The Dissertation Committee for Rezwana Reaz
certifies that this is the approved version of the following dissertation:

Theory and Practice of Firewall Outsourcing

Committee:

Mohamed G. Gouda, Supervisor

Aloysius K. Mok

Lili Qiu

Hrishikesh B. Acharya

Theory and Practice of Firewall Outsourcing

by

Rezwana Reaz

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2019

Dedicated to my parents and my daughter.

Acknowledgments

My Phd journey at UT Austin has been a long journey of six and half years. I am thankful to many great people who helped me make this journey happen.

I would like to express my deep gratitude to my supervisor Prof. Mohamed G. Gouda for believing in me and giving me the opportunity to work under his invaluable guidance. I am very grateful to him for introducing me to the area of formal analysis of network firewalls, which is a perfect blend of theory and systems. As I mainly worked on theoretical computer science during my undergraduate and masters theses and was willing to work in systems during my PhD, this area matched both my expertise and interest. Having the privilege of working with Prof. Gouda, who has years of experience in systems and formal methods, has been a great learning experience for me where I have learned to appreciate the value and importance of formal methods in systems. He always showed great enthusiasm about my work. He encouraged and appreciated every little contribution I made towards my dissertation. He was always there to help overcome every single difficulty I faced. His attention to detail always helped me find better ways to shape my work.

I owe a special thanks to my committee members: Prof. Lili Qiu, Prof. Aloysius K. Mok, and Prof. Hrishikesh B. Acharya for agreeing to serve on

my committee, and providing useful feedback. It has been a great honor to have them in my doctoral committee. I am also very grateful to Prof. Ehab. S. Elmallah for his collaboration and contribution in my research. His careful observations and practical suggestions helped greatly to improve the quality of my work. I also want to thank Prof. Marijn J. H. Heule for his collaboration in one of my early work during PhD. He introduced me to various techniques of SAT-based approach of analyzing firewalls. I am also thankful to all my colleagues in Prof. Gouda's Lab, especially to Muqeet Ali. He helped me get familiar with my research field when I joined Prof. Gouda's Lab and collaborated with me in research for the first two years of my PhD.

I must thank Prof. Tandy Warnow who was my first PhD supervisor. I could work with her only in my very first semester at UT Austin, as she left UT Austin the next semester. It was a very short but a wonderful journey with her. Her expertise and art of supervising brought out the best out of me even within just one semester. The work I had done with her during that short period resulted in one of my most cited papers.

I also want to extend my gratitude to my undergraduate and master's thesis supervisor Prof. M. Sohel Rahman. I am indebted to him for preparing me for the PhD journey abroad. His faith in me and his moral support encouraged me a lot to apply for a PhD program at a prestigious university in USA.

I would like to thank all those people whom I met during my PhD life outside the boundary of my Lab, who encouraged me and helped me get

through the challenging grad life with their kind words and sensible actions. Time would fail me to name them all.

My Phd journey would not be very smooth without the support of my entire family. I thank them all from the bottom of my heart. I would like to start by thanking my three and half year old daughter Zunairah. She was born during the third year of my PhD. As soon as she turned one, my husband, who was also a Phd student that time at UT Austin, graduated and left USA for his job back home. Although at first it felt like I would not be able make progress in my dissertation with a young child and for not having my husband around while I was already in the middle of all sorts of uncertainty of Phd life, it went quite smooth as time had passed. A special credit goes to my wonderful daughter, who is very sensible and understanding towards my work. I am also deeply thankful to my husband Bayzid, who had to stay away from us and miss a lot of milestones of toddler Zunairah, for his unconditional support and patience. I want to express my deep gratitude towards my parents, who have always helped me in the best possible way they can. I would never be able to make progress in my dissertation if they weren't there to take care of my daughter while my husband was away. I am indebted to them for their love and support. I also would like to thank my sister, Rumana, for always being there to boost up my confidence whenever I suffered lack of confidence. I want to extend my gratitude to my parents-in-laws and other members of my in-laws family for supporting and appreciating my work.

Finally, I deeply express my sincere gratitude to the endless kindness of

almighty Allah. His guidance and generosity, made it possible for me to carry out my research and complete my dissertation.

Theory and Practice of Firewall Outsourcing

Publication No. _____

Rezwana Reaz, Ph.D.
The University of Texas at Austin, 2019

Supervisor: Mohamed G. Gouda

A firewall system is a packet filter that is placed at the entry point of an enterprise network in the Internet. Packets that attempt to enter the enterprise network through this entry point are examined, one by one, against the rules of some underlying firewall F of the firewall system. Each rule in F has a decision which is either "accept" or "reject". For any incoming packet p, the firewall system identifies the first rule (in the sequence of rules in F) that matches p. If the decision of this rule is "accept", then the firewall system forwards p to the enterprise network. Otherwise the decision of this rule is "reject" and packet p is discarded and prevented from entering the network.

Each firewall system consists of two units: a rule matching unit and a decision unit. Both units are usually executed in the firewall system. To simplify the task of managing the firewall system, we identify a special class of firewall systems, called the outsourced system, where the rule matching unit is executed in a public cloud. Unfortunately, public clouds are usually unreliable

and execution of the rule matching unit in a public cloud can be vulnerable to two types of attacks: verifiability attacks and privacy attacks.

The main objective of this dissertation is to discuss how to execute the rule matching unit of an outsourced system in a public cloud such that verifiability and privacy attacks are prevented from occurring. The main contribution of this dissertation is three-fold.

First, we discuss how to design outsourced firewall system such that execution of the designed system in the public clouds prevents the occurrence of verifiability and privacy attacks. The resulting system, called the private system, make use of two public clouds. We show that this private system prevents verifiability and privacy attacks under the assumption that the two public clouds used in this system are both "sensible" and "non-colluding".

Second, we identify a special class of firewalls, called the partially specified firewall, where a firewall is called partially specified when the decisions of some of the rules in the firewall are not specified as "accept" or "reject". We show that for every partially specified firewall PF, there is a (fully specified) firewall F such that PF and F are equivalent. We discuss how to design an outsourced system whose underlying firewall is a partially specified firewall PF such that the designed system prevents both verifiability and privacy attacks. We achieve this outsourced system by obtaining an equivalent firewall F from PF and designing a private system for F.

Third, we present a generalization of firewalls called firewall expres-

sions. A firewall expression is specified using one or more component firewalls and three firewall operators: "not", "and", and "or". For example, the firewall expression (G and H) consists of two component firewalls G and H and one firewall operator "and". This firewall expression accepts a packet p iff both firewalls G and H accept p. For any underlying firewall expression FE, we define an *Expression System* as a generalization of firewall systems that takes as input any packet p and determines whether the underlying firewall expression FE accepts or rejects packet p.

We design an outsourced expression system for any underlying firewall expression FE. We achieve this outsourced expression system by using a private system for each component firewall of FE and combining these private systems through an overall decision unit to determine whether any packet is accepted or rejected according to the firewall expression FE.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

A firewall system is a packet filter that is placed at the entry point of an enterprise network in the Internet. The function of the firewall system is to examine the packets that attempt to enter the enterprise network through the entry point, identify malicious packets, and prevent the malicious packets from entering the network. Thus, a firewall system is a critical component in the security of an enterprise network.

Each firewall system is built on top of an underlying firewall F. A firewall F is a sequence of rules where each rule consists of a sequence number, a predicate, and a decision. The sequence number of each rule is a unique integer in the range from 1 to n, where n is the number of rules in F. The predicate of each rule is defined using t attributes u_1, u_2, ..., and u_t. The decision of each rule is either "accept" or "reject".

Packets that attempt to enter the enterprise network through the entry point are examined, one by one, against the rules of the underlying firewall F of the firewall system. Examining a packet against the rules of the underlying firewall F, the firewall system determines whether to allow the packet to be accepted and forwarded to the enterprise network or to be rejected and

prevented from entering the network. A packet p is accepted (or rejected, respectively) by the underlying firewall F iff the decision of the first rule in F that matches p is accept (or reject, respectively).

In this dissertation, we study the execution of firewall systems using public clouds. Executing firewall system using public clouds can simplify the task of managing the firewall system for an enterprise network. Each firewall system consists of two units: a rule matching unit and a decision unit. Typically, both the rule matching unit and the decision unit of the firewall system are executed by the system itself. In this dissertation, we are interested to investigate a class of firewall systems, called outsoured firewall systems, whose rule matching units are executed by public clouds. Unfortunately, public clouds can be unreliable causing outsourced systems to be vulnerable against two types of attacks: verifiability attacks and privacy attacks. In this dissertation, we explore different designs of outsourced firewall systems with an objective that the designed system takes advantage of public clouds and at the same time prevents the verifiability and privacy attacks from occurring.

1.1 Firewall Systems

For any firewall F, we can define a *Firewall System* that takes as input any packet p and determines whether packet p is accepted or rejected according to the rules in F. In this case, we call firewall F the *underlying firewall* of the firewall system. The architecture of a firewall system is presented in Figure 1.1. This system consists of two units: a rule matching unit and a decision unit.

Both the rule matching unit and the decision unit are built on top of the underlying firewall F.

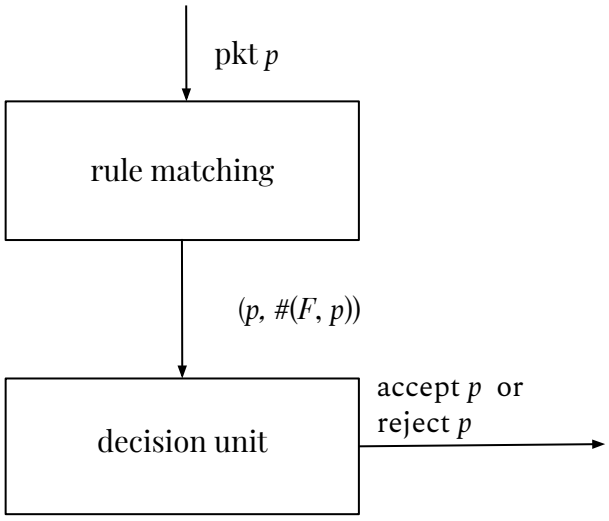

Figure 1.1: The architecture of a firewall system

When a packet p attempts to pass this firewall system, p is first directed to the rule matching unit. The task of the rule matching unit is to determine the sequence number of the first rule in F that matches p and send this sequence number to the decision unit. In Figure 1.1, the notation $\#(F, p)$ denote the sequence number of the first match rule in firewall F for packet p. The task of the decision unit is to determine the decision of the rule whose sequence number is $\#(F, p)$.

If the first match rule in F for p has a decision "accept", then the decision unit forwards p to the enterprise network. Otherwise, the first match

rule in F for p has a decision "reject", and in this case the decision unit discards packet p and prevents it from entering the firewall system.

1.2 Firewall Outsourcing

Traditionally a firewall systems is designed and implemented such that all the tasks of the firewall system are executed by the system itself. A comparatively newer approach is to design and implement a firewall system such that some tasks of the firewall system are implemented and executed in public clouds. The approach of implementing and executing part of a firewall system in public clouds is called *firewall outsourcing*. Such a firewall system is called an *outsourced firewall system*.

In recent years, with the rise of cloud computing, enterprises have become interested in implementing and managing their firewall systems by using public clouds to reduce the associated cost and management complexity [37, 58], and [69]. According to a survey of firewall systems, firewall outsourcing can provide three benefits [61]. First, reduces the initial investment and the operational cost of the firewall system by taking advantage of the pay-per-use model of the cloud. Second, reduces the number or staff needed to manage and implement the firewall system. Third, increases availability of the firewall system by maintaining necessary back-ups.

Despite these benefits of firewall outsoutcing, an outsourced system can become vulnerable to security attacks caused by the fact that public clouds are usually unreliable.

In this dissertation, we identify a class of outsourced systems whose rule matching units are executed in public clouds. We obtain this outsourced system from the firewall system presented in Figure 1.1 by executing the rule matching unit in a public cloud C. Since cloud C is unreliable, the outsourced system is vulnerable to two types of security attacks: verifiability attacks and privacy attacks.

