESTABLISHED,RELATED -j DROP
r4 == iptables -A OUTPUT -p tcp -m owner --uid-owner $firefox \
    -m state --state NEW,ESTABLISHED,RELATED -j DROP
r5 == iptables -A OUTPUT -p tcp -m owner --uid-owner $youtube \
    -m state --state NEW,ESTABLISHED,RELATED -j DROP
```

The policy $\mathsf{Pol}_{\mathsf{Work}}$ implements the BYOD requirements of Alice's employer, and is given as:

$$\mathsf{Pol}_{\mathsf{Work}} == \mathcal{I}(\mathcal{M}(\mathsf{r}_1)) \fatsemi \mathcal{I}(\mathcal{M}(\mathsf{r}_2)) \fatsemi \mathcal{I}(\mathcal{M}(\mathsf{r}_3)) \fatsemi \mathcal{I}(\mathsf{r}_4) \fatsemi \mathcal{I}(\mathsf{r}_5)$$

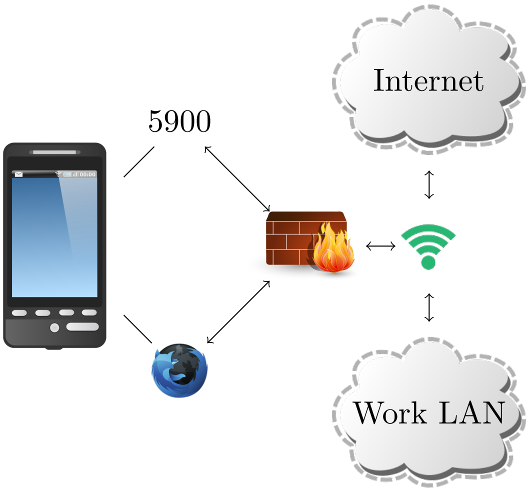


Figure 8.2: 9:30 a.m. ($\mathsf{Pol}_{\mathsf{Work}}$)

Alice needs to manage her policies. If she simply concatenated $\mathsf{Pol}_{\mathsf{Home}}$ and $\mathsf{Pol}_{\mathsf{Work}}$ then she introduces anomalies, in particular, shadowing anomalies; whereby rules $\mathsf{r}_1 \mathinner{\ldotp\ldotp} \mathsf{r}_3$ from $\mathsf{Pol}_{\mathsf{Home}}$ shadow (respectively) rules $\mathsf{r}_3 \mathinner{\ldotp\ldotp} \mathsf{r}_5$ in $\mathsf{Pol}_{\mathsf{Work}}$. We observe that:

$$(\mathbf{not}\ \mathsf{Pol}_{\mathsf{Home}}) \fatsemi \mathsf{Pol}_{\mathsf{Work}} = \mathsf{Pol}_{\mathsf{Work}}$$

From this, we note that a *one-size-fits-all* approach does not apply to smartphone firewall policies.

5:30 p.m. Alice likes visiting her favourite café after work, and regularly uses the open-access WiFi. Unfortunately for her, *Mike* is a customer also, and he is running target/service enumeration scans on the café network.

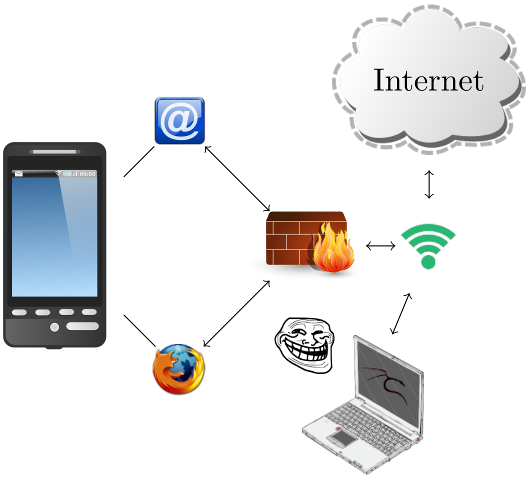


Figure 8.3: 5:30 p.m. (Internet café)

The following iptables rules implement a partial compliance policy NIST-800-114, for Alice, whereby the firewall will: *"silently ignore unsolicited requests sent to it, which essentially hides the device from malicious parties"* [125].

r_1 == `iptables -A INPUT -p tcp --tcp-flags` ALL NONE `-j DROP`

r_2 == `iptables -A INPUT -p tcp --tcp-flags` ALL ALL `-j DROP`

r_3 == `iptables -A INPUT -p tcp --tcp-flags` ALL FIN,PSH,URG `-j DROP`

r_4 == `iptables -A INPUT -p tcp --tcp-flags` \
ALL SYN,RST,ACK,FIN,URG `-j DROP`

r_5 == `iptables -A INPUT -p tcp --tcp-flags` \
FIN,PSH,URG FIN,PSH,URG `-j DROP`

r_6 == `iptables -A INPUT -p tcp --tcp-flags` ALL FIN `-j DROP`

r_7 == `iptables -A INPUT -j ACCEPT`

The policy NIST-800-114 denies various TCP flag combinations used in common port scanning techniques [91], while allowing all other network traffic, and is given as:

$$\mathsf{NIST\text{-}800\text{-}114} == \mathcal{I}(\mathcal{M}(r_1)) \fatsemi \mathcal{I}(\mathcal{M}(r_2)) \fatsemi \mathcal{I}(\mathcal{M}(r_3)) \fatsemi \mathcal{I}(\mathcal{M}(r_4)) \fatsemi \\ \mathcal{I}(\mathcal{M}(r_5)) \fatsemi \mathcal{I}(\mathcal{M}(r_6)) \fatsemi \mathcal{I}(\mathcal{M}(r_7))$$

Assume, that the firewall policy currently enforced on Alice's system is $\mathsf{Pol}_{\mathsf{Curr}}$, then if $\mathsf{Pol}_{\mathsf{Curr}} \sqsubseteq \mathsf{NIST\text{-}800\text{-}114}$ then Alice's current system policy complies with best practice recommendations outlined in [125] for unsolicited requests sent to the device.

9:30 p.m. The WiFi signal disappears, while the family (Bob and Claire) need to use the Internet. Bob has a work-related email to send and Claire needs to upload an assignment due for college. Tech-savvy Alice enables tethering. However, Alice's system again faces a new set of threats. For example, there may be Malware running on either/both Bob's and Claire's machines. Alice would like to reason with a degree of confidence that the traffic passing through her firewall is not part of some form of DoS attack.

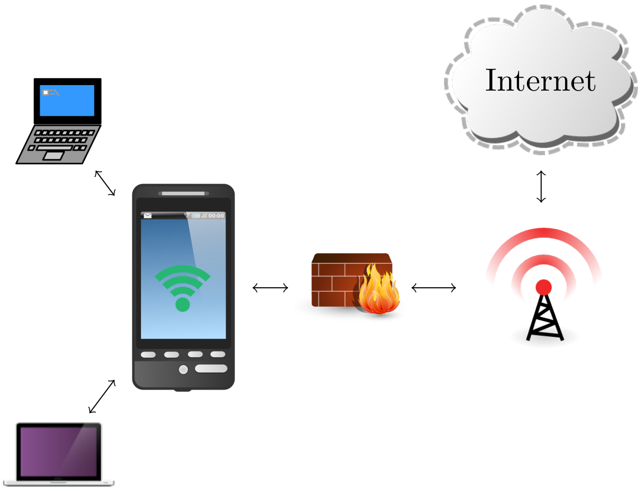


Figure 8.4: 9:30 p.m. (device tethering)

Recall, the IP spoof-mitigation compliance policies $\mathsf{RFC5735}^{\mathsf{I}}$, $\mathsf{RFC5735}^{\mathsf{O}}$ and $\mathsf{RFC5735}^{\mathsf{F}}$ defined is Chapter 5 Section 5.4.1. Then given the policy currently enforced on Alice's system as $\mathsf{Pol}_{\mathsf{Curr}}$, we have that if $\mathsf{Pol}_{\mathsf{Curr}} \sqsubseteq (\mathsf{NIST\text{-}800\text{-}114} \sqcap \mathsf{RFC5735}^{\mathsf{I}} \sqcap \mathsf{RFC5735}^{\mathsf{O}} \sqcap \mathsf{RFC5735}^{\mathsf{F}})$, then Alice's current system policy complies with best practice recommendations outlined in [125] for unsolicited requests sent to the device and the best practice recommendations outlined in [33] for IP address spoof-mitigation.

8.2.3 MASON

End-user management of security configurations that mitigate smartphone threats is complex and error-prone. As a consequence, misconfiguration of a security configuration may unnecessarily expose a smartphone to known threats. There are a number of existing techniques for static and dynamic analysis of smartphone applications. The authors in [127] adopt a static analysis approach to detect Android-based Malware. In [43], a tool called PiOS is developed and uses static analysis techniques to detect data flows in Mach-0 binaries. This provides a basis to detect privacy leaks in Apple's iOS applications. TaintDroid [45] is a smartphone application that uses dynamic analysis techniques to detect privacy leaks in Android applications. A machine learning approach is taken in [130] to detect application anomalies. There are a number of Android apps for firewall configuration management, for example DroidWall [163], WhipserMonitor [92] and NoRoot [147]. However, in existing works [92, 147, 163], Android firewall configuration is performed on an ad-hoc basis.

MASON [55], is a prototype automated agent app for Android that manages the firewall configuration on behalf of the non-expert end-user. In contrast to [92, 147, 163], the automatic generation of smartphone firewall configurations in this research is guided by best practice recommendations. The current version of MASON minimises the potential for firewall configuration conflicts as follows. Generalisation firewall rules that apply to app's as a whole, for example anti-port scanning and anti-bogon firewall rules, are given precedence over the disjoint singleton (specific) firewall rules. For the most part, firewall rules are disjoint singleton rules where rule ordering is irrelevant. That is, for each app requiring network access, there is a corresponding firewall rule that also filters based on that app's UID. The current implementation of MASON assumes that the firewall configuration is conflict free and does not consider firewall policy structural analysis [6, 35]. We argue that extending MASON with the $\mathcal{FW}_1$ algebra provides a means of anomaly-free, dynamic firewall policy reconfiguration for Android. An extended version of the prototype may also include a means to manage policies other than those specific to network access control, for example, policies to manage Android permission assignment to system applications using the Android Device Administration API [142].

8.3 The Android Permission Model

At the core of the Android security framework is the permission-based model. The model restricts permission assignment to an application in two ways: by user confirmation and through signatures by developer keys [46, 141]. An application has no associated permissions by default, and permissions required by the application must be specified by the developer in the application's Manifest file. Permissions are categorized in terms of four threat-levels [144]:

- **Normal:** *permission is granted to any application that requests it.*
- **Dangerous:** *permission is not automatically granted to the requesting application and requires user confirmation.*
- **Signature:** *permission is granted if the requesting application is signed with the same certificate as the application that declared the permission.*
- **Signature or System:** *permission is granted if the requesting application has the same signature as the application that declared the permission, or is granted to apps in the Android system image.*

For Android applications signed with the same certificate and specifying the sharedUserId attribute in their respective Manifest files, then these Applications share the set union of their permissions. For platform version Android 5.1 (API level 22) and lower, *dangerous* permissions must be granted to the application at install-time, or the install will fail. A problem with this, is that it may introduce over-privileged, potentially malicious applications to the system. The user may blindly accept the permission requests, or be unable to deny individual requests considered to be unnecessary for the legitimate operation of the application. Once granted, a permission cannot be revoked.

Barrera et al. [12], present an empirical analysis of the Android permission model using 1,100 apps as a case-study. Results show that a small subset of Android permissions are used frequently, while a large subset of permissions were used by very few applications. The authors note potential points of improvement for the Android permission model, such as the lack of expressiveness in the Internet permission. For example, an Android app that holds the Internet permission has unrestricted network access over WiFi and 3g connections. Enck et al. [46], present Kirin, a security service that reads application permission requirements during installation and checks them against a set of security rules. Rules are used to identify dangerous application permission configurations, such as applications that require Internet access and the ability to process audio and record outgoing calls, as an app with these permissions may potentially record and forward phonecalls to a remote location. If an application fails on any security rule, Kirin blocks the install and the user is alerted. While a light-weight application certification mechanism such as Kirin is desirable, it requires modification of the Android security framework to integrate with the installation process of an Android application. The authors do not consider applications sharing permissions through use of the sharedUserId attribute.