The verifiability attacks, caused by cloud C can be described as follows. When cloud C executes the rule matching unit of the outsourced system and computes the sequence number of the first match rule in the underlying firewall F for an incoming packet p, C may compute a wrong value. In particular, C may compute a sequence number v of a match rule (not the first match rule) in F for p and send the wrongly computed sequence number v to the decision unit. In this case, the decision unit will accept or reject packet p according to the decision of the rule whose sequence number is v in F. As a result, the decision unit may end up incorrectly accepting a packet instead of rejecting it or may end up incorrectly rejecting a packet instead of accepting it.

The privacy attacks, caused by cloud C can be described as follows. If C knows the rules of the underlying firewall F, C may leak F to potential attackers of the system.

Our goal in this dissertation is to design outsourced firewall systems by taking advantage of public clouds, such that verifiability and privacy attacks are prevented from occurring.

1.3 Limitation of Prior Outsourced Systems

Several outsourced firewall systems, for example, the systems in [20, 37, 42, 43, 60, 62, 68, 75], and [49], have been presented in the literature that take advantage of one or more public clouds.

An important limitation of all prior outsourced systems is that none of the systems defends against both verifiability and privacy attacks. We can divide the prior outsourced systems into three categories as follows.

1. Systems that defend only against verifiability attacks.
2. Systems that defend only against privacy attacks.
3. Systems that do not defend against verifiability and privacy attacks.

In the firewall systems presented in [20, 75], and [76], the rules of the underlying firewall F are stored in the clear in the cloud. Each incoming packet to the enterprise network is directed in the clear to the cloud. For each incoming packet p, the cloud determines whether to accept or reject p according to the rules of the underlying firewall F which are stored in the cloud. If the cloud determines to accept p, then the cloud forwards p to the entry point of the enterprise network. Then the firewall systems in [20, 75], and [76] verify that packet p is indeed accepted according the underlying firewall F. Therefore, these firewall systems defend against verifiability attacks.

Whereas the firewall systems in [75] and [76] execute the verification steps online, the firewall system in [20] executes the verification steps offline.

Moreover, because the rules of the underlying firewall F are stored in the clear in the cloud, the cloud can leak these rules to potential attackers of the system. Therefore, the firewall systems in [20, 75], and [76] defends against verifiability attacks but do not defend against privacy attacks.

In the firewall systems presented in [37, 42, 43, 60, 62, 68], and [49], the rules of the underlying firewall F are encrypted before they are stored in the cloud. Each incoming packet to the enterprise network is directed to the cloud. For each incoming packet p, the cloud determines whether to accept or reject p according to the encrypted rules of the underlying firewall F which are stored in the cloud. If the cloud determines to accept p, then the cloud forwards p to the entry point of the enterprise network. Because the rules of the underlying firewall F which are stored in the cloud are encrypted, the cloud cannot know the rules of the underlying firewall F and so cannot leak these rules to potential attackers of the system.

However, none of these firewall systems verifies that packet p that has been forwarded to the entry point of the enterprise network from the cloud is indeed accepted according to the underlying firewall F. Therefore the firewall systems in [37, 42, 43, 60, 62], and [68] defend against privacy attacks but do not defend against verifiability attacks.

The outsourced systems in [25] and [61] are designed assuming that public clouds are reliable. Thus, in these systems the rules of the underlying firewall F are stored in the clear in the cloud. Each incoming packet to the enterprise network is directed in the clear to the cloud. For each incoming

packet p, the cloud determines whether to accept or reject p according to the rules of the underlying firewall F which are stored in the cloud. These firewall systems do not verify that packet p is indeed accepted or rejected according the underlying firewall F. Moreover, because the rules of the underlying firewall F are stored in the clear in the cloud, the cloud can leak these rules to potential attackers of the system. Therefore, these systems do not defend against verifiability and privacy attacks.

1.4 Our Contributions

In this dissertation, we present different designs of outsourced systems such that the designed system takes advantage of public clouds but prevents the occurrence of verifiability and privacy attacks. There are three main contributions in this dissertation.

The first contribution in this dissertation is to discuss how to design an outsourced firewall system whose rule matching units are executed in public clouds such that verifiability and privacy attacks cannot occur. The resulting outsourced system, called the private system, makes use of two public clouds in order to execute the rule matching units. We show that this private system prevents verifiability and privacy attacks under the assumption that the two public clouds used in this system are both "sensible" and "non-colluding".

Our second contribution is to design an outsourced system for a special class of firewalls, called the partially specified firewall. A firewall is called partially specified when the decisions of some of the rules in the firewall are not

specified as "accept" or "reject". We show that for every partially specified firewall PF, there is a (fully specified) firewall F such that PF and F are equivalent. We discuss how to design an outsourced system whose underlying firewall is a partially specified firewall PF such that the designed system prevents both verifiability and privacy attacks. We achieve this outsourced system by obtaining an equivalent firewall F from PF and designing a private system for F.

In this dissertation, our third contribution is to present a generalization of firewalls into firewall expressions and design and outsourced systems for firewall expressions. A firewall expression is specified using one or more component firewalls and the three firewall operators: "not", "and", and "or". For example, a firewall expression ((G and H) or not(G)) consists of two component firewalls G and H and the three firewall operators "and", "or", and "not". This firewall expression accepts a packet p iff both firewalls G and H accept p or firewall G rejects p.

For any underlying firewall expression, we define an *Expression System* as a generalization of firewall systems that takes as input any packet p and determines whether the underlying firewall expression accepts packet p or rejects p.

We design an outsourced expression system for any underlying firewall expression FE. We achieve this outsourced expression system by using a private system for each component firewall of FE and combining these private systems through an overall decision unit to determine whether any packet p is

accepted or rejected according to the firewall expression FE.

1.5 Organization of the Dissertation

The rest of this dissertation is organized as follows.

In Chapter 2, we present basic concepts related to firewalls and perform a literature review on firewalls. We divide the literature on firewalls into five categories: firewall design, firewall analysis, property verification of firewalls, packet classification and firewall outsourcing. A brief survey of the research that have been conducted in the first four categories are presented in Section 2.3. We review the research works that fall in the category of firewall outsourcing in Section 2.4.

In Chapter 3, we show how to design an outsourced system whose rule matching unit is executed in a public cloud such that the resulting system prevents verifiability and privacy attacks. We discuss execution of an outsourced system in Section 3.2 and formally specify verifiability and privacy attacks in Section 3.3. We design an outsourced system, called the verifiable system that prevents verifiability attacks in Section 3.4. In Section 3.5, we modify the verifiable system to a system, called the private system that prevents occurrence of both attacks.

Chapter 4 presents a special class of firewalls, called partially specified firewalls. This chapter proceeds by first presenting several definitions such as definition of partially specified firewalls, definition of a packet being accepted

or rejected by a partially specified firewall and so on. Then, we show that every partially specified firewall is equivalent to a (fully specified) firewall. Finally, we show how to design an outsourced system when the underlying firewall is partially specified.

In Chapter 5, we introduce a generalization of firewalls, called firewall expressions. In Section 5.2, we present formal definition of firewall expressions and discuss three theorems that state fundamental properties of firewall expressions. In Section 5.3, we discuss an algorithm that can be used to evaluate a given firewall expression for any input stream of packets. Sections 5.4 and 5.5 present the logical analysis to determine whether the given firewall expressions satisfy some logical properties such as adequacy, implication, and equivalence.

Chapter 6 presents a generalization of firewall systems, called the expression systems that accepts or rejects incoming packets based on an underlying firewall expression. The architecture of an expression system in presented in Section 6.2. We design an outsourced expression system using public clouds in Section 6.3. We describe the execution of our designed outsourced expression system in Section 6.4. We discuss in Section 6.5 that verifiability and privacy attacks cannot occur in the designed outsourced expression system.

We conclude this dissertation in Chapter 7. In this chapter, we identify some open research problems related to firewall outsourcing and shed some light on how to approach some of these open problems by taking advantage of the outsourcing techniques presented in this dissertation.

Chapter 2

Firewalls

The function of the firewall system of an enterprise network is to identify malicious packets that aim to attack the enterprise network and prevent these packets from entering the network. Packets that attempt to enter the enterprise network through the entry point are examined, one by one, by the firewall system that is placed at the entry point. Examining a packet, the firewall system determines whether to allow the packet to proceed into the enterprise network or to be rejected and prevented from entering the network.

Each firewall system is built on top of an underlying firewall F. The firewall system determines whether to accept or reject an incoming packet according to the rules in F. We now present the formal definition of firewall F.

A firewall F is a sequence of rules where each rule is of the following form:

$$\langle \text{sequence number} \rangle \ \langle \text{predicate} \rangle \ \rightarrow \ \langle \text{decision} \rangle$$

Each rule in F consists of a sequence number, a predicate, and a decision. The sequence number of each rule is a unique integer in the range

from 1 to n, where n is the number of rules in F. The predicate of each rule is defined using t attributes u_1, u_2, ..., and u_t. The decision of each rule is either "accept" or "reject".

An example of a firewall F that consists of three rules is as follows.

$$
\begin{array}{lll}
1 & \big((u_1 \in [1,4]) \wedge (u_2 \in [8,9])\big) & \rightarrow \text{reject} \\
2 & \big((u_1 \in [2,4]) \wedge (u_2 \in [7,9])\big) & \rightarrow \text{accept} \\
3 & \big((u_1 \in [1,9]) \wedge (u_2 \in [1,9])\big) & \rightarrow \text{reject}
\end{array}
$$

Note that the predicate of each rule in this firewall F is defined using two attributes u_1 and u_2 whose integer values are taken from the integer interval [1, 9]. The first rule in F is called rule 1, the second rule in F is called rule 2, and so on.

A firewall can also be represented as a decision tree [7, 44, 74] or as a finite automata [41] instead of as a sequence of rules.

Now consider two packets p and q where each packet is defined as a tuple of two integers. Packet p is defined as the tuple ($u_1 = 3$, $u_2 = 7$) and packet q is defined as the tuple ($u_1 = 2$, $u_2 = 6$). Packet p does not match rule 1, but matches rule 2. So the first match rule in F for packet p is rule 2. Similarly, packet q does not match rule 1 and rule 2. Rather it matches rule 3. So the first match rule in F for packet q is rule 3.

When a packet p matches more than one rule in firewall F, the decision

of the first match rule is applied to p. For example, Because the first rule in F that matches q is rule 3 and because this rule has a decision "reject", packet q is rejected by firewall F.

2.1 Firewall Concepts

We now present formal definition for each of the following concepts: Attributes, Predicates, Rules, Packets, First Match Rule and Complete Firewalls.

Attributes

An attribute is a "variable" that has a "name" and a "value". We denote t attributes as u_1, u_2, ..., and u_t. The value of each attribute u_i is taken from an interval that is called the domain of attribute u_i and is denoted $D(u_i)$.

Predicates

A predicate is of the form $((u_1 \in X_1) \wedge \cdots \wedge (u_t \in X_t))$, where each u_i is an attribute, each X_i is an interval that is contained in the domain $D(u_i)$ of attribute u_i, and $\wedge$ is the logical AND or conjunction operator.

A predicate $((u_1 \in X_1) \wedge \cdots \wedge (u_t \in X_t))$, where each interval X_i is the whole domain of the corresponding attribute u_i, is called the ALL predicate.

Throughout this dissertation, we assume that the number of attributes, t, in each rule is a fixed value. More precisely, we assume that each rule in

any firewall is defined over five attributes: source IP address, destination IP address, source port number, destination port number, and transport protocol.

Rules

A rule in a firewall F is defined as a tuple, a sequence number, a predicate and a decision, written as follows:

$$\langle \text{sequence number} \rangle \ \langle \text{predicate} \rangle \ \rightarrow \ \langle \text{decision} \rangle$$

The first rule in F is called rule 1, the second rule in F is called rule 2 and so on.

We assume that there are two distinct decisions: "accept" and "reject". A rule whose decision is "accept" is called an accept rule, and a rule whose decision is "reject" is called a reject rule. An accept rule whose predicate is the ALL predicate is called an accept-ALL rule, and a reject rule whose predicate is the ALL predicate is called the reject-ALL rule.

Packets

A packet is a tuple $(b_1, \ldots, b_t)$ of t integers, where t is the number of attributes and each integer b_i is taken from the domain $D(u_i)$ of attribute u_i. We adopt P to denote the set of all packets. Note that set P is finite.

Matching Rule

A packet p is said to match a rule in a firewall F iff the packet matches the predicate of the rule.

First Match Rule

A rule r_i in a firewall F is called the *first match* rule in F for p iff the following two conditions hold:

- packet p matches rule r_i in F
- packet p does not match any of the rules r_1, ..., r_{i-1} in F

where, where $i \in \{1, \ldots, n\}$ and n is the number of rules in F.

We adopt the notation $\#(F, p)$ to denote the sequence number of the first match rule in firewall F for a packet p.

A firewall F is said to accept, or reject respectively, a packet p iff the rule whose sequence number is $\#(F, p)$ has a decision "accept" or "reject" respectively.

Complete Firewalls

A firewall F is complete iff every packet is either accepted by F or rejected by F. Throughout this dissertation, when we refer a firewall F, we mean a complete firewall F.

Let F be a firewall. We adopt the notation not(F) to denote the firewall that is obtained from firewall F by (1) replacing each "accept" decision in F by a "reject" decision in not(F) and (2) replacing each "reject" decision in F by an "accept" decision in not(F).

Note that a firewall F is complete iff the firewall not(F) is complete.

2.2 A Firewall Example

Consider the network shown in Figure 2.1. This network has a firewall system which is situated between the Internet and the enterprise network. The enterprise network consists of a mail sever with IP address 192.168.0.1 and two hosts: host 1 with IP address 192.168.0.2 and host 2 with IP address 192.168.0.3.

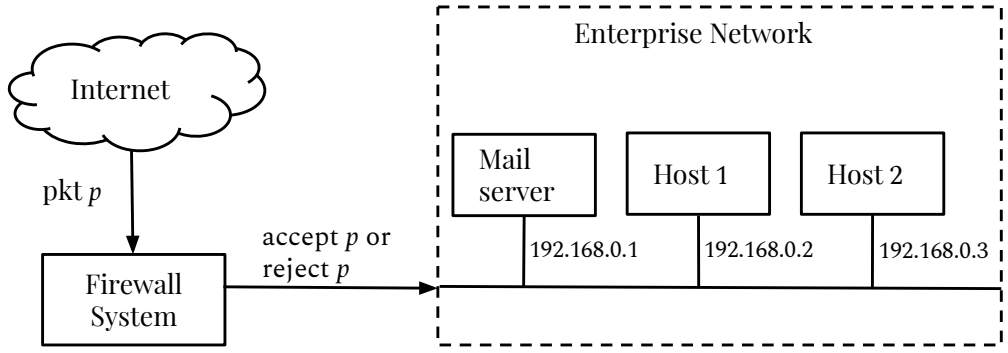

Figure 2.1: An example firewall system and an enterprise network

The firewall system in Figure 2.1 is built on top of an underlying firewall F. Suppose the requirement specification for F is given as follows.

1. The mail server, with IP address 192.168.0.1, can receive emails at port 25. Any other packet destined to the mail server is rejected.

2. Any packet originated from the malicious domain 172.23.0.0/16 destined to the mail server, host 1 and host 2 should be rejected.

3. Host 1 can only receive TCP packets.

4. Host 2 can receive both TCP and UDP packets.

In this example, we assume that each rule in F is defined over five attributes: source IP address (u_1), destination IP address (u_2), source port number (u_3), destination port number (u_4), and transport protocol (u_5).

Domain of these attributes are defined as follows. The domain u_1 and u_2 is the integer interval $[0,\ 2^{32}-1]$ and the domain of u_3 and u_4 is the integer interval [0, 100]. The domain of u_5 is the integer interval [0, 1] where 0 denotes that the transport protocol is UDP (user datagram protocol) and 1 denotes that the transport protocol is TCP (transmission control protocol).

Firewall F can be defined with the following rules that satisfy the above mentioned specification.

1 $(u_1 \in [172.23.0.0, 172.23.255.255]) \wedge (u_2 \in [192.168.0.1, 192.168.0.3])$
$\wedge\ (u_3 \in [0, 100]) \wedge (u_4 \in [0, 100]) \wedge (u_5 \in [0, 1]) \rightarrow$ reject

2 $(u_1 \in [0, 255.255.255.255]) \wedge (u_2 \in [192.168.0.1, 192.168.0.1])$
$\wedge\ (u_3 \in [0, 100]) \wedge (u_4 \in [25, 25]) \wedge (u_5 \in [1, 1]) \rightarrow$ accept

3 $(u_1 \in [0, 255.255.255.255]) \wedge (u_2 \in [192.168.0.1, 192.168.0.1])$
$\wedge\ ((u_3 \in [0, 100]) \wedge (u_4 \in [0, 100]) \wedge (u_5 \in [0, 1]) \rightarrow$ reject

4 $(u_1 \in [0, 255.255.255.255]) \wedge (u_2 \in [192.168.0.2, 192.168.0.2])$
$\wedge\ (u_3 \in [0, 100]) \wedge (u_4 \in [0, 100]) \wedge (u_5 \in [1, 1]) \rightarrow$ accept

5 $(u_1 \in [0, 255.255.255.255]) \wedge (u_2 \in [192.168.0.2, 192.168.0.2])$
$\wedge\ (u_3 \in [0, 100]) \wedge (u_4 \in [0, 100]) \wedge (u_5 \in [0, 1]) \rightarrow$ reject

6 $(u_1 \in [0, 255.255.255.255]) \wedge (u_2 \in [192.168.0.3, 192.168.0.3])$

$\wedge\ ((u_3 \in [0, 100]) \wedge (u_4 \in [0, 100]) \wedge (u_5 \in [0, 1]) \rightarrow \text{accept}$

7 $(u_1 \in [0, 255.255.255.255]) \wedge (u_2 \in [0, 255.255.255.255])$

$\wedge\ (u_3 \in [0, 100]) \wedge (u_4 \in [0, 100]) \wedge (u_5 \in [0, 1]) \rightarrow \text{reject}$

The meaning of each of these rules is as follows.

- Rule 1 corresponds to the second specification. Any packet originated from the malicious domain 172.23.0.0/16 is rejected.
- Rule 2 and 3 correspond to the first specification. Rule 2 says if a packet p has a destination IP 192.168.0.1 and destination port 25, then p is accepted. Rules 3 says if a packet p has a destination IP 192.168.0.1 but the destination port is not 25, then p is rejected.
- Rule 4 and 5 correspond to the third specification. Rule 4 says if a packet p has a destination IP 192.168.0.2 and the transport protocol is TCP, then p is accepted. Rule 4 says if a packet p has a destination IP 192.168.0.2 and it is not accepted by Rule 4, then p is accepted.
- Rule 6 corresponds to the forth specification. Rule 5 says if a packet p has a destination IP 192.168.0.3 and the transport protocol is TCP or UDP, then p is accepted.
- Rule 7 ensures that if any packet p is not accepted by any of rules 1, 2, 3, 4, 5, and 6, then p is rejected.

Now consider a packet p which is originated from a host with IP address 201.124.65.16 in the Internet and it is destined to the mail server in the enterprise network in Figure 2.1. Suppose p is defined as the tuple (201.124.65.16, 192.168.0.1, 90, 25, 0). When p attempts to enter the enterprise network, it passes through the firewall system where p is examined against the rules of the underlying firewall F. Packet p does not match rule 1 in F but matches rule 2. Because rule 2 has a decision 'accept', packet p enters the enterprise network. Note that the arrow between the firewall system and enterprise network and the label (accept p or reject p) in Figure 2.1 are the symbolic representation of the following logic: if the firewall system concludes that the decision for p is 'reject', then p is discarded at the firewall system and so cannot enter the enterprise network. Otherwise the firewall system concludes that the decision for p is 'accept' and in this case the system forwards p to the enterprise network.

2.3 Literature Review of Firewalls

We divide the literature on firewalls into five categories: firewall design, firewall analysis, property verification of firewalls, packet classification and firewall outsourcing. A brief survey of the research that have been conducted in the first four categories is in order. We review the research works that fall in the category of firewall outsourcing in the next section.

Firewall Design

A firewall should be carefully designed so that designed firewall adhere to the design specification. Some prominent methods that can be used in designing firewalls are reported in [28], [29], [44], [4], [56], and [54].

The method for designing firewalls in [28, 29] consists of two steps. First the designer designs the desired firewall using a large conflict-free decision diagram. Second the designer uses several algorithms to convert the large decision diagram into a compact, yet functionally equivalent, sequence of rules. This design method can be referred to as "simplifying firewalls by introducing conflicts".

The method for designing firewalls in [44] consists of three steps. First, the same specification of the desired firewall is given to multiple teams who independently design different versions of the firewall. Second, the resulting multiple versions of the firewall are compared with one another. Third, all discrepancies between the multiple firewall versions are resolved, and a final firewall that is agreed upon by all teams is generated. This design method can be referred to as "diverse firewall design".

The method for designing firewalls in [4] consists of three steps. First, the set of all expected packets is partitioned into non-overlapping subsets S_1, S_2, $\cdots$, S_k. Second, for each subset S_i (obtained in the first step), design a firewall F_i that accepts some of the packets in the subset S_i. Third, identify firewalls F_1, F_2, $\cdots$, F_k generated in the second step as the desired firewall.

This design methods can be referred to as "divide-and-conquer".

The method for designing firewalls in [56] consists of k steps. First, the designer starts with a simple firewall F_1 that accepts more packets than the designer wishes. Second, the designer designs a second firewall F_2 such that if any packet is accepted by F_2 then the same packet is also accepted by F_1. This process is repeated k times until the designer reaches a firewall F_k that accepts those packets and only those packets that the designer wishes to be accepted. This design method can be referred to as "step-wise refinement".

In [54], a bottom-up design method has been presented that can be followed by a designer in designing firewalls. This design method proceeds as follows. First, the designer designs several simple firewalls. Second, the designer combines these simple firewalls using the three firewall operators "not", "and", and "or" into a single firewall expression.

Firewall Analysis

Rule Anomaly Detection:

Firewall rules can be overlapping or disjoint. When rules are disjoint, the ordering of the rules is insignificant. Two rules conflict when they are overlapping and have conflicting decisions. It is possible that a packet matches both of the two conflicting rules. In this case, firewall rules are assigned priority and are ordered from higher priority to lower priority. The conflict is resolved by choosing the first match rule. Thus finding the correct ordering of the rules is very important and can be challenging when a firewall has large number of

rules. Moreover, when the firewall contains a large number of rules, the possibility of writing conflicting or redundant rules is relatively high. Therefore, it is of utmost importance to detect the conflicting rules in a firewall, as well as other anomalies, such as existence of redundant rules, shadowed rules etc.

A classification of anomalies in a firewall, as well as algorithms to detect them, is presented in a series of work [6–8] by Al-Shaer et. al. While in [8] the authors defined intra-firewall anomalies, in [6, 7] the authors defined both intra-firewall and inter-firewall anomalies. Besides classifying anomlaies, the authors also proposed algorithms to detect and resolve these anomalies. These works resulted in a tool called Firewall Policy Advisor. This tool can automatically discover firewall rule anomalies after any rule insertion, removal, and modification takes place, and can generate anomaly-free firewall. Like Firewall Police Advisor, several other tools to detect and resolve firewall anomalies have been proposed. FIREMAN [74] and FAME [35] are among these tools. Firewall Policy Advisor only has the capability of detecting pairwise anomalies in firewall rules. FIREMAN can detect anomalies among multiple rules by analyzing the relationships between one rule and all preceding rules. FAME considers all preceding and all subsequent rules when performing anomaly analysis.

While the above mentioned works resolve anomalies preserving the priority order of the rules, the authors in [31] claims that resolving rule conflicts based on prioritizing conflicting rules, and choosing the higher priority rule does not always work. For example, they considered the case when each at-

tribute in a rule is defined as a bit string. They proposed a scheme for conflict resolution by modifying existing rules, inserting resolve rules and choosing the best match rule. An linear space conflict detection technique has been presented in [19].