For platform version Android 6.0 (API level 23) and higher, a *dangerous* permission is granted by the user to the application at run-time, and this can be revoked/granted again later. While this run-time permission model is a notable improvement over the install-time model, there are still significant security considerations to be addressed. For example, in [44], attacks against the system are composed from seemingly innocuous applications requesting apparently benign collections of permissions, and many permissions used to construct the attacks are classified under the run-time permission model as *normal*, for example, the Internet permission and the permission to start at system boot.

8.3.1 Encoding The Permission Model

In this section, we develop a formal model of Android permission management for system applications.

Applications. We define Android applications by their UID. Thus, we define the set of all Android apps for a system to be *UID*, given in Chapter 3 Section 3.2.4.

Permissions. Android permissions categorized as *dangerous* [144] may be granted to/revoked from system applications using the Android Device Administration API [142]. Let *Permission* be the set of Android permissions characterized in [144] as having a threat-level of *dangerous*. We define:

$$\begin{aligned}Permission ::= {} & \mathsf{read\_calendar} \mid \mathsf{write\_calendar} \mid \mathsf{camera} \mid \mathsf{read\_contacts} \mid \\ & \mathsf{write\_contacts} \mid \mathsf{get\_accounts} \mid \mathsf{access\_fine\_location} \mid \mathsf{access\_coarse\_location} \mid \\ & \mathsf{record\_audio} \mid \mathsf{read\_phone\_state} \mid \mathsf{call\_phone} \mid \mathsf{read\_call\_log} \mid \mathsf{write\_call\_log} \mid \\ & \mathsf{add\_voicemail} \mid \mathsf{use\_sip} \mid \mathsf{process\_outgoing\_calls} \mid \mathsf{body\_sensors} \mid \mathsf{send\_sms} \mid \\ & \mathsf{receive\_sms} \mid \mathsf{read\_sms} \mid \mathsf{receive\_wap\_push} \mid \mathsf{receive\_mms} \mid \\ & \mathsf{read\_external\_storage} \mid \mathsf{write\_external\_storage}\end{aligned}$$

Permission Policies. An Android permission policy is a pair of relations that define the application/permission mappings/pairs that may be granted or denied access to a given resource by the system. Let $\mathit{Policy}_{\mathit{Perm}}$ define the set of all Android permissions policies, whereby:

$$\mathit{Policy}_{\mathit{Perm}} == \{G, D : (\mathit{UID} \leftrightarrow \mathit{Permission}) \mid G \cap D = \emptyset\}$$

A permissions policy $(G, D) \in \mathit{Policy}_{\mathit{Perm}}$ defines that an application/permission pair $a \mapsto p \in G$ should be granted access to a given resource by the system, while an application/permission pair $a \mapsto p \in D$ should be denied access to a given resource by the system. Policy accessor functions *grant* and *deny* for permissions policies are assumed, and are analogous to functions *first* and *second* for ordered pairs. Definitions for permissions policy refinement and composition are analogous to the definitions given for the $\mathcal{FW}_0$ firewall policy model in Chapter 4 Section 4.2. Definitions for policy construction, policy negation and policy sequential composition are also analogous to those given for the $\mathcal{FW}_0$ policy algebra in Chapter 4. Thus, we define the Android permissions policy algebra as:

$$\mathit{Android}_{\mathit{Perm}} == (\mathit{Policy}_{\mathit{Perm}}, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$$

Note, the lattice properties of the $\mathcal{FW}_0$ policy algebra, described in Chapter 4 Section 4.2.1, also apply to $\mathit{Policy}_{\mathit{Perm}}$ policies in the $\mathit{Android}_{\mathit{Perm}}$ algebra.

8.4 A Compliance-driven Threat Model

In this section, we extend the threat-based model developed for MASON in [55], and consider the threat of apps with over/under-privileged Android permissions.

8.4.1 Catalogues of Best Practice

A best practice standard is a high-level document that defines a set of recommended best practices (countermeasures) to protect sensitive and critical system resources. The following best practice standards NIST 800-41 [152], NIST 800-41rev1 [123], NIST 800-124 [79], NIST 800-114 [125] and NIST 800-153 [133] for firewall access control have been encoded as part of this work. Excerpts of these catalogues are illustrated in Tables B.1, B.2, B.3, B.4 and B.5. For example, Table B.4 and Table B.5 illustrate excerpts of recommended best practice for general firewall configuration management [152] and firewall configuration management whilst teleworking [125], respectively.

The advantage of developing catalogues from best practice standards is it provides a basis to automatically generate compliant firewall configurations. For example, NIST 800-41rev1 recommendation FBPr1-2 in Table B.4 recommends that (spoofed) packets arriving on an external interface claiming to have originated from either of the three RFC1918 reserved internal IP address ranges should be dropped. Such traffic indicates a Denial of Service attack typically involving the TCP SYN flag. NIST 800-114 recommendation TBP-1 in Table B.5 recommends that in a teleworking scenario, a firewall should be configured with a whitelist of trusted network-based apps.

Catalogues developed as part of this work extends the catalogues in [61] specialised for mobile devices. New best practice catalogues, namely NIST 800-124 [79], NIST 800-114 [125] and NIST 800-153 [133] have also been developed. The catalogue of firewall best practice for smartphones developed as part of this research consists of one hundred and thirty five distinct threat and countermeasure pairs. Future research should extend this catalogue to include knowledge about other best practice standards. Note, the majority of the catalogue countermeasures are templates. For example, the following firewall rules outlined in TBP-1 Table B.5:

```
iptables -A OUTPUT -m owner --uid-owner $appUID \
    -m state --state NEW,ESTABLISHED,RELATED -j ACCEPT

iptables -A INPUT -m owner --uid-owner $appUID \
    -m state --state ESTABLISHED,RELATED -j ACCEPT
```

are template countermeasures that have a UID variable `$appUID` that is modified each time the firewall rules are applied to a locally executing network-based smartphone app. These rules ensure that outbound/inbound local application whitelist traffic is permitted, and as such, avoid the stateful anomalies of [66] that result from the omission of explicit rules in a policy.

8.4.2 Threat Taxonomy

Having analysed the best practice standards outlined in Section 8.4.1, known network-based threats were categorised in the following way: *Spoofing*, *Denial of Service*, *Scanning*, *Source Routing*, *Malicious Content*, *Promiscuity Level* and *Non-Audit*. We also consider NIST 800-163, that recommends limiting Android permissions, whereby: *"apps should have only the minimum permissions neces-*

sary and should only grant other applications the necessary permissions" [112], this is categorised as *Permission Promiscuity Level*. Note, other threat categories could be chosen, for example Microsoft's STRIDE classification [74]. Table 8.1 illustrates a fragment of the threat classification developed.

Detailed Threats	**Threat Category**
FBPr1-2 Threats	*Spoofing*
FBPr1-2 Threats FBPr1-4 Threats FBPr1-5 Threats	*DoS*
TBP-2 Threats	*Scanning*
FBPr1-3 Threats	*Source Routing*
TBP-4 Threats	*Malicious Content*
FBPr1-1 Threats TBP-1 Threats TBP-3 Threats	*Promiscuity Level*
TBP-5 Threats	*Non-Audit*
PBP-1 Threats	*Permission Promiscuity Level*

Table 8.1: Extract of threat catalogue

Spoofing. Threats classified as *Spoofing* are those that refer to IP address spoofing. For example, threats described by the FBPr1-2 recommendation in Table B.4 are considered spoofing threats.

DoS. *Denial of Service* threats are those that have the capability of flooding network resources. For example, in Table B.4 FBPr1-4 recommends IP address broadcast mitigation and FBPr1-5 recommends threshold-limiting to mitigate connection-based Denial of Service threats. Note, recommendation FBPr1-4 currently considers the more common /24 network broadcast range only and does not consider additional network broadcast ranges for example /25 or /26.

Scanning. Network information disclosure threats, for example, those outlined by the NIST 800-114 recommendation TBP-2 in Table B.5, are classified in this dissertation as *Scanning* threats.

Source Routing. *Source Routing*, for example NIST 800-41rev1 recommendation FBPr1-3 in Table B.4, is a threat classification where an attacker may specify the route the packet takes through the network and has the potential to bypass firewalls.

Malicious Content. From a firewall policy configuration perspective, *Malicious Content* threats are those that contain malformed application payloads such as URL parameters, form elements and SQL queries. Malicious Content may be mitigated in a variety of ways for example blacklisting known TCP/UDP ports or performing Deep Packet Inspection (DPI) on known malicious signatures. Recommendations TBP-4 in Table B.5 and FBP-2 in Table B.3 illustrate template DPI firewall rules that mitigate outbound and inbound Malicious Content threat communication.

Promiscuity Level. Threats that are categorised as *Promiscuity Level* are those that refer to IP address (and/or port) reachability in terms of unintended whitelisting or blacklisting. That is, an overly-promiscuous firewall configuration (unintended whitelisting), while permitting normal operation of the smartphone app, may expose other apps to unintended threats, whilst an overly-restrictive firewall configuration (unintended blacklisting) may prevent normal interoperation of services with the resulting failure of the smartphone app. An example of this is outlined by NIST 800-114 recommendation TBP-1 Table B.5.

Non-Audit. *Non-Audit* threats are those that do not log relevant traffic communications. From a compliance perspective, it is considered best practice to log traffic for auditing purposes. For example, NIST SP800-114 recommendation TBP-5 in Table B.5 outlines teleworking auditing threats and their corresponding firewall mitigation. Similarly, recommendation WiFiBP-2 in Table B.2 advocates logging for auditing purposes. Extending the $\mathcal{FW}_1$ policy algebra with a target action of *log* for firewall rules is a topic for future research, and is discussed in Chapter 9 Section 9.2. This extension to $\mathcal{FW}_1$ is necessary to consider the *Non-Audit* ruleset as part of a firewall configuration while using the algebra.

Permission Promiscuity Level. Threats that are categorised as *Permission Promiscuity Level* are those that refer to system resource reachability, for example, the device microphone/camera/etc., in terms of unintended whitelisting or blacklisting of apps and the associated required permission/s. That is, an overly-promiscuous application/permission configuration (unintended whitelisting), while permitting normal operation of the smartphone app, may expose other apps and/or the device user to unintended threats. For example, an untrusted application accessing the users' calendar and/or location, or using services that cost the user money, such as sending SMS or calling premium-rate phone numbers. Conversely, an overly-restrictive application/permission configuration (unintended blacklisting) may prevent normal interoperation of services with the resulting failure of the smartphone app. An example of this is outlined by NIST 800-163 recommendation PBP-1 Table B.6.

8.4.3 Device Security State

The security *State* of a smartphone represents attributes of a phone in use that may introduce vulnerabilities and/or influence how threats are mitigated. These attributes may correspond to, for example, user-preferences (indicating for instance, security risk appetite), or how the smartphone is currently used (for instance, communicating over a WiFi or 3g Internet connection). While there is potentially a large number of such attributes, for this part of the research we focussed on six, which in-part, based on best practice recommendations, have a direct impact on network access controls on smartphones. We argue that certain attributes, for example, the users' risk appetite, the device battery level, and how the device is currently used (for instance, operating in a teleworking scenario) also, in-part, based on best practice recommendations, have a direct impact on Android permission policy management.

Risk Appetite Attribute. This user-selected attribute reflects the level of risk that the user is willing to accept [149]. An appetite of hungry means that the user is willing to take risks and is satisfied with minimal countermeasures necessary to mitigate threats. An appetite of averse means that the user wishes for the most extensive countermeasures, for example, defense in depth. We define:

$$\mathit{RiskAppetite} ::= \mathsf{averse} \mid \mathsf{hungry}$$

Note, future research may consider additional risk appetite granularity and include minimalist, cautious and open attributes [149].