The problem of detection and removal of firewall rule redundancy has also been addressed in [3] and [45].

Vulnerability Analysis:

A firewall vulnerability is defined as an error made during firewall design, implementation, or configuration, that can be exploited to attack the trusted network that the firewall is supposed to protect [36]. Several methods for the logical analysis of firewalls have been reported in [34, 36, 48, 50, 70], and [14]. A framework for understanding the vulnerabilities in a single firewall is outlined in [22], and an analysis of these vulnerabilities is presented in [36]. A quantitative study of configuration errors for a firewall is presented in [70]. An example of an efficient firewall analysis algorithm is given in FIREMAN [74]. An integrated analysis engine for firewalls in a network is given in Fang [48] where the authors developed a firewall analysis tool to perform customized queries on a set of filtering rules and to manually verify the correctness of the firewall policy. A firewall test generation tool, called Blowtorch has been presented in [34].

Property Verification of Firewalls

Over the last couple of years, researchers have shown interest in determining several logical properties of a given firewall. Examples of some logical properties include adequacy, implication, equivalence etc. Adequacy property refers to problem of determining whether a given firewall accepts at least one packet. Implication property refers to the problem of determining whether a given firewall P accepts all packets that are accepted by another given firewall Q. These properties have been formally defined in [18]. Also, it has been shown in [18] that the problems of determining whether given firewalls defined over any number of attributes satisfy some desired properties of adequacy, implication, and equivalence are all NP-hard.

In [2], the authors present a polynomial time approach, called the PSP method, to verify whether a given firewall satisfies a given logical property (defined as a logical predicate) under the assumption that the number of attributes in the firewall is fixed. PSP method has been later used to design a polynomial time algorithm in [56] to solve the implication problem under the assumption that the number of attributes in a rule in firewall is fixed. An incremental verification approach has been presented in [17].

Besides the assumption of fixed number of attributes, there are two main approaches to face the NP-hardness of determining whether given firewall satisfy some desired properties of adequacy, implication, and equivalence. The first approach is to use SAT solvers, for example as discussed in [33], [77], and [5], to determine whether a given firewall satisfies some desired proper-

ties of adequacy, implication, and equivalence. Note that the time complexity of using SAT solvers is polynomial in most practical situations. The second approach is to use probabilistic algorithms [1]. Note that the time complexities of probabilistic algorithms are always polynomial but unfortunately these algorithms can yield wrong determinations in rare cases.

Moreover, the authors in [40, 41] investigated a novel representation of firewalls as finite automata rather than as sequences of rules. They showed later in [38], how to use the automata representation of a given firewall to determine whether the given firewall satisfies some desired properties of adequacy, implication, and equivalence.

Packet Classification

Given a packet p and a firewall F, a packet classification algorithm for firewalls determines whether p is accepted or rejected according to the rules in F.

When the firewall is represented as a sequence of rule, the simplest algorithm is linear search of the firewall rules to determine the first match rule for p. Linear search exhibits packet classification complexity of $\mathcal{O}(t \times n)$ where n is the number of rules in a firewall and t is the number of attributes.

Note that a firewall can also be represented as a decision tree [7, 44, 74] or as a finite automata [41] instead of as a sequence of rules. So packet classification is not only limited to the linear search approach. Several other packet classification approaches are decision tree methods (for example, Hyper-

Cuts [63]), partitioning methods (for example, Tuple Space Search (TSS) [64]), hybrid methods that use both decision trees and partitioning (for example, Smartsplit [32], PartitionSort [72]), and TCAM-based methods [47].

Decision tree based classification algorithms [63] exhibit logarithmic complexity in packet classification. However, updating a rule sometimes require reconstruction of the decision tree.

Partitioning methods, for example TSS [64], partitions the original ruleset into smaller rulesets based on rule characteristics such that each partition can be searched and updated in $\mathcal{O}(t)$ time where t is the number of attributes. Although updating a rule is faster in TSS than that in decision tree approaches, but classification time increases when the number of partitions increases because each partition must be searched for each packet.

Hybrid approaches [32, 72] use both the partition approach to partition the ruleset and decision tree approach for searching each resulting ruleset. As a result, hybrid approaches improve rule update time over decision tree methods as decision trees are constructed for smaller rulesets. Hybrid approaches improve the classification time over partition methods by producing a smaller number of partitions. A recent hybrid approach, Partition Sort [72] achieves both logarithmic classification and logarithmic rule update time.

Ternary content addressable memories (TCAMs [47]) are used to perform high speed packet classification. A TCAM is a memory chip where each entry can store a packet classification rule that is encoded in ternary format.

Given a packet, the TCAM hardware can compare the packet with all stored rules in parallel and then return the decision of the first rule that the packet matches. Thus, it takes $\mathcal{O}(1)$ time to find the decision for any given packet. Given a firewall, the problem of generating another semantically equivalent firewall that requires fewer number of TCAM entries has been addressed in [46] and [16].

2.4 Literature Review of Firewall Outsourcing

Firewall outsourcing started to gain attention because of the related economic benefits since the first decade of the twenty-first century. Some cloud service providers (CSP) or internet service providers (ISP), for example AT&T, started to offer outsourced firewall systems as a service to enterprise networks [15, 30, 58, 59, 69]. In such a service model, the firewall system of an enterprise network is implemented and managed by the service provider. An enterprise requires to provide its firewall rules to its CSP/ISP to configure the firewall system. However, in this case enterprises do not have much control over the design and execution of the outsourced system.

Since more and more enterprises were becoming interested in using outsouced firewall systems, both academic and industry researchers became interested in proposing models to design customized outsourced firewall systems for an enterprise network. As a result, starting from the beginning the current decade, several models for outsourced firewall systems have been proposed which enterprises can follow to design their own firewall systems using

public clouds.

Some recent efforts in this area are surveyed in [67].

In 2012, Sherry et al. [61] studied the benefits outsourcing of firewall systems by conducting a survey over 57 enterprise networks to estimate the associated cost and complexity to implement and manage their firewall systems. They also proposed an architecture, called APLOMB, for outsourced firewall systems for software defined network (SDN [21]). This system is designed to be executed in a public cloud. They have argued that the enterprises can reduce the associated cost and management complexity to implement and manage their firewall systems by adopting APLOMB architecture for firewall outsourcing.

In the same year, Gibb et al. [25] also proposed an outsourced firewall system for software defined network (SDN). This system is designed to be executed in any location, for example, inside any local network or in a public cloud, without requiring any changes in the design of the system. The enterprise network only requires to forward the packets to the location where the firewall system is being executed. The location where the firewall system is being executed can be geographically distant from the enterprise network.

The above mentioned outsourced systems provide the underlying firewall in the clear to the public clouds and also do not verify that the task executed by public clouds is indeed correct. Thus, these systems neither defend against privacy attacks or verifiability attacks described in Section 1.2 in

Towards the end of 2012, Khakpour and Liu [37] proposed an outsourced firewall system considering the fact that public clouds should not be given the underlying firewall in the clear while outsourcing. Because if the cloud knows the underlying firewall, it may leak that firewall to potential attackers of the system. So they designed an outsourced firewall system where the underlying firewall is encrypted before it is outsourced to the cloud. They encrypt the underlying firewall in two steps. First, they use a Firewall Decision Diagram (FDD) [29] to represent the rules of the firewall. Second, they use Bloom Filters [13] to represent edges of the FDD. Because the rules of the underlying firewall are encrypted before they are outsourced to the cloud, the cloud cannot know the rules of the underlying firewall and so cannot leak these rules to potential attackers of the system. Their work was the first attempt to design outsourced system when the underlying firewall is encrypted before it is outsourced.

Following their work, several outsourced systems [42, 43, 49, 60, 62], and [68] have been proposed afterwards that encrypt the underlying firewall while outsourcing to public clouds. Among these systems, the outsourced system in [43] encrypts the packets that are sent to the public cloud as well as the underlying firewall. Embark enables the cloud to check the encrypted packets against the encrypted firewall. The system in [49] used partial homomorphic encryption. This system also requires the packets to be encrypted before processing by the cloud. However, encryption of packets are done by a trusted component of the

public cloud. Other systems only encrypts the underlying firewall.

These systems differ from each other mainly in their architecture and in the encryption mechanisms. For example, each of the outsourced systems in [37, 43, 62] is executed by one public cloud, each of the outsourced systems in [60, 68] is executed by two cooperative public clouds, and the outsourced system in [42] is executed partially by a public cloud and partially by a private cloud. Each of these systems aims to protect the underlying firewall from being leaked to public clouds. Therefore, all of these systems defend against privacy attacks. However, none of these systems verifies that the task executed by a public cloud is indeed correct. Thus, these systems do not defend against verifiability attacks.

A very few outsourced systems, for example, the systems in [20, 75], and the system in [76], have been proposed in the literature that consider the fact that the tasks that are executed by the public clouds need to be verified at the enterprise end. Whereas the firewall systems in [75] and [76] execute the verification steps online, the firewall system in [20] executes the verification steps offline. However, the rules of the underlying firewall are stored in the clear in the cloud in each of these systems. Therefore, these systems defend against verifiability attacks but do not defend against privacy attacks.

The above mentioned efforts are mainly focused on outsourced firewall systems. Besides there efforts, there exists a handful of research works that deal with verifiability concerns in public cloud computing in general. Different verifiable computation schemes have been proposed over time such as

trusted platform module [66], interactive proofs [26], probabilistic checkable proofs [10], non-interactive verifiable computation [23] and so on. A brief survey of some these schemes are presented in [73]. In 2010, Gennaro et al. [23] defined a non-interactive verifiable computation scheme for any function using Yao's garbled circuit [71] combined with a fully homomorphic encryption system [24]. This scheme is called the verifiable fully homomorphic encryption (VFHE) which accounts for both privacy of outsourced computation and correctness of computed results. However, the problem of how to adopt VFHE in an outsourced firewall system is still open. Melis et al. [49] used homomorphic encryption to encrypt underlying firewall and incoming packets to design an outsourced firewall system. But their system does not verify the correctness of the computation executed by the public cloud.

Recent efforts have been made to discuss how to outsource systems of access control policies, such as XACML polices [9], into public clouds [11, 12]. These systems do not defend against verifiability or privacy attacks.

Chapter 3

Outsourcing of Firewalls

3.1 Introduction

The material presented in this chapter is based on our paper [57][1]. In this chapter, we present a special class of firewall systems called *outsourced systems*. Like a regular firewall system, an outsourced system consists of two units: a rule matching unit and a decision unit. To simplify the architecture of the outsourced system, the rule matching unit of this system is executed by a public cloud C.

The architecture of the outsourced system is shown in Figure 3.1. Note that the only difference between the regular firewall system in Figure 1.1 and the outsourced system in Figure 3.1 is that the rule matching unit in the former system is executed by the firewall system itself whereas the rule matching unit in the latter system is executed by a public cloud C.

Using public cloud C to execute the rule matching unit of an outsourced system has a number of benefits and some disadvantages. The benefits of using

[1]Rezwana Reaz, Ehab S. Elmallah, and Mohamed G. Gouda. Executing firewalls in public clouds. In Proceedings of the 10th international conference computing, communication and networking technologies (ICCCNT). IEEE, 2019. (Accepted for publication). Rezwana Reaz is the only student author in this paper and contributed the most in this paper.

cloud C to execute the rule unit are as follows [37, 58, 61, 69]. First, it can reduce the initial investment and the operational cost of the firewall system by taking advantage of the pay-per-use model of the cloud. Second, it can reduce the number of staff needed to manage and implement the firewall system. Third, it can increase the availability of the firewall system by maintaining necessary back-ups.

The disadvantage of using public cloud C in executing the rule matching unit is that cloud C is unreliable and so the outsourced system is vulnerable to two types of attacks: verifiability attacks and privacy attacks. We describe these two types of attacks in Section 3.3 below.

Prior work in this area [20, 37, 43, 60, 61, 75] yielded outsourced systems that can defend against one of these two types of attacks, but none of the systems can defend against both types of attacks. Our goal in this chapter is to design outsourced systems that can prevent these two types of attacks from occurring.

Table 3.1 classifies the prior outsourced systems into three categories: (1) systems that do not defend against any attacks, (2) systems that can defend only against verifiability attacks, and (3) systems that can defend only against privacy attacks.

Table 3.1: Summary of Prior Systems

Category	Systems
Do not defend against any attacks	[25] and [61]
Can defend only against verifiability attacks	[20], [75], and [76]
Can defend only against privacy attacks	[37], [42], [43], [60], [62], and [68]
Can prevent both attacks	This chapter

3.2 Execution of Outsourced Systems

The architecture of an outsourced system whose underlying firewall is F is shown in Figure 3.1. This outsourced system consists of two units: the rule matching unit which is executed by a public cloud C, and the decision unit which is executed by the firewall system. Both the rule matching unit and the decision unit are built on top of the same underlying firewall F.

When a packet p attempts to pass this outsourced system, p is first directed to cloud C which hosts the rule matching unit. Cloud C uses the underlying firewall F to compute a sequence number v where v is the sequence number $\#(F, p)$. Then, C forwards the pair (p, v) to the decision unit which is executed by the firewall system.

The decision unit uses firewall F to compute the decision ("accept" or "reject") of the rule whose sequence number is $\#(F, p)$.

If the rule whose sequence number is $\#(F, p)$ has a decision "accept", then the decision unit forwards p to the enterprise network. Otherwise, the rule

whose sequence number is $\#(F, p)$ has a decision "reject" and so the decision discards packet p and does not forward it to the enterprise network.

3.3 Unreliable Public Clouds

A cloud C in the outsourced system is *reliable* iff C satisfies the following two conditions. First, when C sends a pair (p, v) to the decision unit, value v is indeed the sequence number of the first match rule (rather than any other match rule) in the underlying firewall F for packet p. Second, if C knows the rules in F, C does not leak F to any potential attacker of the system.

The outsourced system in Figure 3.1 is correct only if the public cloud C is reliable.

But cloud C is in fact unreliable. Hence, the outsourced system in Figure 3.1 is vulnerable to two types of attacks: verifiability attacks and privacy attacks. We describe these two types of attacks next.

The verifiability attacks caused by cloud C can be described as follows. When cloud C executes the steps to compute the sequence number $\#(F, p)$ of the first match rule in the underlying firewall F for the incoming packet p, C may compute a wrong value. In particular, the computed value v can be the sequence number for a match but not for the first match rule in F for p.

The privacy attacks caused by cloud C can be described as follows. If cloud C knows the rules of the underlying firewall F, C can leak the underlying firewall F to any potential attacker of the firewall system.

As summarized in Table 3.1, all prior work on designing outsourced firewall systems either defend against verifiability attacks or defend against privacy attacks, but do not defend against both types of attacks. For example, the outsourced systems in [20] and [76] defend against verifiability attacks, but do not defend against privacy attacks. Also the outsourced systems in [37] and [43] defend against privacy attacks but not against verifiability attacks.

We now discuss how to design an outsourced system that can prevent both verifiability and privacy attacks from occurring. In the following two sections, we present two designs of outsourced systems. The first system is called the verifiable firewall system. This system can prevent verifiability attacks but cannot prevent privacy attacks. The second system is called the private firewall system. This system can prevent both verifiability and privacy attacks from occuring.

3.4 Verifiable Firewall Systems

The verifiable system in this section is obtained from the outsourced system in Section 3.2 by performing the following three modifications. First, cloud C in the outsourced system is replaced by two identical public clouds C_1 and C_2. Second, the rule matching unit that is hosted in cloud C is replaced by two identical rule matching units that are hosted in clouds C_1 and C_2 as shown in Figure 3.2. Third, the decision unit in the outsourced system is replaced by a verifiable decision unit as shown in Figure 3.2.

Next, we describe the tasks that need to be performed by the verifiable

decision unit.

When a packet p attempts to pass the verifiable system in Figure 3.2, packet p is directed to the rule matching unit hosted in cloud C_1 so that C_1 can compute a sequence number v_1 and send the pair (p, v_1) to the verifiable decision unit. Also, packet p is directed to the rule matching unit hosted in cloud C_2 so that C_2 can compute a sequence number v_2 and send the pair (p, v_2) to the verifiable decision unit.

Cloud C_1 is supposed to compute v_1 as equal to the sequence number $\#(F, p)$ of the first match rule in F for p. But because C_1 is a public cloud, and so is unreliable, the computed value v_1 can end up being the sequence number of any match rule (not necessarily the first match rule) in F for p. Similarly, cloud C_2 is supposed to compute v_2 as equal to the sequence number $\#(F, p)$ of the first match rule in F for p. But because C_2 is a public cloud, and so is unreliable, the computed value v_2 can end up being the sequence number of any match rule (not necessarily the first match rule) in F for p.

If value v_i computed by cloud C_i and sent to the verifiable decision unit equals the sequence number $\#(F, p)$, then cloud C_i is said to "have told the truth" to the verifiable decision unit. On the other hand, if value v_i computed by cloud C_i and sent to the verifiable decision unit is not equal to the sequence number $\#(F, p)$, then cloud C_i is said to "have lied" to the verifiable decision unit.

A cloud C_i is said to be *sensible* iff C_i does not lie when the other cloud

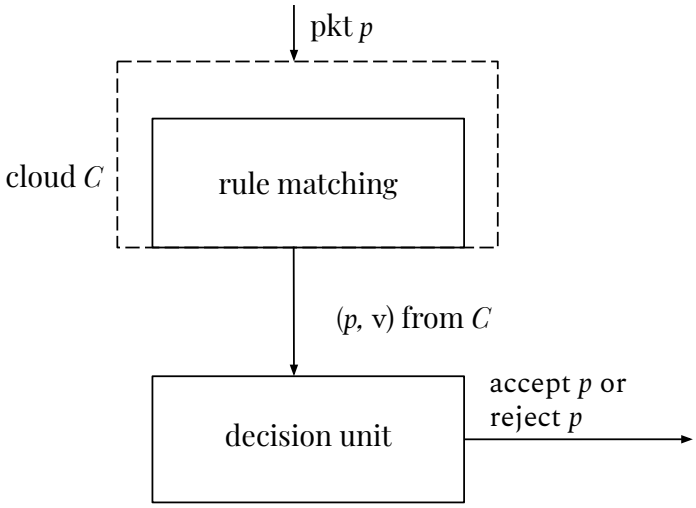

Figure 3.1: A firewall system with outsourcing

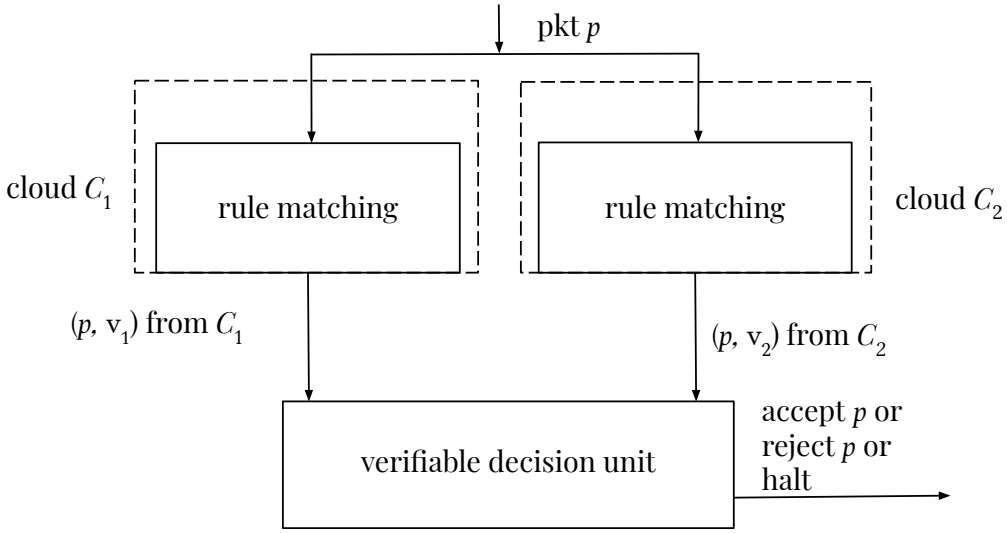

Figure 3.2: Verifiable firewall system

can tell the truth and enable the decision unit to detect that C_i has lied.

Two public clouds C_1 and C_2 are said to be *non-colluding* iff when cloud C_1 sends a pair (p, v_1) and cloud C_2 sends a pair (p, v_2) to the decision unit, then v_1 and v_2 can be different.

Note that if both clouds are sensible and non-colluding then neither cloud will lie. This is because if one cloud, say C_1, lies then there is a possibility that the other cloud C_2 does not lie and sends the sequence number $\#(F, p)$ and hence enables the decision unit to detect that C_1 has lied. In contrast, if collusion occurs and the two clouds agree on sending the same sequence number of a match rule then the two sensible clouds can lie simultaneously.

Theorem 3.4.1. *Under the assumption that the two clouds C_1 and C_2 are both sensible and non-colluding, the two pairs (p, v_1) and (p, v_2), computed respectively by clouds C_1 and C_2, are such that both v_1 and v_2 equal the sequence number $\#(F, p)$. This indicates that verifiability attack cannot occur in the verifiable system.*

Proof. There are three cases to consider.

1. Case 1: Cloud C_1 has lied and cloud C_2 may or may not have lied. In this case, v_1 is not equal to $\#(F, p)$. If v_2 equals $\#(F, p)$, then v_1 is strictly greater than v_2 and the decision unit can detect that cloud C_1 has lied. By the assumption that C_1 and C_2 are non-colluding, v_2 can be $\#(F, p)$. In other words, C_2 may have told the truth and have enabled

the decision unit to detect that C_1 has lied. By the assumption that C_1 is sensible, C_1 cannot lie in this case and so Case 1 is not possible.

2. Case 2: Cloud C_2 has lied and cloud C_1 may or may not have lied. In this case, v_2 is not equal to $\#(F, p)$. If v_1 equals $\#(F, p)$, then v_2 is strictly greater than v_1 and the decision unit can detect that cloud C_2 has lied. By the assumption that C_1 and C_2 are non-colluding, v_1 can be $\#(F, p)$. In other words, C_1 may have told the truth and have enabled the decision unit to detect that C_2 has lied. By the assumption that C_2 is sensible, C_2 cannot lie in this case and so Case 2 is not possible.

3. Case 3: Neither C_1 nor C_2 has lied. If cloud C_1 lies, then this case is Case 1 and Case 1 is not possible by the assumption that cloud C_1 is sensible. Similarly, If cloud C_2 lies, then this case is Case 2 and Case 2 is not possible by the assumption that cloud C_2 is sensible. Therefore, neither C_1 nor C_2 has lied which makes Case 3 possible.

Since Case 3 is the only possible case, each cloud send the sequence number $\#(F, p)$ to the decision unit. This indicates that verifiability attack cannot occur. □

From Theorem 3.4.1, each of the two sequence numbers v_1 and v_2 sent to the verifiable decision unit respectively by clouds C_1 and C_2, is equal to the sequence number $\#(F, p)$. Thus, the verifiable system prevents verifiability attacks from occurring.

Although the two sequence number v_1 and v_2 are expected to be equal when the verifiable decision receives them, v_1 and v_2 can be different if any of these two sequence numbers gets corrupted before it reaches the verifiable decision unit. If the decision unit detects that v_1 and v_2 are not equal, then the decision unit concludes that corruption of v_1 or v_2 has occurred. In this case, the decision unit discards packet p and puts the verifiable system into a halt so that no more incoming packets can be allowed to enter the verifiable system.

Therefore, after the verifiable decision unit receives the two pairs (p, v_1) and (p, v_2), the decision unit is required to compare the two sequence numbers v_1 and v_2 to check whether they are equal or they are not equal indicating that a corruption has occurred. If the two values are equal, then the decision unit uses the underlying firewall F to compute the decision of the rule whose sequence number is v_1. If the decision of this rule is "accept", then the verifiable decision unit forwards packet p to the enterprise network. Otherwise, the decision of this rule is "reject" and so the decision unit discards packet p.