Teleworking Attribute. This attribute indicates whether the smartphone is used in teleworking, or non-teleworking mode. This is defined as:

$$\mathit{Telework} ::= \mathsf{true} \mid \mathsf{false}$$

Battery Level Attribute. The experimental results outlined in [55] found that the number of firewall rules can have an impact on battery consumption.

Therefore, when battery power is low, a user with a low risk appetite may wish to reduce the number of rules in the firewall. Thus, we include the current battery level in the state of the smartphone. We define:

$$Battery ::= \mathsf{lo} \mid \mathsf{hi}$$

Network Interface Attribute. A smartphone may be configured to communicate over WiFi and/or 3g networks. Note that a network interface configuration of WiFi and 3g, combined, corresponds to a tethering state. Let *Iface* define the set of possible network interface configurations, whereby:

$$Iface == \mathbb{P}\{\mathsf{wifi}, \mathsf{3g}\}$$

Network Connection Attribute. Different network connections may be trusted in different ways. For example, a WiFi connection providing WPA2-Enterprise security may be considered trusted, while an open WiFi connection in a default configuration may be considered untrusted. Let *NetConn* define the possible network connection attribute states, whereby:

$$NetConn ::= \mathsf{trusted} \mid \mathsf{untrusted}$$

Data Quota Attribute. This user-selected attribute reflects whether the user wishes to apply a maximum data download capacity. If a data quota is to be configured, for example in a scenario where a smartphone is operating in roaming mode, it will be applied to a relevant set of white-listed apps. We define:

$$Quota ::= \mathsf{true} \mid \mathsf{false}$$

Security State. The set of all possible states of the smartphone is defined as:

$$State == RiskAppetite \times Telework \times Battery \times Iface \times NetConn \times Quota$$

The (six-tuple) security *State* space provides a total of sixty-four states in which a smartphone may operate. However, we argue that certain attribute combinations are not valid and therefore the security state space may be reduced to forty. This is discussed in more detail in [55]. Table B.7 illustrates the valid security state matrix for MASON.

8.5 A System Policy Model

In this section, we develop a formal model for Android firewall and permission policy management.

System Policies. An Android system policy is an $\mathcal{FW}_1$ firewall policy and an Android permission policy. Let $\mathit{Policy}_{\mathit{Sys}}$ define the set of all Android system policies, whereby:

$$\mathit{Policy}_{\mathit{Sys}} == \mathit{Policy} \times \mathit{Policy}_{\mathit{Perm}}$$

Definitions for destructor functions *firewall* and *permission* are assumed, and are analogous to functions *first* and *second* for ordered pairs. Thus, we have for all $P \in \mathit{Policy}_{\mathit{Sys}}$ then $P = (\mathit{firewall}\,(P), \mathit{permission}\,(P))$. Definitions for system policy refinement and composition, including sequential composition, are defined as the Cartesian product of firewall and permission policy refinement and composition, respectively. Thus, we define the Android system policy algebra as:

$$\mathit{Android}_{\mathit{Sys}} == (\mathit{Policy}_{\mathit{Sys}}, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$$

Note, the lattice properties of the $\mathcal{FW}_0$ policy algebra, described in Chapter 4 Section 4.2.1, also apply to $\mathit{Policy}_{\mathit{Sys}}$ policies in the $\mathit{Android}_{\mathit{Sys}}$ algebra.

8.5.1 Security Configuration Synthesis

The current implementation of MASON makes configuration decisions for firewall policies based on six different binary attributes, comprising the security *State*, defined in Section 8.4.3. Suitable firewall configurations are automatically generated for each security state using the information contained in Table B.7 and the threat catalogues (for example Table B.5). The catalogues of best practice for network-based threat mitigation using a firewall are encoded as collections of iptables rules. The best-practice rules can be encoded in the $\mathcal{FW}_1$ policy algebra as compliance policies, for example, $\mathsf{RFC5735}^{\mathsf{I}}, \mathsf{RFC5735}^{\mathsf{O}}, \mathsf{RFC5735}^{\mathsf{F}}$ and $\mathsf{NIST\text{-}800\text{-}114}$, and as policies to manage the user-defined blacklisted and whitelisted networked apps by UID. NIST 800-114 recommendation TBP-3 in Table B.5 recommends that different Web browsers such as Firefox and Google Chrome, should be used in teleworking and non-teleworking scenarios. This is to minimise the Web browser used for general use, which may have become compromised with malicious plugins, from communicating in a teleworking scenario. User-specified application/permission blacklists are in keeping with NIST 800-163 recommendation PBP-1 in Table B.6, and may be managed by MASON through the Android Device Administration API [142]. Extending MASON with the $\mathit{Android}_{Sys}$ algebra provides a means of anomaly-free, dynamic firewall and permission policy reconfiguration for Android.

Example 1 Consider, how the algebra $\mathit{Android}_{Sys}$ is used to build the system policy for a given state. We have $\mathsf{Pol}_{\mathsf{Apps}} \in \mathit{Policy}_{Sys}$, whereby for this state:

$$\mathit{allow}(\mathit{firewall}\,(\mathsf{Pol}_{\mathsf{Apps}}))$$

defines the whitelist of network applications permitted network access by UID, and $\mathit{deny}(\mathit{firewall}\,(\mathsf{Pol}_{\mathsf{Apps}}))$ defines the blacklist of network applications denied network access by UID. Similarly,

$$\mathit{grant}(\mathit{permission}\,(\mathsf{Pol}_{\mathsf{Apps}}))$$

defines the whitelist of the application/permission pairs permitted access to given resources in the system for this state, and $\mathit{deny}(\mathit{permission}\,(\mathsf{Pol}_{\mathsf{Apps}}))$ is the blacklist of application/permission pairs. The permission policy for a given state is $\mathsf{Pol}_{\mathsf{Perm}} \in \mathit{Policy}_{Perm}$, whereby:

$$\mathsf{Pol}_{\mathsf{Perm}} == \mathit{permission}\,(\mathsf{Pol}_{\mathsf{Apps}})$$

Suppose, that in this state, we require that the system configuration complies with the best practice recommendations outlined in NIST 800-114 for threats within the scanning category. Consider, $(\mathsf{NIST\text{-}800\text{-}114}, \mathsf{Pol}_{\mathsf{Perm}}) \in \mathit{Policy}_{Sys}$. When constructing the security configuration for this state, that complies with NIST 800-114 for unsolicited requests sent to the device, while also enforcing the networked-app blacklist/whitelist for the state, and the application/permission blacklist/whitelist for the state, then the following policy fragment:

$$(\mathsf{NIST\text{-}800\text{-}114}, \mathsf{Pol}_{\mathsf{Perm}}) \fatsemi \mathsf{Pol}_{\mathsf{Apps}}$$

defines this part of the system policy. △

While various security states may have been related to the same threat categories, the security configuration generated for each state may be different.

Example 2 Consider security states state-3 and state-25 in Table B.7. Both security states are threatened by threats within the category IP spoofing. However, the specific/individual IP spoofing threats will differ for both security states. Because security state state-25 is concerned with tethering, it must consider additional firewall access-control rules that mitigate IP spoofing threats along its iptables FORWARD chain to protect smartphone tethered devices [79]. Note, in a tethering scenario, the smartphone is an Internet gateway for tethered devices.

Consider, system policies $(\mathsf{RFC5735}^{\mathsf{I}}, \mathsf{Pol}_{\mathsf{Perm}})$, $(\mathsf{RFC5735}^{\mathsf{O}}, \mathsf{Pol}_{\mathsf{Perm}})$, $(\mathsf{RFC5735}^{\mathsf{F}}, \mathsf{Pol}_{\mathsf{Perm}}) \in \mathit{Policy}_{\mathit{Sys}}$. Then for security state state-3, we construct the IP spoof mitigation policy as follows:

$$((\mathsf{RFC5735}^{\mathsf{I}}, \mathsf{Pol}_{\mathsf{Perm}}) \sqcap (\mathsf{RFC5735}^{\mathsf{O}}, \mathsf{Pol}_{\mathsf{Perm}}))$$

whereby the iptables rules that mitigate IP spoofing threats apply to the INPUT and OUTPUT chains. Similarly, we construct the IP spoof mitigation policy for state-25 as follows:

$$((\mathsf{RFC5735}^{\mathsf{I}}, \mathsf{Pol}_{\mathsf{Perm}}) \sqcap (\mathsf{RFC5735}^{\mathsf{O}}, \mathsf{Pol}_{\mathsf{Perm}}) \sqcap (\mathsf{RFC5735}^{\mathsf{F}}, \mathsf{Pol}_{\mathsf{Perm}}))$$

The security states state-3 and state-25 are also threatened by scanning. When constructing $\mathsf{Pol}_{\mathsf{Curr}} \in \mathit{Policy}_{\mathit{Sys}}$, whereby $\mathsf{Pol}_{\mathsf{Curr}}$ defines the overall security policy for a given state, then if this state is threatened by scanning and IP spoofing threats, and for each other threat category the state is threatened by, then:

$$\begin{aligned}\mathsf{Pol}_{\mathsf{Curr}} == ((&\mathsf{RFC5735}^{\mathsf{I}}, \mathsf{Pol}_{\mathsf{Perm}}) \sqcap (\mathsf{RFC5735}^{\mathsf{O}}, \mathsf{Pol}_{\mathsf{Perm}}) \sqcap ..\\ &.. \sqcap (\mathsf{NIST\text{-}800\text{-}114}, \mathsf{Pol}_{\mathsf{Perm}}))\, ⨾\, \mathsf{Pol}_{\mathsf{Apps}}\end{aligned}$$

defines the anomaly-free security configuration for the current state. △

Example 3 There are also scenarios where permitted (trusted) network apps in one security state may no longer be permitted in another security state. For example, trusted networked apps such as Telnet, FTP or games for example in security state-3 may alternate between whitelists and blacklists in a security state that involves teleworking, for example security state-1. Suppose, that $\mathsf{Pol}_{\mathsf{Perm}}$ defines the permission policy for the state, then when defining $\mathsf{Pol}_{\mathsf{Apps}}$ in a state alternating between the network app whitelist/blacklist, we have the application policy:

$$(\mathbf{not}\ (\mathit{firewall}\ (\mathsf{Pol}_{\mathsf{Apps}})), \mathsf{Pol}_{\mathsf{Perm}})$$

This ensures compliance with NIST 800-114 recommendation TPB-1 in Table B.5.

That is, only trusted apps defined in accordance with the enterprise-level teleworking security policy may be permitted network access. △

The lattice of policies $\mathit{Android}_{\mathit{Sys}}$ provides us with an algebra for constructing and interpreting Android system polices. An anomaly-free security configuration is defined by composing policy fragments derived from the threat catalogues as $P, Q \in \mathit{Policy}_{\mathit{Sys}}$. The current prototype implementation makes configuration decisions for firewall policies based on six different binary attributes, comprising the security state *State*. In a model of extended security states, the system policy ordering relation $\sqsubseteq$ may be used as a formal verification of the security policy to be enforced for a state configuration change. Extending the security states by exhaustively enumerating *State* as a means of manually building a catalogue of countermeasures is not scalable. Looking for optimal policy configurations in extended security states is a topic for future research, and is considered in Chapter 9 Section 9.2.

8.6 Discussion

In this chapter, a policy algebra $\mathit{Android}_{\mathit{Sys}}$ is defined, in which Android system policies, comprising firewall and permission policies, can be specified and reasoned about. The $\mathit{Android}_{\mathit{Sys}}$ algebra incorporates the $\mathcal{FW}_1$ firewall policy algebra, defined in Chapter 5 Section 5.3, and the Android permission policy algebra, defined in Section 8.3 of this chapter. The set of Android system policies form a lattice under safe replacement and this enables consistent operators for safe composition to be defined. Policies in this lattice are anomaly-free by construction, and thus, composition under glb and lub operators preserves anomaly-freedom. A policy sequential composition operator is also proposed that can be used to give precedence to the security requirements of one system policy over another in composition.