3.5 Private Firewall Systems

The verifiable system that is discussed in the previous section prevents verifiability attacks but is still vulnerable to privacy attacks caused by the fact that cloud C_i which hosts the rule matching unit of the verifiable system, knows the the underlying firewall F. Because C_i is unreliable, C_i can leak the

underlying firewall F to any potential attacker of the system.

To prevent privacy attacks from occurring, we design the private firewall system from the verifiable system presented in the previous section as follows. We replace each rule matching unit that uses firewall F by a rule matching unit that uses the incomplete version IF of F.

The incomplete version IF of F is the same as F except that the decisions of all the rules in IF are unspecified. For example, if the underlying firewall F is as follows:

$$\begin{array}{lll} 1 & \big((u_1 \in [1,4]) \wedge (u_2 \in [8,9])\big) & \rightarrow \quad \text{reject} \\ 2 & \big((u_1 \in [2,4]) \wedge (u_2 \in [7,9])\big) & \rightarrow \quad \text{accept} \\ 3 & \big((u_1 \in [1,9]) \wedge (u_2 \in [1,9])\big) & \rightarrow \quad \text{reject} \end{array}$$

then the incomplete version IF of F is as follows:

$$\begin{array}{lll} 1 & \big((u_1 \in [1,4]) \wedge (u_2 \in [8,9])\big) & \rightarrow \quad \text{unspecified} \\ 2 & \big((u_1 \in [2,4]) \wedge (u_2 \in [7,9])\big) & \rightarrow \quad \text{unspecified} \\ 3 & \big((u_1 \in [1,9]) \wedge (u_2 \in [1,9])\big) & \rightarrow \quad \text{unspecified} \end{array}$$

The first rule in this example of IF is called incomplete rule 1, the second rule in IF is called incomplete rule 2, and so on. Now consider two packets p and q where p is defined as the tuple ($u_1 = 3$, $u_2 = 7$) and q is defined

as the tuple ($u_1 = 2$, $u_2 = 6$). Packet p does not match incomplete rule 1, but matches incomplete rule 2. So the first match incomplete rule in IF for p is the incomplete rule 2. Similarly, packet q does not match incomplete rule 1 and incomplete rule 2. Rather it matches incomplete rule 3. So the first match incomplete rule in IF for q is the incomplete rule 3.

We adopt the notation $\#(IF, p)$ to denote the sequence number of the first match incomplete rule in IF for packet p. For example, the sequence number $\#(IF, p)$ is 2 and the sequence number $\#(IF, q)$ is 3. Observe that the sequence number $\#(F, p)$ is equal to the sequence number $\#(IF, p)$. Therefore, the notations $\#(IF, p)$ and $\#(F, p)$ can be used interchangeably.

A packet p that attempts to pass the private system whose underlying firewall is F, is first directed to each of the rule matching units hosted in clouds C_1 and C_2. Each rule matching unit in the private system uses the incomplete firewall IF instead of the complete firewall F and computes the sequence number $\#(IF, p)$ which equals the sequence number $\#(F, p)$. Each cloud C_i then sends its computed value $\#(F, p)$ to the verifiable decision unit along with packet p. The verifiable decision unit of the private system computes the decision for p in the same way the verifiable decision unit does in the verifiable system.

Note that in the private system, for each incoming packet p, each cloud knows the sequence number $\#(F, p)$ but does not know the decision of the rule whose sequence number is $\#(F, p)$. Therefore, neither cloud knows the rules of F and so cannot leak these rules to potential attackers.

Based on these discussions, correctness of the private system is obtained from Theorem 3.5.1.

Theorem 3.5.1. *Each cloud C_i in the verifiable system knows the underlying firewall F and can leak F to potential attackers of the system. By contrast, no cloud C_i in the private system knows the underlying firewall F and so cannot leak F to potential attackers of the system. This indicates that no privacy attack can occur in the private system.*

3.6 Chapter Summary

Our contributions in this chapter are two folds. First, we present a family of firewall systems which is shown in Figure 3.3. Each member in this family consists of a rule matching unit and a decision unit. The firewall system without outsourcing executes the tasks of the rule matching unit and the decision unit without any help from a public cloud. In contrast, the outsourced system outsources the rule matching unit to a public cloud.

Unfortunately, public clouds are unreliable which makes the outsourced system vulnerable to two types of attacks: verifiability attacks and privacy attacks. To prevent these attacks from occurring, we present designs of two outsourced systems: the verifiable system and the private system. The verifiable system outsources the task of the rule matching unit to two public clouds and can prevent verifiability attacks under the assumption that both clouds are sensible and non-colluding. However, the verifiable system does not prevent privacy attacks.

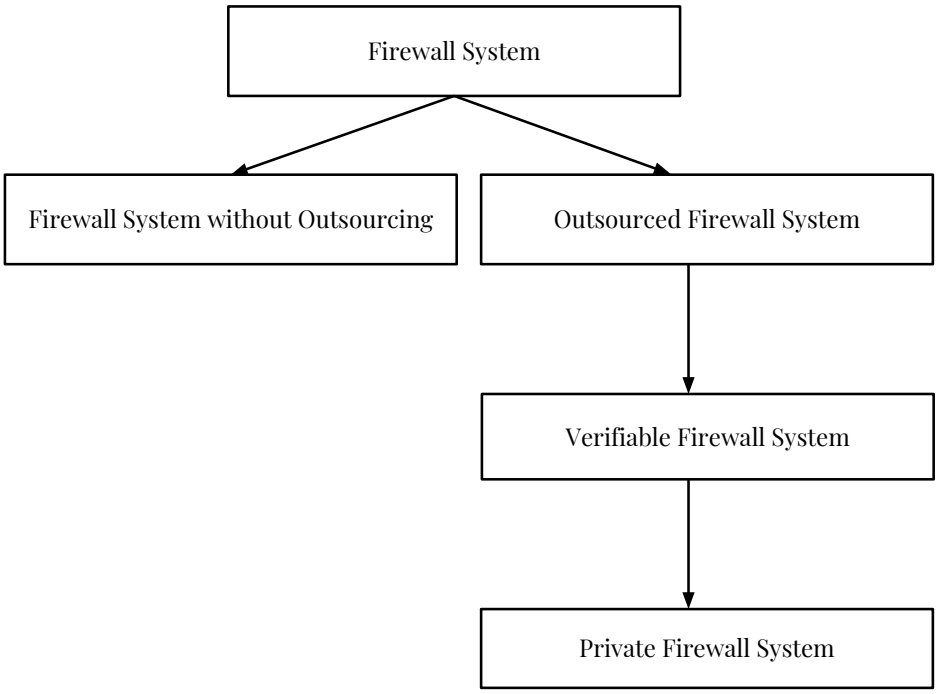

Figure 3.3: Our family of firewall systems

Our second contribution in this chapter is a presentation of the private system which can prevent both verifiability and privacy attacks from occurring. Prior work on designing outsourced systems using public clouds either defend against verifiability attacks, for example [20] and [75], or defend against privacy attacks, for example [37] and [60], but do not defend against both attacks.

The private system presented in this chapter uses two public clouds and can prevent both verifiability and privacy attacks under the assumption that the two public clouds are sensible and non-colluding. An extension of the work presented in this chapter is to design an outsourced system that can prevent both verifiability and privacy attacks under the assumption that the two public clouds can be colluding.

Chapter 4

Outsourcing of Partially Specified Firewalls

So far we defined the decision of a rule in a firewall to be either "accept" or "reject". In this chapter, we consider firewalls where the decision of each rule is "accept", "reject", or "unspecified". We refer to this class of firewalls as partially specified firewalls and discuss techniques for outsourcing partially specified firewalls.

An example of a partially specified firewall PF is as follows.

$$
\begin{array}{lll}
1 & \big((u_1 \in [1,4]) \wedge (u_2 \in [8,9])\big) & \rightarrow \text{ reject} \\
2 & \big((u_1 \in [2,6]) \wedge (u_2 \in [7,8])\big) & \rightarrow \text{ unspecified} \\
3 & \big((u_1 \in [5,5]) \wedge (u_2 \in [6,8])\big) & \rightarrow \text{ unspecified} \\
4 & \big((u_1 \in [2,4]) \wedge (u_2 \in [7,9])\big) & \rightarrow \text{ accept} \\
5 & \big((u_1 \in [1,9]) \wedge (u_2 \in [1,9])\big) & \rightarrow \text{ reject}
\end{array}
$$

Note that this partially specified firewall has two attributes u_1 and u_2 and the domain of each attribute is the integer interval [1, 9]. The decision of each rule is either "accept","reject", or "unspecified".

A packet p is said to be *accepted* or *rejected* respectively by a partially specified firewall PF iff PF has an accept rule or reject rule r that matches

packet p and all the rules that match p and precede r in PF are unspecified rules. For example, packet p ($u_1 = 3$, $u_2 = 7$) is accepted by the partially unspecified firewall PF mentioned above because PF has an accept rule, rule 4 that matches packet p and all the rules that match p and precede rule 4 in PF are unspecified rules.

A partially specified firewall PF is said to be *complete* iff each packet p is either "accepted" or "rejected" by PF. For example, the above partially specified firewall is complete.

From now on, whenever we mention a partially specified firewall we mean a complete partially specified firewall.

A partially specified firewall PF is equivalent to a firewall F iff every packet p that is accepted or rejected by PF respectively is also accepted or rejected by F respectively, and vice versa.

Theorem 4.0.1. *For every partially specified firewall PF there exists a firewall F such that PF and F are equivalent.*

Proof. Let PF be a partially specified firewall and let F be the firewall that is obtained from PF by removing all unspecified rules in PF. From definition of a packet p being accepted or rejected respectively by a partially specified firewall PF, packet p is accepted or rejected respectively by PF iff p is accepted or rejected respectively by F. Therefore, firewall F is equivalent to the partially specified firewall PF. □

We now discuss how to modify a firewall system when the underlying firewall is a partially specified firewall PF. The architecture of a firewall system has been presented in Figure 1.1 which consists of a rule matching unit and a decision unit. When the underlying firewall is a partially specified firewall, then the task of rule matching unit has to be redefined as follows.

For any packet p, the rule matching unit requires to compute the sequence number of the 'first match rule that has a decision either accept or reject' in PF for p. Note that 'first match rule that has a decision either accept or reject' in PF for p is not necessarily the sequence number of the 'first match rule' in PF for p. After computing the sequence number of the 'first match rule that has a decision either accept or reject' the rule matching unit sends this sequence number to the decision unit and the decision unit applies the decision of this rule to p.

Above design of the firewall system whose underlying firewall is a partially specified firewall PF suggests that the rule matching unit needs to know the decisions of the rules of PF. This makes the design of an outsourced system for a partially specified firewall PF challenging because the outsourced systems presented in this dissertation require that the rule matching units are executed in clouds and clouds do not know the decision of the rules of the underlying firewall.

An alternative way to design an outsourced system for a partially specified firewall PF is to find an equivalent firewall F and design the outsourced system for F. From Theorem 4.0.1 for any PF there exists a firewall F such

that PF and F are equivalent.

We now design an outsourced system for a partially specified firewall PF in two steps.

- In the first step, we obtain a firewall F from the partially specified firewall PF by removing all unspecified rules such that PF and F are equivalent.
- In the second step, we design a private system presented in Chapter 3 for underlying firewall F obtained in the first step. The designed private system for firewall F is the desired outsourced system for partially specified firewall PF.

Chapter 5

Firewall Expressions

5.1 Introduction

The material presented in this chapter is based on our papers [54, 55][1]. In this chapter, we present a generalization of firewalls called firewall expressions. A firewall expression is specified using one or more firewalls and the three firewall operators: "not", "and", and "or". We show that firewall expressions can be utilized to support bottom-up methods for designing firewalls. We also show that each firewall expression can be represented by a set of special types of firewalls, called slices. Moreover, we present several algorithms that use the slice representation of given firewall expressions to verify whether the given firewall expressions satisfy logical properties such as adequacy, implication, and equivalence.

We now present examples of two firewalls G and H and use these examples to introduce the concept of "firewall expressions".

[1]Rezwana Reaz, H. B. Acharya, Ehab S. Elmallah, Jorge A. Cobb, and Mohamed G Gouda. Policy expressions and the bottom-up design of computing policies. Computing, 101(9):13071326, 2019. Rezwana Reaz is the only student author in this paper and the main contributor in this paper.

Let firewall G consists of three rules which are defined as follows:

$$\begin{aligned}
\big((u_1 \in [1,4]) \wedge (u_2 \in [8,9])\big) &\rightarrow \text{reject} \\
\big((u_1 \in [2,4]) \wedge (u_2 \in [7,9])\big) &\rightarrow \text{accept} \\
\big((u_1 \in [1,9]) \wedge (u_2 \in [1,9])\big) &\rightarrow \text{reject}
\end{aligned}$$

The predicate of each rule in this firewall G is defined using two attributes u_1 and u_2 whose integer values are taken from the integer interval [1, 9]. The first rule states that each packet (b_1, b_2), where the value of b_1 is an integer in the interval $[1, 4]$ and where the value of b_2 is an integer in the interval $[8, 9]$, is to be rejected. The second rule states that each packet (b_1, b_2), that does not match the first rule and where the value of b_1 is an integer in the interval $[2, 4]$ and where the value of b_2 is an integer in the interval $[7, 9]$, is to be accepted. The third rule states that each packet (b_1, b_2) that does not match the first two rules is to be rejected. Thus, the set of packets that are accepted by firewall G is $\{(2, 7), (3, 7), (4, 7)\}$. Notice that because the third rule rejects all packets that do not match the first two rules.

A second firewall H that consists of three rules, where each rule is defined over attributes u_1 and u_2, is as follows:

$$\begin{aligned}
\big((u_1 \in [2,3]) \wedge (u_2 \in [7,7])\big) &\rightarrow \text{accept} \\
\big((u_1 \in [2,4]) \wedge (u_2 \in [7,8])\big) &\rightarrow \text{accept} \\
\big((u_1 \in [1,9]) \wedge (u_2 \in [1,9])\big) &\rightarrow \text{reject}
\end{aligned}$$

r The set of packets that are accepted by H is $\{(2,7),(3,7),(4,7),(2,8),(3,8),(4,8)\}$ and all other packets are rejected.

Now assume that we need to use the two given firewalls G and H to design a firewall expression (G or H). This firewall expression accepts every packet that is accepted by firewall G or accepted by firewall H. Thus, the set of packets that is accepted by (G or H) is $\{(2,7),(3,7),(4,7),(2,8),(3,8),(4,8)\}$. Firewalls G and H are called the *component* firewalls of the firewall expression (G or H).

In this chapter, we show that every firewall expression that is specified using one or more firewalls and the three firewall operators "not", "and", and "or" can be represented by a set $\{S_1, S_2, \cdots, S_k\}$ of a special class of firewalls called slices such that the following condition holds. A packet is accepted by a firewall expression iff this packet is accepted by at least one slice in the set of slices that represents the firewall expression.

As an example, the firewall expression (G or H) can be represented by the set of three slices $\{S_1, S_2, S_3\}$ according to Algorithm 4 presented in Section 5.4.

Slice S_1 is defined as follows:

$$\begin{aligned} \big((u_1 \in [1,4]) \wedge (u_2 \in [8,9])\big) &\rightarrow \text{reject} \\ \big((u_1 \in [2,4]) \wedge (u_2 \in [7,9])\big) &\rightarrow \text{accept} \end{aligned}$$

Slice S_2 is defined as follows:

$$\big((u_1 \in [2,3]) \wedge (u_2 \in [7,7])\big) \rightarrow \text{accept}$$

Slice S_3 is defined as follows:

$$\big((u_1 \in [2,4]) \wedge (u_2 \in [7,8])\big) \quad \rightarrow \quad \text{accept}$$

(Notice that, as discussed in Section 5.4, each slice is a firewall that consists of zero or more reject rules followed by exactly one accept rule.)

Similarly, consider a firewall expression (G and H). This firewall expression accepts any packet p iff both polices G and H accept p. The firewall expression (G and H) can be represented by the set of two slices $\{S_4, S_5\}$ according to Algorithm 3 presented in Section 5.4.

Slice S_4 is defined as follows:

$$\big((u_1 \in [1,4]) \wedge (u_2 \in [8,9])\big) \quad \rightarrow \quad \text{reject}$$
$$\big((u_1 \in [2,3]) \wedge (u_2 \in [7,7])\big) \quad \rightarrow \quad \text{accept}$$

Slice S_5 is defined as follows:

$$\big((u_1 \in [1,4]) \wedge (u_2 \in [8,9])\big) \quad \rightarrow \quad \text{reject}$$
$$\big((u_1 \in [2,4]) \wedge (u_2 \in [7,8])\big) \quad \rightarrow \quad \text{accept}$$

Based on the above discussions, this chapter suggests a novel bottom-up design method that can be followed by a designer in designing firewalls. This design method proceeds as follows. First, the designer designs several simple component firewalls. Second, the designer combines these component firewalls using the three firewall operators "not", "and", and "or" into a single

firewall expression FE. Finally, the designer uses the algorithms in Section 6 below to determine that the designed firewall expression FE is adequate, and that FE implies or is equivalent to a desired firewall expression.

As an example, a designer can start by designing two firewalls G and H, then use these two firewalls to design the firewall expression (G and not(H)). This firewall expression accepts every packet that is accepted by firewall G and rejected by firewall H. Then the designer can use Algorithm 7 in Section 5.5 below to prove that this firewall expression implies both firewall G and firewall not(H).

Other methods that can be used in designing firewalls are reported in [28], [29], [44], [56], and [4]. A brief survey of these methods has been presented in Chapter 2.

These design methods, along with the bottom-up method in the current chapter can constitute a library of firewall design methods. When designing a firewall, it is up to the designer to decide which design method in this library will the designer follow to generate the desired firewall.

The rest of this chapter is organized as follows. In Section 5.2, we present our formal definition of firewall expressions and discuss three theorems that state fundamental properties of firewall expressions. In Section 5.3, we discuss an algorithm that can be used to evaluate a given firewall expression for any input stream of packets. In Section 5.4, we introduce the concept of a base of a firewall expression as a set of slices that satisfies the following

condition. For every incoming packet p, the firewall expression accepts p iff at least one slice in the base of the firewall expression accepts p. Also in Section 5.4, we present algorithms for constructing a base for every firewall expression. In Section 5.5, we show that the bases of given firewall expressions can be used to determine whether the given firewall expressions satisfy some logical properties such as adequacy, implication, and equivalence. Finally, we conclude this chapter in Section 5.6.

5.2 Definition of Firewall Expressions

In this section, we define firewall expressions. Informally, a firewall expression is specified using one or more firewalls and three firewall operators: "not", "and", and "or". Each one of these firewall operators can be applied to one or two firewall expressions to produce a firewall expression.

Formally, a ⟨firewall expression FE⟩ is defined recursively as one of the following four options:

A complete firewall G

A complete firewall not(G)

⟨firewall expression FE_1⟩ and ⟨firewall expression FE_2⟩

⟨firewall expression FE_1⟩ or ⟨firewall expression FE_2⟩

An example of a firewall expression is as follows:

(G and not(H)) or (not(G) and H)

In this example, G and H are complete firewalls and are called component

firewalls of the firewall expression. Also "not", "and", and "or" are called firewall operators.

Associated with each firewall expression FE is a packet set PS defined as follows:

- If FE is a complete firewall G,
  then PS is the set of all packets accepted by G
- If FE is a complete firewall not(G),
  then PS is the set of all packets accepted by not(G) or equivalently PS is the set of all packets rejected by G
- If FE is a firewall expression (FE_1 and FE_2),
  then PS is the intersection of two packet sets PS_1 and PS_2 where PS_1 is the packet set associated with FE_1 and PS_2 is the packet set associated with FE_2
- If FE is a firewall expression (FE_1 or FE_2),
  then PS is the union of two packet sets PS_1 and PS_2 where PS_1 is the packet set associated with FE_1 and PS_2 is the packet set associated with FE_2

As an example, the packet set associated with the firewall expression (G and not(H)) is the intersection of the two packet sets PS_1 and PS_2, where PS_1 is the set of all packets accepted by firewall G and PS_2 is the set of all packets accepted by firewall not(H).

Two firewall expressions FE_1 and FE_2 are said to be *equivalent* iff the two packet sets associated with FE_1 and FE_2 are identical.

For example, the firewall expression (G and not(H)) and the firewall expression (not(G) and H) are equivalent.

Let FE be a firewall expression. We adopt the notation not(FE) to denote the firewall expression that is recursively obtained from FE as follows:

- If FE is a complete firewall G,
  then not(FE) denotes the firewall expression not(G)
- If FE is a complete firewall not(G),
  then not(FE) denotes the firewall expression G
- If FE is a firewall expression (FE_1 and FE_2),
  then not(FE) denotes the firewall expression (not(FE_1) or not(FE_2))
- If FE is a firewall expression (FE_1 or FE_2),
  then not(FE) denotes the firewall expression (not(FE_1) and (FE_2))

As an example, not$\big($(G and not(H)) or (not(G) and H)$\big)$ denotes the firewall expression $\big($(not(G) or H) and (G or not(H))$\big)$.

The following three theorems state fundamental properties of firewall expressions.

Theorem 5.2.1. *For every firewall expression FE,* (1) *the packet set associated with the firewall expression* (FE and not(FE)) *is the empty set, and* (2)

the packet set associated with the firewall expression (FE or not(FE)) *is the set* P *of all packets.*

Proof. Our proof of this theorem makes use of the following definition of the "rank" of a firewall expression FE.

The rank k of a firewall expression FE is a non-negative integer defined recursively as follows:

- If FE is a complete firewall G or is a complete firewall not(F), then $k = 0$
- If FE is of the form (FE_1 and FE_2) or is of the form (FE_1 or FE_2), then $k = (1 + \max(k_1, k_2))$, where k_1 is the rank of FE_1 and k_2 is the rank of FE_2

Our proof of this theorem is by induction on the rank k of the firewall expression FE. Details of this proof are presented in [53].

□

Theorem 5.2.2. *For every firewall expression* FE*, the packet set associated with the firewall expression* not(FE) *is* $(P - PS)$*, where* P *is the set of all packets,* PS *is the packet set associated with* FE*, and "−" is the set difference operator. (Note that the packet set* $(P - PS)$ *can be written as the compliment of set* PS*.)*

Proof. Let NS denote the packet set associated with not(FE). Thus, the packet set associated with the firewall expression (FE and not(FE)) is ($PS \bigcap NS$), and the packet set associated with the firewall expression (FE or not(FE)) is ($PS \bigcup NS$). Hence, from Theorem 1, the set ($PS \bigcap NS$) is empty and the set ($PS \bigcup NS$) is the set P of all packets. Therefore, set NS is ($P - PS$). □

A firewall expression FE is said to be *complete* iff for every packet p either FE accepts p or FE rejects p.

Theorem 5.2.3. *Every firewall expression is complete.*

Proof. The proof is by contradiction. Assume that there is a firewall expression FE that is not complete. Thus, there is a packet p such that FE neither accepts p nor rejects p. Hence, from Theorem 2, packet p is neither in the packet set PS associated with FE nor in the packet set ($P - PS$) associated with not(FE). Therefore, packet p is not in the union of the two sets PS and ($P - PS$), which constitutes the set P of all packets. This contradicts the fact that p is a packet in the set P of all packets. □

5.3 Evaluation of Firewall Expressions

In this section, we discuss an algorithm that takes as input any given firewall expression FE and any given packet p and produces as output a determination of whether or not FE accepts p. This algorithm can be used to evaluate the given firewall expression FE for any input stream of packets.

The main idea of this algorithm is to use the input pair (p, FE) to produce a Boolean expression BE, that involves the two Boolean values "TRUE" and "FALSE", and the three Boolean operators "$\neg$", "$\wedge$", and "$\vee$".