This chapter also extended the threat-based model defined in [55] for MASON to consider the threat of apps with over/under-privileged Android permissions. Catalogues developed as part of this work extend the catalogues in [61] with an emphasis on mobile devices and provided a basis with which to evaluate the security model. A more fine-grained approach to encoding threats related to apps with over/under-privileged Android permissions is a topic for future research, and is considered in Chapter 9 Section 9.2. Extending the MASON prototype with the proposed model would require mapping the catalogues of best practice policy fragments into the algebra as compliance policies, recall, for example,

$\mathsf{RFC5735}^{\mathsf{I}}$, $\mathsf{RFC5735}^{\mathsf{O}}$, $\mathsf{RFC5735}^{\mathsf{F}}$ and NIST-800-114. Given that Android applications are usually written in Java, then the Python $\mathcal{FW}_1$ algebra implementation, presented in Chapter 6, may be incorporated into the MASON prototype using Jython [145]; a Java-based implementation of Python that allows developers to run Python on any Java platform. Alternatively, the existing Python implementation may be incorporated in MASON using Kivy [150]; an open source Python library that runs on Android. The Android Device Administration API may be incorporated into the prototype to enable the end-user to specify blacklist/whitelist permission-policies, these policies can be incorporated into the algebra as sets, thereby enabling MASON to dynamically manage policy reconfiguration.

We argue that the MASON prototype, extended with the policy framework described in this chapter, may be used by non-expert end-users to automatically generate and reason over suitable anomaly-free firewall and permission policy configurations on Android systems, that are compliant with best practice recommendations, such as [79, 112, 123, 125, 133, 152].

Chapter 9

Conclusion and Future Research

Firewall policy management is complex and error-prone. Policies may need to be reconfigured for highly dynamic environments, and misconfiguration is common. Typical errors range from invalid syntax and incorrect rule ordering, to a failure to uphold a security policy due to lack of GUI-based firewall rule granularity, to errors resulting from the poor comprehension of a firewall configuration. An effective firewall policy may be further hampered by the poor understanding and/or management of the overall high-level security requirements. Policy management is often reliant on the expert-knowledge of security administrators, and drawing from best practice. A significant challenge is to reason confidently that a policy is anomaly-free, and adequately mitigates the threats outlined in the network security policy. There is a rich literature of work on managing firewall policy configurations. The work in [1, 5, 6, 30, 35, 66, 73, 87, 159] is focused on detecting and resolving anomalies in firewall policy configurations, while the approaches in [47, 52, 88, 95] enable an administrator to query a policy configuration. Work such as [3, 13, 39, 72, 84] permits an administrator to specify at a high-level of abstraction what would otherwise be low-level firewall rules. However, in general, literature is focused on a five-tuple firewall rule with a binary target action of *allow* or *deny*, and most consider only packet-filter policy configurations. The thesis of this dissertation is that *a firewall policy should be anomaly-free by construction, and as such, there is a need for a firewall policy language that allows for constructing, comparing, and composing anomaly-free policies.*

9.1 Overview

The objective of this dissertation has been to develop a theory about composing anomaly-free firewall policies, as having a consistent means of anomaly-free firewall policy composition enables a means of anomaly-free, dynamic firewall policy reconfiguration. The thesis of this dissertation was evaluated as follows. A model has been constructed to describe what it means to specify, compare and compose anomaly-free policies. The model is developed as a lattice structure and provides sound and consistent operators for firewall policy composition. The core filter condition constructs developed for firewall rules specify a partial mapping of the iptables `filter` table. This mapping has been used to define filter condition constraints that extend the conventional five-tuple firewall rule, used for example in [1, 5, 6, 30, 35, 73, 87, 159], to include additional filter condition attributes such as, for example, TCP flags, ICMP Codes/Types and time-based filtering. A simple firewall policy algebra $\mathcal{FW}_0$ has been developed for policies that are defined in terms of constraints on individual IP addresses, ports, protocols and additional filter condition attributes. The purpose of developing $\mathcal{FW}_0$ was to demonstrate the utility of specifying an algebra for firewall policies that supports rules with complex range-based constraints.

The firewall policy algebra $\mathcal{FW}_1$ developed in Chapter 5 defines the foundations for the work of this dissertation. Policies in the $\mathcal{FW}_1$ framework are defined over stateful and stateless firewall rules constructed in terms of constraints on source/destination IP/port ranges, the TCP, UDP and ICMP protocols, and additional filter condition attributes. The algebra allows policies to be composed in such a way, that the result upholds the access requirements of each policy involved; and permits one to reason as to whether some policy is a safe (secure) replacement for another policy in the sense of [56, 57, 77]. The set of policies form a lattice under safe replacement and this enables consistent operators for safe composition to be defined. Policies in this lattice are anomaly-free by construction, and thus, composition under glb and lub operators preserves anomaly-freedom. A policy sequential composition operator is also proposed that can be used to interpret firewall policies defined more conventionally as sequences of rules. The proposed algebra is used to reason about iptables firewall policy configurations. $\mathcal{FW}_1$ is a generic firewall algebra that can be used to model different firewall systems, and an n-tuple firewall rule filter condition specification is supported by the model. The effectiveness of the algebra is demonstrated by its application to anomaly detection, and standards compliance. Firewall rules in $\mathcal{FW}_0/\mathcal{FW}_1$ policies are defined in terms of a binary target action of *allow* or *deny*. We describe an extension of $\mathcal{FW}_1$ to incorporate an additional firewall rule target action of *log* as part of future research in Section 9.2.

We consider a number of possible areas to demonstrate the effectiveness of the approach in practice. A proof of concept prototype policy management toolkit that implements $\mathcal{FW}_1$ firewall policies for iptables is developed. Experiments have been conducted for $\fatsemi$, $\sqcup, \sqcap$ and $\sqsubseteq$ policy operators, and the preliminary results are reported in Chapter 6. Overall, the results are promising.

The cloud computing paradigm has become widely adopted, however, managing the host-based and network access controls within and across cloud deployments is complex and error-prone. The environment is highly dynamic due to platform and service migration, and cross-tenant accesses are often required to facilitate service-to-service communication. A policy model $\mathcal{FW}_{\mathit{OpenStack}}$ is proposed in Chapter 7 for OpenStack firewall policies using a derivation of $\mathcal{FW}_1$. OpenStack avails of multiple access control policies of varying types for a firewall deployment, and we use $\mathcal{FW}_{\mathit{OpenStack}}$ to provide a uniform way to specify and reason about OpenStack security group policies and perimeter firewall policies. A case study OpenStack deployment illustrates practical use of the algebra.

Deploying a fixed security configuration for a global set of threats is not practical for devices operating in mobile environments. Therefore, security configurations for mobile devices must be dynamic in order mitigate the relevant threats within a given scenario. Mobile devices may host a variety of security mechanisms, however, security configuration is typically performed by non-technical end-users. A policy management framework for Android is proposed in Chapter 8, and a case study deployment illustrates practical use of $\mathcal{FW}_1$. An algebra $\mathit{Android}_{\mathit{Sys}}$ is proposed that incorporates $\mathcal{FW}_1$ and the Android permission policy algebra $\mathit{Android}_{\mathit{Perm}}$, for managing the unified reconfiguration of Android firewall and Android permission policies. The compliance-driven threat model developed for the MASON prototype is extended to include knowledge related to Android permissions. The policy framework amalgamates the threat model and $\mathit{Android}_{\mathit{Sys}}$, and defines a means to dynamically manage the security configuration for firewall policies and *dangerous* run-time permissions on Android systems. We describe how integrating the policy management framework with the MASON prototype can be used to dynamically synthesise standards-compliant anomaly-free security configurations on behalf of the end-user of Android systems.

9.2 Future Research

Extending The Firewall Policy Algebra. There are a number of areas for future work.

- **A Log Action.** In this dissertation, the focus was on firewall rules with a target action of either *allow* or *deny*. From a compliance perspective, it is considered best practice to log traffic for auditing purposes [123]. Future work should extend the $\mathcal{FW}_1$ algebra to include a target action of *log* for firewall rules. An approach may be taken, whereby we extend $(A, D) \in \mathit{Policy}$ to $(A, D, L) \in \mathit{Policy}$, where $L \in \alpha[FC]$ and a filter condition $f \in L$ should be logged by the firewall. We give the destructor function *log* for firewall policies; whereby $\mathit{log}\,(A, D, L) = L$. For policy composition, then for $P, Q \in \mathit{Policy}$, we have $P \sqcap Q$ signifies the operation $(\mathit{log}\, P \oplus \mathit{log}\, Q)$ for logged filter conditions. Similarly, for $P \sqcup Q$ we have the logged filter conditions $(\mathit{log}\, P \otimes \mathit{log}\, Q)$. From this, we have that the ordering for logged filter conditions is defined similarly to the ordering for denied filter conditions. In practice, a logged filter condition may be shadowed by a filter condition with a target action of *allow* or *deny*. However, it is not the case that a filter condition with a target action of *log* can shadow a filter condition with a target action of *allow* or *deny*. Therefore, for sequential composition, then we have $P \mathbin{⨟} Q$ defines the logged filter conditions for the resulting policy as: $(\mathit{log}\, P \oplus (\mathit{log}\, Q \setminus_{\alpha[fc]} (\mathit{allow}\, P \oplus \mathit{deny}\, P)))$.

- **A Definition for Network Address Translation (NAT).** The NAT routing technology is often combined with firewalling [123]. This dissertation focused on the firewalling aspects of iptables, that is, the `filter` table. Future work should extend the $\mathcal{FW}_1$ algebra to include a definition for NAT. An approach may be taken, whereby we define the filter condition attribute *Table*, and $\mathsf{mangle}, \mathsf{nat}, \mathsf{filter}, \in \mathit{Table}$. Additionally, we require $\mathsf{prerouting}, \mathsf{postrouting} \in \mathit{Chain}$. Given that NAT is the process of transposing one IP address space into another, then rules in the algebra need to include definitions for source and destination NAT address. These address spaces may be modelled as S, $T \in \mathit{IP}_{\mathit{Spec}}$. Similarly, to include attributes for source and destination port translation, a rule would additionally include S, $T \in \mathit{Prt}_{\mathit{Spec}}$. The model of Netfilter in [3] includes a definition for NAT.

Extending The Android Policy Framework. Future work should consider the following areas.

- **The Threat-based Model.** A future iteration of the threat-based model for MASON should consider additional threats and attributes for the device security *State*. For example, the physical location of the device, where it may be advantageous to prevent the device operating in a teleworking scenario, for example, when it is located in a certain (untrusted) country or region of the world. Best practice recommendations such as, for example [112, 126], provide sources from where new threats and attributes may be identified. For example, in [126], it is recommended that user and application access to hardware such as the USB interface be restricted, to prevent misuse by unauthorized parties, and that all network interfaces not needed by the device, such as Bluetooth and NFC, be disabled, thereby reducing the overall attack surface of the system.

- **Optimal Policy Configurations in Extended Security States.** The current implementation of MASON makes configuration decisions for firewall policies based on six different binary attributes, comprising the security *State*. Pruning invalid attribute combinations resulted in forty configuration scenarios for which corresponding policies were manually constructed from the collections of best-practice firewall rules. However, to allow for an arbitrary number of additional security states in future work, then manually constructing a catalogue of best-practice policies is not scalable. For example, supporting three risk-appetite attribute values, or adding an additional binary attribute potentially doubles the size of the matrix of valid security states for MASON. Future research should investigate how policies can be constructed as a set of constraints over *State*, the associated threats and the collections of best-practice countermeasures; whereby given some $s \in State$, then finding an acceptable security policy $P \in Policy_{Sys}$ amounts to a Constraint Satisfaction Problem. Comparable techniques have been successfully used to generate secure configurations [9] given a collection of system constraints. Belhaouane et al. [16] propose a series of quantitative metrics to evaluate the comprehensive complexity of policies. A definition of policy complexity is proposed, along with a set of metrics to evaluate the reasoning effort needed to understand a policy. Future work should investigate a means of policy comparison in the $\mathcal{FW}_1$ algebra that incorporates metrics, as such an approach could be used to allow one to reason consistently about how much more restrictive one policy is when compared to another, thereby enabling trade-offs to be made when considering optimal policy decisions.

- **Combinations of Dangerous Permissions.** A more fine-grained approach to encoding threats related to apps with over/under-privileged Android permissions should also be considered. For example, in [44], it is shown how attacks incorporating a seemingly benign collection of permissions can pose a significant threat to the security of the system.

Interoperation With Other Domains. The general idea of a policy algebra for network access control can be applied to various domains of interest. For example, future work should investigate the application of the $\mathcal{FW}_1$ algebra to policy composition in Cyber Physical Systems [59] and in SDN hypervisors [117].

References

[1] M. Abedin, S. Nessa, L. Khan, and B.M. Thuraisingham. Detection and Resolution of Anomalies in Firewall Policy Rules. *Data and Applications Security XX, 20th Annual IFIP WG 11.3 Working Conference on Data and Applications Security, Sophia Antipolis, France*, July 2006.

[2] A. Abou El Kalam, R. El Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin. Organization Based Access Control. *4th IEEE International Workshop on Policies for Distributed Systems and Networks, Como, Italy*, June 2003.

[3] P. Adão, C. Bozzato, G. Dei Rossi, R. Focardi, and F.L. Luccio. Mignis: A semantic based tool for firewall configuration. In *Computer Security Foundations Symposium (CSF), 2014 IEEE 27th*, pages 351–365. IEEE, 2014.

[4] E. Al-Shaer. *Automated Firewall Analytics - Design, Configuration and Optimization.* Springer, 2014.

[5] E. Al-Shaer and H. Hamed. Firewall Policy Advisor for Anomaly Discovery and Rule Editing. *8th IFIP/IEEE International Symposium on Integrated Network Management, Colorado Springs, USA*, March 2003.

[6] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan. Conflict Classification and Analysis of Distributed Firewall Policies. *IEEE Journal on Selected Areas in Communications, Issue: 10, Volume: 23, Pages: 2069 - 2084*, October 2005.

[7] R.E. Allen, editor. *Concise Oxford Dictionary.* Oxford University Press, Oxford, UK, eight edition, 1990.

[8] A. Avizienis. The n-version approach to fault-tolerant software. *IEEE Trans. Software Eng.*, 11(12):1491–1501, 1985.

[9] B. Aziz, S.N. Foley, J. Herbert, and G. Swart. Configuring storage-area networks using mandatory security. *Journal of Computer Security*, 17(2):191–210, 2009.

[10] M. Balanza, O. Abendan, K. Alintanahin, J. Dizon, and B. Caraig. DroidDreamLight Lurks Behind Legitimate Android Apps. *6th International Conference on Malicious and Unwanted Software (MALWARE)*, April 2011.

[11] J.J. Barbish. IPFW - FreeBSD firewall. `https://www.cisco.com/c/en/us/td/docs/ios/12_2/security/configuration/guide/fsecur_c/scfacls.html`. [Online; accessed August-2016].

[12] D. Barrera, H. G. Kayacik, P. C. van Oorschot, and A. Somayaji. A methodology for empirical analysis of permission-based security models and its application to android. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 73–84, New York, NY, USA, 2010. ACM.

[13] Y. Bartal, A. Mayer, K. Nissim, and A. Wool. Firmato: A Novel Firewall Management Toolkit. *20th IEEE Symposium on Security and Privacy, Oakland, CA, USA*, May 1999.

[14] S. A. Baset and H. Schulzrinne. An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. *25th IEEE International Conference on Computer Communications (INFOCOM)*, April 2006.

[15] K. Basil et al. *OpenStack Security Guide, Sixth Revision.* OpenStack Foundation, April 2015.

[16] M. Belhaouane, J. García-Alfaro, and H. Debar. Evaluating the comprehensive complexity of authorization-based access control policies using quantitative metrics. In *SECRYPT 2015 - Proceedings of the 12th International Conference on Security and Cryptography, Colmar, Alsace, France, 20-22 July, 2015.*, pages 53–64, 2015.

[17] M. Belhaouane, J. García-Alfaro, and H. Debar. On the isofunctionality of network access control lists. In *10th International Conference on Availability, Reliability and Security, ARES 2015, Toulouse, France, August 24-27, 2015*, pages 168–173, 2015.

[18] G. Birkhoff. *Lattice Theory. Volume XXV of American Mathematical Society Colloquium Publications.* American Mathematical Society, Providence, 3rd edition, 1967.

[19] S. Bistarelli, S.N. Foley, and B. O'Sullivan. Reasoning about secure interoperation using soft constraints. In *Formal Aspects in Security and Trust: Second IFIP TC1 WG1.7 Workshop on Formal Aspects in Security and Trust (FAST), an event of the 18th IFIP World Computer Congress, August 22-27, 2004, Toulouse, France*, pages 173–186, 2004.

[20] S. Bistarelli, S.N. Foley, and B. O'Sullivan. A soft constraint-based approach to the cascade vulnerability problem. *Journal of Computer Security*, 13(5):699–720, 2005.

[21] A. Blaich, Q. Liao, A. Striegel, and D. Thain. Simplifying Network Management with Lockdown. *Symposium on Usable Privacy and Security (SOUPS) Pittsburgh, PA, USA*, July 2008.

[22] A. D. Brucker, L. Brügger, and B. Wolff. Formal firewall conformance testing: An application of test and proof techniques. *Softw. Test. Verif. Reliab.*, 25(1):34–71, January 2015.

[23] L. Buttyán, G. Pék, and T. Vinh Thong. Consistency verification of stateful firewalls is not harder than the stateless case. *Infocommunications Journal*, 64(1):2–8, 2009.

[24] B. Chapman and E.D. Zwicky. *Building Internet Firewalls.* O'Reilly and Associates, November 1995.

[25] M.J. Chapple, J. D'Arcy, and A. Striegel. An Analysis of Firewall Rulebase (Mis)Management Practices. *Journal of the Information Systems Security Association*, February 2009.

[26] Z. Chen, P. Wei, and A. Delis. Catching Remote Administration Trojans (RATs). *Software Practice and Experience*, 38(7):667–703, 2008.

[27] W.R. Cheswick and S.M. Bellovin. *Firewalls and Internet Security: Repelling The Wily Hacker.* Addison-Wesley Professional, April 1994.

[28] W.R. Cheswick, S.M. Bellovin, and A.D. Rubin. *Firewalls and Internet security: Repelling The Wily Hacker.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2003.

[29] E. Chin, A.P. Felt, K. Greenwood, and D. Wagner. Analyzing inter-application communication in android. *9th international conference on Mobile systems, applications, and services, (MobiSys), ACM, USA*, 2011.

[30] T. Chomsiri and C. Pornavalai. Firewall Rules Analysis. *International Conference on Security and Management (SAM), Las Vegas, Nevada, USA*, June 2006.

[31] Cisco. Cisco ACLs. `https://www.cisco.com/c/en/us/td/docs/ios/12_2/security/configuration/guide/fsecur_c/scfacls.html`. [Online; accessed August-2016].

[32] D.D. Clark and D.R. Wilson. A comparison of commercial and military computer security policies. In *Security and Privacy, 1987 IEEE Symposium on*, pages 184–184. IEEE, 1987.

[33] M. Cotton and L. Vegoda. Special Use IPv4 Addresses. RFC 5735, January 2010.

[34] PCI Security Standards Council. Navigating PCI DSS: Understanding the Intent of the Requirements, Version 2. *Payment Card Industry (PCI) Data Security Standard*, October 2010.

[35] F. Cuppens, N. Cuppens-Boulahia, and J. García-Alfaro. Detection and Removal of Firewall Misconfiguration. *IASTED International Conference on Communication, Network and Information Security (CNIS)*, November 2005.

[36] F. Cuppens, N. Cuppens-Boulahia, and J. García-Alfaro. Misconfiguration Management of Network Security Components. *7th International Symposium on System and Information Security (SSI), Sao Paulo, Brazil*, November 2005.

[37] F. Cuppens, N. Cuppens-Boulahia, and J. García-Alfaro. Detection of Network Security Component Misconfiguration by Rewriting and Correlation. *1st Joint Conference on Security in Networks Architectures (SAR) and Security of Information Systems (SSI). Seignosse, France*, June 2006.

[38] F. Cuppens, N. Cuppens-Boulahia, J. García-Alfaro, T. Moataz, and X. Rimasson. Handling stateful firewall anomalies. In *Information Security and*

Privacy Research - 27th IFIP TC 11 Information Security and Privacy Conference, SEC 2012, Heraklion, Crete, Greece, June 4-6, 2012. Proceedings, pages 174–186, 2012.

[39] F. Cuppens, N. Cuppens-Boulahia, T. Sans, and A. Miège. A Formal Approach to Specify and Deploy a Network Security Policy. *2nd Workshop on Formal Aspects in Security and Trust (FAST)*, August 2004.

[40] R.A. Deal. *Cisco Router Firewall Security.* Cisco Press, August 2004.

[41] B. Dempster and J. Eaton-Lee. *Configuring IPCop Firewalls: Closing Borders with Open Source: How to set up, configure, and manage your Linux firewall, web proxy, DHCP, DNS, time server, and VPN with this powerful Open Source solution.* PACKT Publishing, October 2006.

[42] D.E. Denning. A lattice model of secure information flow. *Commun. ACM*, 19(5):236–243, May 1976.

[43] M. Egele, C. Kruegel, E. Kirda, and G. Vigna. PiOS: Detecting Privacy Leaks in iOS Applications. *Network and Distributed System Security Symposium (NDSS), California, USA*, February 2011.

[44] A. Egners, U. Meyer, and B. Marschollek. Messing with android's permission model. In *Proceedings of the 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, TRUSTCOM '12, pages 505–514. IEEE Computer Society, 2012.

[45] W. Enck. Defending users against smartphone apps: techniques and future directions. *7th international conference on Information Systems Security (ICISS), Kolkata, India*, December 2011.

[46] W. Enck, M. Ongtang, and P. McDaniel. On lightweight mobile phone application certification. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 235–245. ACM, 2009.

[47] P. Eronen and J. Zitting. An Expert System for Analyzing Firewall Rules. *6th Nordic Workshop on Secure IT Systems (NordSec), Copenhagen, Denmark*, pages 100–107, November 2001.

[48] D.L. Evans, P.J. Bond, and A.L. Bement. Security Requirements For Cryptographic Modules. *Federal Information Processing Standards Publication, FIPS 140-2*, May 2001.

[49] M. Fabrice. iptables Extensions - Time module. `http://ipset.netfilter.org/iptables-extensions.man.html`. [Online; accessed September-2016].

[50] G. Farnham and A. Atlasis. Detecting dns tunneling. *SANS Institute InfoSec Reading Room*, pages 1–32, 2013.

[51] P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827, May 2000.

[52] W.M. Fitzgerald and S.N. Foley. Management of heterogeneous security access control configuration using an ontology engineering approach. In *3rd ACM Workshop on Assurable and Usable Security Configuration, SafeConfig 2010, Chicago, IL, USA, October 4, 2010*, pages 27–36, 2010.

[53] W.M. Fitzgerald, S.N. Foley, and M. Ó Foghlú. Network access control interoperation using semantic web techniques. In *Security in Information Systems, Proceedings of the 6th International Workshop on Security in Information Systems, WOSIS 2008, In conjunction with ICEIS 2008, Barcelona, Spain, June 2008*, pages 26–37, 2008.

[54] W.M. Fitzgerald, U. Neville, and S.N. Foley. Automated smartphone security configuration. In *Data Privacy Management and Autonomous Spontaneous Security, 7th International Workshop, DPM 2012, and 5th International Workshop, SETOP 2012, Pisa, Italy, September 13-14, 2012. Revised Selected Papers*, pages 227–242, 2012.

[55] W.M. Fitzgerald, U. Neville, and S.N. Foley. MASON: Mobile Autonomic Security for Network Access Controls. *Journal of Information Security and Applications (JISA)*, 18(1):14–29, 2013.

[56] S.N. Foley. Reasoning about confidentiality requirements. In *Seventh IEEE Computer Security Foundations Workshop - CSFW'94, Franconia, New Hampshire, USA, June 14-16, 1994, Proceedings*, pages 150–160, 1994.

[57] S.N. Foley. The specification and implementation of commercial security requirements including dynamic segregation of duties. In *ACM Conference on Computer and Communications Security*, pages 125–134, 1997.

[58] S.N. Foley. Conduit cascades and secure synchronization. In *Proceedings of the 2000 Workshop on New Security Paradigms, Ballycotton, Co. Cork, Ireland, September 18-21, 2000*, pages 141–150, 2000.

[59] S.N. Foley. Reasoning about threats and defenses in a cyber physical system. Working notes, University College Cork, Ireland, 2016.

[60] S.N. Foley and W.M. Fitzgerald. An approach to security policy configuration using semantic threat graphs. In *23rd Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSec)*, pages 33–48. Springer LNCS, 2009.

[61] S.N. Foley and W.M. Fitzgerald. Management of Security Policy Configuration using a Semantic Threat Graph Approach. *Journal of Computer Security (JCS), IOS Press, Volume 19, Number 3*, 2011.

[62] S.N. Foley and U. Neville. A firewall algebra for openstack. In *2015 IEEE Conference on Communications and Network Security, CNS 2015, Florence, Italy, September 28-30, 2015*, pages 541–549. IEEE, 2015.

[63] OpenStack Foundation. OpenStack - Open source software for creating private and public clouds. `https://www.openstack.org/`. [Online; accessed August-2016].

[64] OpenStack Foundation. *OpenStack Cloud Administrator Guide, Eight Revision.* February 2015.

[65] Python Software Foundation. Python multiprocessing package - Process-based “threading” interface. `https://docs.python.org/2/library/multiprocessing.html`. [Online; accessed August-2016].

[66] J. García-Alfaro, F. Cuppens, N. Cuppens-Boulahia, S. Martínez Perez, and J. Cabot. Management of stateful firewall misconfiguration. *Computers & Security*, 39:64–85, 2013.

[67] J. García-Alfaro, F. Cuppens, N. Cuppens-Boulahia, and S. Preda. MIRAGE: A management tool for the analysis and deployment of network security policies. In *Data Privacy Management and Autonomous Spontaneous Security - 5th International Workshop, DPM 2010 and 3rd International Workshop, SETOP 2010, Athens, Greece, September 23, 2010, Revised Selected Papers*, pages 203–215, 2010.

[68] L. Gheorghe. *Designing and Implementing Linux Firewalls with QoS using netfilter, iproute2, NAT and l7-filter.* PACKT Publishing, October 2006.

[69] K. Golnabi, R. Min, L. Khan, and E. Al-Shaer. Analysis of Firewall Policy Rule Using Data Mining Techniques. *10th IEEE/IFIP Network Operations and Management Symposium (NOMS), Vancouver, Canada*, April 2006.

[70] L. Gong and X. Qian. The complexity and composability of secure interoperation. In *Research in Security and Privacy, 1994. Proceedings., 1994 IEEE Computer Society Symposium on*, pages 190–200. IEEE, 1994.

[71] M. Gouda and A. Liu. Firewall Design: Consistency, Completeness and Compactness. *24th IEEE International Conference on Distributed Computing Systems (ICDCS), Tokyo, Japan*, March 2004.

[72] J.D. Guttman. Filtering Postures: Local Enforcement for Global Policies. *IEEE Symposium on Security and Privacy*, pages 120–129, May 1997.

[73] A. Hari, S. Suri, and G. Parulkar. Detecting and Resolving Packet Filter Conflicts. *19th Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM*, 3:1203–1212, March 2000.

[74] S. Hernan, S. Lambert, T. Ostwald, and A. Shostack. Uncover Security Design Flaws Using The STRIDE Approach. *http://microsoft.com/.*

[75] M. Howard, J. Pincus, and J.M. Wing. Measuring relative attack surfaces. In *Computer Security in the 21st Century*, pages 109–137. Springer, 2005.

[76] ISO. ISO 8601: Data elements and interchange formats – Information interchange – Representation of dates and times, 1988.

[77] J.L. Jacob. The varieties of refinement. In J.M. Morris and R.C. Shaw, editors, *Proceedings of the 4th Refinement Workshop*, pages 441–455. Springer, Heidelberg, 1991.

[78] S. Jajodia, P. Samarati, M.L. Sapino, and V.S. Subrahmanian. Flexible support for multiple access control policies. *ACM Trans. Database Syst.*, 26(2):214–260, 2001.

[79] W. Jansen and K. Scarfone. Guidelines on Cell Phone and PDA Security: Recommendations of the National Institute of Standards and Technology. *NIST-800-124*, 2008.

[80] Y. Jarraya, A. Eghtesadi, M. Debbabi, Y. Zhang, and M. Pourzandi. Cloud calculus: Security verification in elastic cloud computing platform. In *Collaboration Technologies and Systems (CTS), 2012 International Conference on*, pages 447–454. IEEE, 2012.

[81] S. Khadem. Security issues in smartphones and their effects on the telecom networks. *MSc Dissertation, Chalmers University of Technology, University of Gothenburg, Sweden*, August 2010.

[82] M. Kuzin and N. Buchka. Good morning Android! - Malware downloaded via the Google AdSense advertising network. `https://securelist.com/blog/incidents/75731/good-morning-android/`. [Online; accessed August-2016].

[83] C. Lathem, B.W. Fortenberry, and J. Reed. *Configuring SonicWALL Firewalls*. Syngress, July 2006.

[84] A. Launay. High Level Firewall Language. `https://www.cusae.com/hlfl/`, 2003. [Online; accessed August-2016].

[85] J. Levandoski et al. iptables L7-filter Pattern Writing. `http://l7-filter.sourceforge.net/Pattern-HOWTO`, April 2008. [Online; accessed September-2016].

[86] J. Levandoski et al. iptables L7-filter Supported Protocols. `http://l7-filter.sourceforge.net/protocols`, August 2008. [Online; accessed August-2016].

[87] A. Liu and M. Gouda. Diverse Firewall Design. *34th IEEE International Conference on Dependable Systems and Networks (DSN), Florence, Italy*, pages 595–604, June 2004.

[88] A. Liu, M. Gouda, H. Ma, and A. Hgu. Firewall Queries. *8th International Conference, On Principles of Distributed Systems (OPODIS), Grenoble, France*, pages 197–212, December 2004.

[89] Lookout. Droiddream Android Malware. `https://blog.lookout.com/droiddream/`. [Online; accessed August-2016].

[90] T.F. Lunt. Access control policies: Some unanswered questions. *Computers & Security*, 8(1):43–54, 1989.

[91] G. Lyon. *NMAP Network Scanning: Official Nmap Project Guide to Network Discovery and Security Scanning.* Insecure LLC, CA, United States, 2008.

[92] M. Marlinspike. WhisperMonitor Android Firewall. `http://www.whispersys.com/`. [Online; accessed May-2011].

[93] R. Marmorstein and P. Kearns. A Tool for Automated iptables Firewall Analysis. *Usenix Annual Technical Conference, Freenix Track, Pages: 71-81*, April 2005.

[94] R. Marmorstein and P. Kearns. Debugging a Firewall Policy with Policy Mapping. *;login: The Usenix Magazine, Volume 32, Number 1, Berkeley, CA, USA*, February 2007.

[95] A. Mayer, A. Wool, and E. Ziskind. Fang: A Firewall Analysis Engine. *21st IEEE Symposium on Security and Privacy, Oakland, CA, USA*, May 2000.

[96] A. Mayer, A. Wool, and E. Ziskind. Offline Firewall Analysis. *International Journal of Information Security*, 5(3):125–144, May 2006.

[97] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Science*, 7:95–132, 1974.

[98] J. Nazario. *Defense and Detection Strategies against Internet Worms.* Artech House, 2004.

[99] NCSC. Trusted network interpretation of the trusted computer system evaluation criteria. 1987, Red Book.

[100] V. Nebehaj. python-iptables - Python bindings for iptables. `https://pypi.python.org/pypi/python-iptables`. [Online; accessed August-2016].

[101] U. Neville and S.N. Foley. Reasoning about firewall policies through refinement and composition. In *Data and Applications Security and Privacy XXX - 30th Annual IFIP WG 11.3 Conference, DBSec 2016, Trento, Italy, July 18-20, 2016. Proceedings*, pages 268–284, 2016.

[102] NISCC. Egress and Ingress Filtering, Technical Note 01/2006, National Infrastructure Security Co-ordination Centre (NISCC). Apr 2006. `http://www.csirtuk.gov.uk/documents/publications/2006/2006004-tn0106_egress_ingress.pdf`.

[103] S. Northcutt, L. Zeltser, S. Winters, K. Kent F., and R.W. Ritchey. *Inside Network Perimeter Security: The definitive Guide to Firewalls, VPNs, Routers, and Intrusion Detections Systems.* New Riders, 2003.

[104] I. Ocean. Pyinter - a small and simple library written in Python for performing interval and discontinous range arithmetic. `https://pypi.python.org/pypi/pyinter/`, August 2015. [Online; accessed August-2016].

[105] R. Oppliger. *Internet and Intranet Security.* Artech House, 2nd revised edition, October 2001.

[106] J. Phillips and B. Harrington. Openclipart vector clip art. `https://openclipart.org/`. [Online; accessed August-2016].

[107] J. Postel. User Datagram Protocol. RFC 768, August 1980.

[108] J. Postel. Internet Control Message Protocol. RFC 792, September 1981.

[109] J. Postel. Transmission Control Protocol. RFC 793, September 1981.

[110] J. Postel, D. Johnson, T. Markson, B. Simpson, and Z. Su. Internet Control Message Protocol (ICMP) Parameters. `https://www.iana.org/assignments/icmp-parameters/icmp-parameters.xhtml`, April 2013. [Online; accessed August-2016].

[111] J. Postel and J. Reynolds. File Transfer Protocol. RFC 959, October 1985.

[112] S. Quirolgico, J. Voas, T. Karygiannis, C. Michael, and K. Scarfone. Vetting the Security of Mobile Applications. *NIST Special Publication 800-163*, January 2015.

[113] F. Rabitti, D. Woelk, and W. Kim. A model of authorization for object-oriented and semantic databases. In *International Conference on Extending Database Technology*, pages 231–250. Springer, 1988.

[114] S.C. Reid. An empirical analysis of equivalence partitioning, boundary value analysis and random testing. In *Software Metrics Symposium, 1997. Proceedings., Fourth International*, pages 64–73. IEEE, 1997.

[115] Y. Rekhter, R. G Moskowitz, D. Karrenberg, G. Jan de Groot, and E. Lear. RFC1918: Address Allocation for Private Internets. *http://ietf.org*, Febuary 1996.

[116] P. Renals and G. N. Jacoby. Blocking Skype through Deep Packet Inspection. *42nd International Conference on System Sciences (HICSS), IEEE Computer Society, Waikoloa, Big Island, Hawaii, USA*, January 2009.

[117] M. Rezvani, A. Ignjatovic, M. Pagnucco, and S. Jha. Anomaly-free policy composition in software-defined networks. In *2016 IFIP Networking Conference, Networking 2016 and Workshops, Vienna, Austria, May 17-19, 2016*, pages 28–36, 2016.

[118] D.J.S. Robinson. *An Introduction to Abstract Algebra.* Walter de Gruyter, 2003.

[119] P. Ruggiero and J. Foote. Cyber threats to mobile phones. *TIP-10-105-01, United States Computer Emergency Readiness Team (US-CERT)*, April 2010.

[120] J.H. Saltzer. Protection and the control of information sharing in multics. *Communications of the ACM*, 17(7):388–402, 1974.

[121] P. Samarati and S. De Capitani di Vimercati. Access control: Policies, models, and mechanisms. In *International School on Foundations of Security Analysis and Design*, pages 137–196. Springer, 2000.

[122] R.S. Sandhu. Lattice-based access control models. *Computer*, 26(11):9–19, November 1993.

[123] K. Scarfone and P. Hoffman. Guidelines on Firewalls and Firewall Policy: Recommendations of the National Institute of Standards and Technology. *NIST Special Publication 800-41, Revision 1*, September 2009.

[124] K. Scarfone, W. Jansen, and M. Tracy. Guide to General Server Security. *NIST Special Publication 800-123*, July 2008.

[125] K. Scarfone and M. Souppaya. User's Guide to Securing External Devices for Telework and Remote Access: Recommendations of the National Institute of Standards and Technology. *NIST-800-114*, 2007.

[126] K. Scarfone and M. Souppaya. Guidelines for Managing the Security of Mobile Devices in the Enterprise. NIST Special Publication 800-124, Revision 1, June 2013.

[127] A.D. Schmidt, R. Bye, H.G. Schmidt, J. Clausen, O. Kiraz, K.A. Yüksel, S.A. Camtepe, and S. Albayrak. Static analysis of executables for collaborative malware detection on android. In *Proceedings of the 2009 IEEE international conference on Communications*, ICC'09, Piscataway, NJ, USA, 2009. IEEE Press.

[128] R. Sedgewick and K. Wayne. *Computer Science: An Interdisciplinary Approach.* Addison-Wesley Professional, 2016.

[129] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer. Google android: A comprehensive security assessment. *Security and Privacy, IEEE Computer Society, Volume 8, Issue 2*, March 2010.

[130] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss. "andromaly": a behavioral malware detection framework for android devices. *J. Intell. Inf. Syst.*, 38(1), February 2012.

[131] H. Shen and P. Dewan. Access control for collaborative environments. In *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, pages 51–58. ACM, 1992.

[132] R. Shirey. Internet Security Glossary. RFC 2828, May 2000.

[133] M. Souppaya and K. Scarfone. Guidelines for Securing Wireless Local Area Networks (WLANs): Recommendations of the National Institute of Standards and Technology. *NIST-800-153*, 2012.

[134] J.M. Spivey. The fuzz manual. *Computing Science Consultancy*, 34, 1992.

[135] J.M. Spivey. *The Z Notation: A Reference Manual.* Series in Computer Science. Prentice Hall International, second edition, 1992.

[136] W. Stallings and L. Brown. *Computer Security: Principles and Practice.* Pearson Prentice Hall, 2008.

[137] T. Strazzere and T. Wyatt. Geinimi Trojan Technical Teardown. *Lookout Mobile Security*, June 2011.

[138] S. Suehring and R. L. Ziegler. *Linux Firewalls: Third Edition.* Novell Publishing, 2006.

[139] Symantic. Internet Security Threat Report (ISTR). `https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf`, April 2016. [Online; accessed September-2016].

[140] Tabascoeye. TikZ networking diagrams. https://github.com/tabascoeye/TikZ-diagrams/tree/master/networking. [Online; accessed August-2016].

[141] Android Core Team. Android Developer Information. Tools and documentation on how to create Android applications. http://developer.android.com. [Online; accessed August-2016].

[142] Android Core Team. Android Device Administration API. https://developer.android.com/guide/topics/admin/device-admin.html. [Online; accessed September-2016].

[143] Android Core Team. Android Interfaces and Architecture. https://source.android.com/devices/. [Online; accessed August-2016].

[144] Android Core Team. Android Permission Categories. https://developer.android.com/guide/topics/manifest/permission-element.html. [Online; accessed September-2011].

[145] Jython Core Team. Jython - an implementation of the Python programming language written in Java. http://www.jython.org/. [Online; accessed August-2016].

[146] Netfilter Core Team. Linux iptables - CLI for configuring the Linux kernel firewall, Netfilter. http://www.netfilter.org/projects/iptables/index.html. [Online; accessed August-2016].

[147] NoRoot Core Team. NoRoot Firewall: Android Firewall. https://play.google.com/store/apps/details?id=app.greyshirts.firewall&hl=en. [Online; accessed August-2016].

[148] H.F. Tipton and M. Krause. *Information Security Handbook, 5th Edition, Volume 3*. Auerbach Publications, 2006.

[149] HM Treasury. Thinking about risk - managing your risk appetite: A practitioner's guide. *London, The Stationery Office*, 2006.

[150] M. Virbel et al. Kivy - an open-source Python library that runs on Linux, Windows, OS X, Android, iOS, and Raspberry Pi. https://kivy.org/. [Online; accessed August-2016].

[151] B. Volz et al. Service Name and Transport Protocol Port Number Registry. https://www.iana.org/assignments/service-names-port-

numbers/service-names-port-numbers.xhtml, August 2016. [Online; accessed August-2016].

[152] J. Wack, K. Cutler, and J. Pole. Guidelines on Firewalls and Firewall Policy: Recommendations of the National Institute of Standards and Technology. *NIST-800-41*, 2002.

[153] M. Wallace, S. Novello, and J. Schimpf. Eclipse: A platform for constraint logic programming. *ICL Systems Journal*, 12(1):159–200, 1997.

[154] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos. Malicious Android Applications in the Enterprise: What Do They Do and How Do We Fix It? *Workshop on Secure Data Management on Smartphones and Mobiles, Washington D.C*, April 2012.

[155] A. Wool. Architecting the Lumeta firewall analyzer. *10th conference on USENIX Security Symposium (SSYM), Washington, D.C., USA*, August 2001.

[156] A. Wool. A Quantitative Study of Firewall Configuration Errors. *IEEE Computer*, 37(6):62–67, June 2004.

[157] A. Wool. Trends in firewall configuration errors: Measuring the holes in swiss cheese. *IEEE Internet Computing*, 14(4):58–65, 2010.

[158] H. Yang and S. Lam. Real-time verification of network properties using atomic predicates. *IEEE/ACM Transactions on Networking*, 24(2):887–900, 2016.

[159] L. Yuan, H. Chen, J. Mai, C. Chuah, Z. Su, and P. Mohapatra. Fireman: A toolkit for firewall modeling and analysis. In *Security and Privacy, 2006 IEEE Symposium on*, pages 15–pp. IEEE, 2006.

[160] H. Zhao. *Security Policy Definition and Enforcement in Distributed Systems.* PhD thesis, Graduate School of Arts and Sciences, Columbia University, USA, 2012.

[161] H. Zhao and S.M. Bellovin. Policy algebras for hybrid firewalls. Number CUCS-017-07, March 2007.

[162] Y. Zhou and X. Jiang. Dissecting android malware: Characterization and evolution. In *2012 IEEE Symposium on Security and Privacy*, pages 95–109. IEEE, 2012.

[163] R. ZR. DroidWall Android Firewall. `https://play.google.com/store/apps/details?id=com.googlecode.droidwall.free&hl=en`. [Online; accessed August-2016].

Part IV

Appendices

Appendix A

Presentation of Mathematical Definitions

A.1 The Z Notation

In this section, we build on the description we gave in [62] for the Z notation. The interested reader is also referred to [135] for comprehensive information on Z. A set may be defined in Z using set specification in comprehension. This is of the form $\{ D \mid P \bullet E \}$, where D represents declarations, P is a predicate and E an expression. The components of $\{ D \mid P \bullet E \}$ are the values taken by expression E when the variables introduced by D take all possible values that make the predicate P true. For example, the set of squares of all even natural numbers is defined as $\{ n : \mathbb{N} \mid (n \mathsf{~mod~} 2) = 0 \bullet n^2 \}$. When there is only one variable in the declaration and the expression consists of just that variable, then the expression may be dropped if desired. For example, the set of all even numbers may be written as $\{ n : \mathbb{N} \mid (n \mathsf{~mod~} 2) = 0 \}$. Sets may also be defined in display form such as $\{1, 2\}$.

In Z, relations and functions are represented as sets of pairs. A (binary) relation R, declared as having type $A \leftrightarrow B$, is a component of $\mathbb{P}(A \times B)$. For $a \in A$ and $b \in B$, then the pair (a, b) is written as $a \mapsto b$, and $a \mapsto b \in R$ means that a is related to b under relation R. Functions are treated as special forms of relations. The schema notation is used to structure specifications. A schema such as $\mathcal{FW}_1$ defines a collection of variables (limited to the scope of the schema) and specifies how they are related. The variables can be introduced via schema inclusion, as done, for example, in the definition of sequential composition.

$\mathbb{N}$	The natural numbers
$\mathbb{P}\, A$	The power set of A
$A \leftrightarrow B$	Relations between A and B
$S \lhd R$	Domain restriction
$S \rhd R$	Range restriction
R^+	Transitive closure
$A \rightarrow B$	Total functions from A to B
$A \pfun B$	Partial functions from A to B
$A \times B$	The Cartesian product of A and B
$X ::= A \mid B\langle\!\langle C \rangle\!\rangle$	Free-type definition
$X == a$	Abbreviation definition
$x = y$	Equality
$x \in S$	Membership
$\emptyset$	Empty set
$\subseteq$	Subset
$\supseteq$	Superset
$\cup$	Set union
$\cap$	Set intersection
$\setminus$	Set difference
$\#$	Cardinality
$\neg\ S$	Negation
$\land$	Conjunction
$\lor$	Disjunction
$\Leftrightarrow$	Equivalence
$\forall\, S \bullet E$	Universal quantification
$\bigcup S$	Generalized union
$\bigcap S$	Generalized intersection
$\langle X \rangle$	Sequence display
$\frown$	Concatenation
$\frown/$	Distributed concatenation
$\mathit{head}\, s$	First element of a sequence
$\mathit{tail}\, s$	All but the *head* of a sequence

Appendix B

Supplementary Android Threat-model Information

B.1 Best Practice Extracts and Android Security States

Table B.1 gives an extract of NIST-800-124: Guidelines on Cell Phone and PDA Security.

ID	Recommendation Description	
CPhBP-1	"*Install and configure additional security controls that are required, including ... remote content erasure*" [79].	
	Threat	**Countermeasure**

	No intended remote erasure whitelist	`iptables -A INPUT -p tcp --sport $port -j ACCEPT`
		`iptables -A OUTPUT -p tcp --dport $port -j ACCEPT`

ID	**Recommendation Description**	
CPhBP-2	"*Curb Wireless Interfaces: turn off Bluetooth, Wi-Fi, infrared, and other wireless interfaces until they are needed.* " [79].	
	Threat	**Countermeasure**
	Inbound local spurious iface traffic	`iptables -A INPUT -i $iface -j DROP`
	Outbound local spurious iface traffic	`iptables -A OUTPUT -o $iface -j DROP`
	Inbound forward spurious iface traffic	`iptables -A FORWARD -i $iface -j DROP`
	Outbound forward spurious iface traffic	`iptables -A FORWARD -o $iface -j DROP`
ID	**Recommendation Description**	
CPhBP-4	"*Network Access - Malware resident on the device is able to use the device for one or more unauthorized network activities, including port scanning or using the device as a proxy for network communications*" [79].	
	Threat	**Countermeasure**
	Outbound local Malware IP Pkt dropped using default drop as a catch all	`iptables -P OUTPUT DROP`

Table B.1: Extract of NIST-800-124

Table B.2 gives an extract of NIST-800-153: Guidelines for Securing Wireless Local Area Networks.

ID	Recommendation Description	
WiFiBP-1	“*For all their WLAN client devices not authorized for dual connections: Implement the appropriate technical security controls … so that all dual connected configurations are prohibited.*” [133].	
	Threat	**Countermeasure**
	Inbound local spurious iface traffic	`iptables -A INPUT -i $ifaceToDisable -j DROP`
	Outbound local spurious iface traffic	`iptables -A OUTPUT -o $ifaceToDisable -j DROP`
	Inbound forward Spurious iface traffic	`iptables -A FORWARD -i $ifaceToDisable -j DROP`
	Outbound forward Spurious iface traffic	`iptables -A FORWARD -o $ifaceToDisable -j DROP`
ID	**Recommendation Description**	
WiFiBP-2	Logging:“*often useful for both periodic assessments and continuous monitoring.*” [133].	
	Threat	**Countermeasure**
	No inbound local audit control	`iptables -A INPUT -j LOG --log-level 7`
	No inbound forward audit control	`iptables -A FORWARD -i $iface -j LOG --log-level 7`

Table B.2: Extract of NIST-800-153

Table B.3 gives an extract of NIST-800-41: Guidelines on Firewalls & Firewall Policy.

ID	Recommendation Description	
FBP-1	Deny “*Inbound or Outbound network traffic containing a source or destination address of 0.0.0.0.*” [152].	
	Threat	**Countermeasure**
	Inbound Local 0.0.0.0/8 Src IP Pkt	`iptables -A INPUT -s 0.0.0.0/8 -j DROP`
	Outbound local 0.0.0.0/8 Src IP Pkt	`iptables -A OUTPUT -s 0.0.0.0/8 -j DROP`
	Inbound Forward 0.0.0.0/8 Src IP Pkt	`iptables -A FORWARD -i $iface -s 0.0.0.0/8 -j DROP`
	Outbound forward 0.0.0.0/8 Src IP Pkt	`iptables -A FORWARD -o $iface -s 0.0.0.0/8 -j DROP`
ID	**Recommendation Description**	
FBP-2	“*Content filtering … virus scanning, filtering, and removal*” [152].	
	Threat	**Countermeasure**
	Inbound local unfiltered traffic	`iptables -A INPUT -m -string --algo bm --string ‘$filterString’ -j DROP`
	Inbound forward unfiltered traffic	`iptables -A FORWARD -i $iface -m -string --algo bm --string ‘$filterString’ -j DROP`
ID	**Recommendation Description**	
FBP-3	Implement stateful rules where possible as “*stateful inspection firewalls are generally considered to be more secure than packet filter firewalls.*” [152].	
	Threat	**Countermeasure**

	No whitelist application communication	`iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT`
		`iptables -A OUTPUT -m owner --uid-owner $appUID state --state NEW,ESTABLISHED, RELATED -j ACCEPT`

Table B.3: Extract of NIST-800-41

Table B.4 gives an extract of NIST-800-41-Rev1: Guidelines on Firewalls & Firewall Policy.

ID	Recommendation Description	
FBPr1-1	"*deny by default policies should be used for incoming TCP and UDP traffic.*" [125].	
	Threat	**Countermeasure**

	No inbound default deny policy
	No outbound default deny policy
	No forward default deny policy
		`iptables -P INPUT DROP`
		`iptables -P OUTPUT DROP`
		`iptables -P FORWARD DROP`

ID	**Recommendation Description**	
FBPr1-2	"*. . . an invalid source address for incoming traffic or destination address for outgoing traffic . . . should be blocked*" that is "*An IPv4 address within the ranges in RFC 1918*" and "*An address that is not in an . . . IANA . . . range*" [123]	
	Threat	**Countermeasure**
	Inbound local 192.168.0.0/16 Src IP Pkt	`iptables -A INPUT -s 192.168.0.0/16 -j DROP`
	Outbound local 192.168.0.0/16 Dst IP Pkt	`iptables -A OUTPUT -d 192.168.0.0/16 -j DROP`
	Inbound forward 192.168.0.0/16 Src IP Pkt	`iptables -A FORWARD -i $iface -s 192.168.0.0/16 -j DROP`
	Outbound forward 192.168.0.0/16 DstIP Pkt	`iptables -A FORWARD -o $iface -d 192.168.0.0/16 -j DROP`
	Inbound local 10.0.0.0/8 Src IP Pkt	`iptables -A INPUT -s 10.0.0.0/8 -j DROP`
	Outbound local 10.0.0.0/8 Dst IP Pkt	`iptables -A OUTPUT -d 10.0.0.0/8 -j DROP`
	Inbound forward 10.0.0.0/8 Src IP Pkt	`iptables -A FORWARD -i $iface -s 10.0.0.0/8 -j DROP`
	Outbound forward 10.0.0.0/8 DstIP Pkt	`iptables -A FORWARD -o $iface -d 10.0.0.0/8 -j DROP`
	Inbound local 172.16.0.0/12 Src IP Pkt	`iptables -A INPUT -s 172.16.0.0/12 -j DROP`
	Outbound local 172.16.0.0/12 Dst IP Pkt	`iptables -A OUTPUT -d 172.16.0.0/12 -j DROP`
	Inbound forward 172.16.0.0/12 Src IP Pkt	`iptables -A FORWARD -i $iface -s 172.16.0.0/12 -j DROP`
	Outbound forward 172.16.0.0/12 Dst IP Pkt	`iptables -A FORWARD -o $iface -d 172.16.0.0/12 -j DROP`
ID	**Recommendation Description**	
FBPr1-3	"*Organizations should also block . . . IP source routing information*" [123]	
	Threat	**Countermeasure**
	SSRR firewall bypass.	`iptables -A FORWARD -m ipv4options --ssrr -j DROP`
	LSRR firewall bypass.	`iptables -A FORWARD -m ipv4options --lsrr -j DROP`
ID	**Recommendation Description**	
FBPr1-4	"*Organizations should also block . . . directed broadcast addresses*" [123]	
	Threat	**Countermeasure**
	Inbound local directed broadcast	`iptables -A INPUT -d x.x.x.255 -j DROP`
	Outbound local directed broadcast	`iptables -A OUTPUT -d x.x.x.255 -j DROP`
	Inbound forward directed broadcast	`iptables -A FORWARD -i $iface -d x.x.x.255 -j DROP`
	Outbound forward directed broadcast	`iptables -A FORWARD -o $iface -d x.x.x.255 -j DROP`
ID	**Recommendation Description**	
FBPr1-5	To limit Denial of Service "*a firewall might redirect the connections made to a particular inside address to a slower route if the rate of connections is above a certain threshold.*" [123]	
	Threat	**Countermeasure**
	Inbound forward DoS to tethered device	`iptables -A FORWARD -i $iface -d $lanIP -m limit --limit $x/s --limit-burst $y -j ACCEPT`

Table B.4: Extract of NIST-800-41-Rev1

Table B.5 gives an extract of NIST-800-114: User's Guide to Securing External Devices for Telework & Remote Access, and Table B.6 gives an extract of NIST-800-163: Vetting the Security of Mobile Applications.

ID	**Recommendation Description**	
TBP-1	Construct an access control whitelist of locally hosted applications trusted for telework network access: "*teleworkers should install and use only trusted software*" [125].	
	Threat	**Countermeasure**
	Inbound local application whitelist traffic not permitted	`iptables -A INPUT -m owner --uid-owner $appUID -m state --state ESTABLISHED,RELATED -j ACCEPT`
	Outbound local application whitelist traffic not permitted	`iptables -A OUTPUT -m owner --uid-owner $appUID -m state --state NEW,ESTABLISHED,RELATED -j ACCEPT`
ID	**Recommendation Description**	
TBP-2	... "*silently ignore unsolicited requests sent to it, which essentially hides the device from malicious parties.*" [125].	
	Threat	**Countermeasure**
	ICMP ping network scan	`iptables -A INPUT -p icmp -j DROP`
	TCP XMAS network scan	`iptables -A INPUT -p tcp --tcp-flags ALL ALL -j DROP`
	TCP Null network scan	`iptables -A INPUT -p tcp --tcp-flags ALL NONE -j DROP`
	TCP Syn Fin network scan	`iptables -A INPUT -p tcp --tcp-flags SYN,FIN SYN,FIN -j DROP`
	TCP Rst Fin network scan	`iptables -A INPUT -p tcp --tcp-flags FIN,RST FIN,RST -j DROP`

	TCP Port 0 network scan	`iptables -A INPUT -p tcp --dport 0 -j DROP`
		`iptables -A INPUT -p tcp --sport 0 -j DROP`

ID	**Recommendation Description**	
TBP-3	"*Use a different brand of Web browser for telework*" [125].	
	Threat	**Countermeasure**
	Regular Web browser usage	`iptables -A OUTPUT -p tcp --dport 80 -m owner --uid-owner $untrustedHTTPUID -j DROP`
	Intended telework Web browser usage not permitted	`iptables -A OUTPUT -p tcp --dport 80 -m owner --uid-owner $trustedHTTPUID state --state NEW,ESTABLISHED -j ACCEPT`
ID	**Recommendation Description**	
TBP-4	"*Configuring primary applications to filter content and stop other activity that is likely to be malicious*" [125]	
	Threat	**Countermeasure**
	Outbound local unfiltered traffic	`iptables -A OUTPUT -m -string --algo bm --string ‘$filterString’ -j DROP`
ID	**Recommendation Description**	
TBP-5	"*Personal firewalls should be configured to log significant events, such as blocked and allowed activity*" [125]	
	Threat	**Countermeasure**
	No inbound local audit control	`iptables -A INPUT -j LOG --log-level 7`
	No inbound forward audit control	`iptables -A FORWARD -i $iface -j LOG --log-level 7`

Table B.5: Extract of NIST-800-114

ID	**Recommendation Description**	
PBP-1	"*Apps should have only the minimum permissions necessary and should only grant other applications the necessary permissions*" [125].	
	Threat	**Countermeasure**
	An overly-promiscuous application/permission configuration	Blacklist unnecessary application/permission pairs

Table B.6: Extract of NIST-800-163

State	Interface	Network Connection	Risk Appetite	Teleworking	Data Quota	Battery	Spoofing	DoS	Scanning	Source Routing	Malicious Content	Promiscuity Level	Non-Audit	Permission Promiscuity Level
state-1	wifi	trusted	averse	true	false	hi	x	x	x		x	x	x	x
state-2	wifi	trusted	averse	true	false	lo	x	x	x		x	x	x	x
state-3	wifi	trusted	averse	false	false	hi	x	x	x		x	x		
state-4	wifi	trusted	averse	false	false	lo						x	x	x
state-5	wifi	trusted	hungry	true	false	hi	x	x	x		x	x	x	
state-6	wifi	trusted	hungry	true	false	lo	x	x	x		x	x	x	x
state-7	wifi	trusted	hungry	false	false	hi						x		
state-8	wifi	trusted	hungry	false	false	lo						x		
state-9	wifi	untrusted	averse	true	false	hi	x	x	x		x	x	x	x
state-10	wifi	untrusted	averse	true	false	lo	x	x	x		x	x	x	x
state-11	wifi	untrusted	averse	false	false	hi	x	x	x		x	x	x	
state-12	wifi	untrusted	averse	false	false	lo	x	x	x		x	x		x
state-13	wifi	untrusted	hungry	true	false	hi	x	x	x		x	x	x	x
state-14	wifi	untrusted	hungry	true	false	lo	x	x	x		x	x	x	x
state-15	wifi	untrusted	hungry	false	false	hi						x		
state-16	wifi	untrusted	hungry	false	false	lo						x		
state-17	3g	trusted	averse	true	false	hi	x	x	x		x	x	x	x
state-18	3g	trusted	averse	true	false	lo	x	x	x		x	x	x	x
state-19	3g	trusted	averse	false	false	hi	x	x	x		x	x		
state-20	3g	trusted	averse	false	false	lo						x		x
state-21	3g	trusted	hungry	true	false	hi	x	x	x		x	x	x	x
state-22	3g	trusted	hungry	true	false	lo	x	x	x		x	x	x	x
state-23	3g	trusted	hungry	false	false	hi						x		
state-24	3g	trusted	hungry	false	false	lo						x		
state-25	3g,wifi	trusted	averse	false	false	hi	x	x	x	x	x	x		x
state-26	3g,wifi	trusted	averse	false	false	lo					x	x		x
state-27	3g,wifi	trusted	hungry	false	false	hi						x		
state-28	3g,wifi	trusted	hungry	false	false	lo						x		
state-29	3g	trusted	averse	true	true	hi	x	x	x		x	x	x	x
state-30	3g	trusted	averse	true	true	lo	x	x	x		x	x	x	x
state-31	3g	trusted	averse	false	true	hi	x	x	x		x	x		
state-32	3g	trusted	averse	false	true	lo						x		x
state-33	3g	trusted	hungry	true	true	lo	x	x	x		x	x	x	x
state-35	3g	trusted	hungry	false	true	hi						x		
state-36	3g	trusted	hungry	false	true	lo						x		
state-37	3g,wifi	trusted	averse	false	true	hi	x	x	x	x	x	x		
state-38	3g,wifi	trusted	averse	false	true	lo					x	x		x
state-39	3g,wifi	trusted	hungry	false	true	hi						x		
state-40	3g,wifi	trusted	hungry	false	true	lo						x		

Table B.7: Matrix of valid security states for MASON

Index