The Boolean expression BE corresponding to the pair (p, FE) is required to satisfy one of the following two conditions:

- (FE accepts p) iff (BE is TRUE)
- (FE rejects p) iff (BE is FALSE)

Now consider a firewall expression FE and a packet p as follows:

$$FE \ = \ (G \text{ and } (H \text{ or } I)) \text{ or not}(H)$$

where G, H, and I are complete firewalls. Assume that G accepts p, H rejects p, and I rejects p. The Boolean expression BE corresponding to the pair (p, FE) can be constructed as follows:

- Because G accepts p, replace firewall G in FE by the Boolean value TRUE in BE
- Because H rejects p, replace firewall H in FE by the Boolean value FALSE in BE
- Because I rejects p, replace firewall I in FE by the Boolean value FALSE in BE

- Replace the firewall operator "not" in FE by the Boolean operator "$\neg$" in BE
- Replace the firewall operator "and" in FE by the Boolean operator "$\wedge$" in BE
- Replace the firewall operator "or" in FE by the Boolean operator "$\vee$" in BE
- The Boolean expression BE can now be computed as follows:
  BE = (TRUE $\wedge$ (FALSE $\vee$ FALSE)) $\vee$ $\neg$FALSE
  = FALSE $\vee$ TRUE = TRUE

Because BE is TRUE, we conclude that the given firewall expression FE accepts the given packet p.

Next, we discuss the time complexity for computing the Boolean expression BE that represents a given firewall expression FE and a given packet p. Assume that the given firewall expression FE has m distinct firewalls and k firewall operators. Also assume that each distinct firewall has t attributes (t is usually 5 for firewalls) and at most n rules. Therefore, the time complexity to determine whether each distinct firewall in FE accepts the given packet p is $\mathcal{O}(n \times t)$. The "length" of the constructed Boolean expression BE is $\mathcal{O}(k)$. Thus, the time complexity to construct the Boolean expression is $\mathcal{O}((n \times t \times m) + (m \times k))$. Also, the time complexity of computing the Boolean value of BE is $\mathcal{O}(k^2)$ [51]. Therefore, the time complexity for

constructing the Boolean expression BE and computing its Boolean value is $\mathcal{O}((n \times t \times m) + ((m \times k) + k^2))$.

5.4 Bases of Firewall Expressions

In the next section, Section 5.5, we discuss several properties of firewall expressions and present algorithms to determine whether given firewall expressions satisfy these properties. For example, we present algorithms to determine whether any given two firewall expressions are equivalent.

Our discussion in Section 5.5 is based on two concepts, namely "slices" and "bases of firewall expressions" that we introduce in the current section.

A *slice* is a firewall that consists of zero or more reject rules followed by exactly one accept rule.

Let SS be a set of slices and let FE be a firewall expression. Set SS is said to be a *base* of the firewall expression FE iff the following condition holds. Each packet p is accepted by at least one slice in set SS iff p is in the packet set associated with the firewall expression FE.

The following five algorithms can be applied to any firewall expression FE to construct a slice set SS that is a base of FE.

Algorithm 1

Input: A complete firewall G

Output: A slice set SS that is a base of G

Steps: For each accept rule ar in G, construct a slice sl in SS as follows. All the reject rules that precede rule ar in G are added to slice sl. Then rule ar is added at the end of slice sl.

Correctness: Proof of Correctness is presented in [53].

Time Complexity: A slice may contain up to n rules where n is the number of rules in the input firewall G and adding one rule to a slice takes $\mathcal{O}(t)$ steps where t is the number of attributes in G. So the time complexity to construct each slice is $\mathcal{O}(n \times t)$. There can be at most n slices in SS, one for each accept rule in G. Therefore, the time complexity of Algorithm 1 is of $\mathcal{O}(n^2 \times t)$ where n is the number of rules and t is the number of attributes in G.

End

Algorithm 2

Input: A complete firewall not(G)

Output: A slice set SS that is a base of not(G)

Steps: For each accept rule ar in not(G), construct a slice sl in SS as follows. All the reject rules that precede rule ar in not(G) are added to slice sl. Then rule ar is added at the end of slice sl.

Correctness: The correctness proof of Algorithm 2 is same as the correctness proof of Algorithm 1.

Time Complexity: The time complexity of Algorithm 2 is same as the time complexity of Algorithm 1.

End

Algorithm 3

Input: A firewall expression FE of the form (FE_1 and FE_2)

A slice set SS_1 that is a base of FE_1

A slice set SS_2 that is a base of FE_2

Output: A slice set SS that is a base of FE

Steps: For every slice sl_1 in SS_1 and every slice sl_2 in SS_2, construct a slice sl in SS as follows:

1. The reject rules of slice sl are constructed by merging the reject rules of sl_1 with the reject rules of sl_2 in any order

2. The accept rule of slice sl is constructed by taking the intersection of the predicates of the two accept rules of slices sl_1 and sl_2. If this intersection is empty, then discard slice sl from the base SS of the firewall expression FE.

Correctness: Proof of Correctness is presented in [53].

Time Complexity: The number of slices in SS is $(m_1 \times m_2)$ where m_1 is the number of slices in SS_1 and m_2 is the number of slices in SS_2. The time complexity to construct a slice in SS is of $\mathcal{O}(n_1 \times t + n_2 \times t)$, where n_1 is the number of rules in the largest slice in SS_1, n_2 is the number of rules in the largest slice in SS_2 and t is the number of attributes. Therefore, the time complexity of Algorithm 3 is of $\mathcal{O}((m_1 \times m_2) \times (n_1 \times t + n_2 \times t))$ where m_1 is the number of slices in SS_1, m_2 is the number of slices in SS_2, n_1 is the

number of rules in the largest slice in SS_1, n_2 is the number of rules in the largest slice in SS_2, and t is the number of attributes.

End

Algorithm 4

Input: A firewall expression FE of the form (FE_1 or FE_2)

A slice set SS_1 that is a base of FE_1

A slice set SS_2 that is a base of FE_2

Output: A slice set SS that is a base of FE

Steps: The slice set SS is constructed as the union of the two slice sets SS_1 and SS_2.

Correctness: Proof of Correctness is presented in [53].

Time Complexity: The time complexity of Algorithm 4 is the sum of the time complexity to add all slices of SS_1 to SS and the time complexity to add all slices of SS_2 to SS. The time complexity to add each slice of SS_1 to SS is of $\mathcal{O}(n_1 \times t)$, where n_1 is the number of rules in the largest slice in SS_1 and t is the number of attributes. Similarly, The time complexity to add each slice of SS_2 to SS is of $\mathcal{O}(n_2 \times t)$, where n_2 is the number of rules in the largest slice in $SS2$. Therefore, The time complexity of Algorithm 4 is of $\mathcal{O}((m_1 \times n_1 \times t) + (m_2 \times n_2 \times t))$ where m_1 is the number of slices in SS_1, m_2 is the number of slices in SS_2, n_1 is the number of rules in the largest slice in SS_1, n_2 is the number of rules in the largest slice in SS_2, and t is the number of attributes.

End

Algorithm 5

Input: A firewall expression FE

Output: A slice set SS that is a base of FE

Steps: SS is constructed by recursively applying the following four steps:

1. If FE is a complete firewall G then use Algorithm 1 to construct SS as a base of G

2. If FE is a complete firewall not(G) then use Algorithm 2 to construct SS as a base of not(G)

3. If FE is (FE_1 and FE_2) and SS_1 is a base of FE_1 and SS_2 is a base of FE_2 then use Algorithm 3 to construct SS as a base of FE from the two slice sets SS_1 and SS_2

4. If FE is (FE_1 or FE_2) and SS_1 is a base of FE_1 and SS_2 is a base of FE_2 then use Algorithm 4 to construct SS as a base of firewall expression FE from the two slice sets SS_1 and SS_2

Correctness: The correctness proof of Algorithm 5 follows from the correctness proofs of Algorithms 1, 2, 3 and 4.

Time Complexity: The time complexity of Algorithm 5 depends on the number and type of operators in the input firewall expression FE. The time complexity of Algorithm 5 has been explained in more detail with an example

in [53].

End

5.5 Properties of Firewall Expressions

In this section, we present several important properties of firewall expressions (namely adequacy, implication, and equivalence) and present algorithms that can be used to determine whether any given firewall expression satisfies these properties.

A firewall expression FE is said to be *adequate* iff FE accepts at least one packet. The following algorithm can be used to determine whether any given firewall expression is adequate.

Algorithm 6

Input: A firewall expression FE

Output: A determination of whether FE accepts a packet.

Steps: Construct a base SS of the firewall expression FE using Algorithm 5. For each slice in the constructed base SS, determine whether this slice accepts a packet using the Probing Algorithm [53]. If one or more slices in SS accepts a packet, then FE accepts a packet. Otherwise, FE does not accept any packet.

Time Complexity: Let T denote the time complexity of Algorithm 5 when applied to the input firewall expression to construct its base SS. Also let m be

the number of slices in the constructed base SS and n be the number of rules in the largest slice in SS. As showed in [53], the time complexity of Probing Algorithm to determine whether a slice of n rules and t attributes accepts a packet is of $\mathcal{O}(n^{t+1} \times t)$. Therefore, the time complexity of Algorithm 6 is of $\mathcal{O}(T + (m \times (n^{t+1} \times t)))$.

End

A firewall expression FE_1 is said to *imply* a firewall expression FE_2 iff the packet set associated with the firewall expression (FE_1 and not(FE_2)) is empty. (Note that $FE1$ implies FE_2 iff every packet that is accepted by FE_1 is also accepted by FE_2.)

Theorem 5.5.1. *FE_1 implies FE_2 iff the packet set PS_1 associated with FE_1 is a subset of the packet set PS_2 associated with FE_2.*

Proof. Proof of the Only-If-Part: Assume that FE_1 implies FE_2. Thus, the packet set associated with the firewall expression (FE_1 and not(FE_2)) is empty. From Theorem 2, the packet set associated with not(FE_2) is the set $(P - PS_2)$, where P is the set of all packets. Therefore, the set $\big(PS_1 \bigcap (P - PS_2)\big)$ is empty and PS_1 is a subset of PS_2.

Proof of the If-Part: Assume that the packet set PS_1 associated with FE_1 is a subset of the packet set PS_2 associated with FE_2. Thus, the set $\big(PS_1 \bigcap (P - PS_2)\big)$, where P is the set of all packets, is empty. From Theorem 2, the packet set associated with not(FE_2) is the set $(P - PS_2)$. Therefore,

the packet set associated with the firewall expression (FE_1 and not(FE_2)) is empty and FE_1 implies FE_2.

□

Algorithm 7

Input: Two firewall expressions FE_1 and FE_2

Output: A determination of whether FE_1 implies FE_2

Steps: First, construct a firewall expression FE from the firewall expression (FE_1 and not(FE_2)) by recursively applying "not" to firewall expression FE_2 until "not" is applied only to the constituent component firewalls of FE_2. Second, use Algorithm 6 to determine whether the constructed firewall expression FE accepts a packet. From the definition of "implies", if FE accepts no packet then FE_1 implies FE_2. Otherwise, FE_1 does not imply FE_2.

Time Complexity: The time complexity of the first step of Algorithm 7 is dominated by the time complexity of the second step which uses Algorithm 6. Therefore, the time complexity of Algorithm 7 is of $\mathcal{O}(T + (m \times n^{t+1} \times t))$, where T is the time complexity for constructing the firewall expression FE and its base SS, m is the number of slices in the constructed base SS, n is number of rules in the largest slice in SS, and t is the number of attributes in each slice in SS.

End

Theorem 5.5.2. *Two firewall expressions FE_1 and FE_2 are equivalent iff*

FE_1 implies FE_2 and FE_2 implies FE_1.

Proof. Proof of the Only-If-Part: Assume that FE_1 and FE_2 are equivalent. Thus, the packet set PS_1 associated with FE_1 and the packet set PS_2 associated with FE_2 are identical. Therefore, PS_1 is a subset of PS_2 and PS_2 is a subset of PS_1. From Theorem 2, FE_1 implies FE_2 and FE_2 implies FE_1.

Proof of the If-Part: Assume that FE_1 implies FE_2 and FE_2 implies FE_1. Thus, from Theorem 2, PS_1 is a subset of PS_2 and PS_2 is a subset of PS_1. Therefore, the packet set PS_1 associated with FE_1 and the packet set PS_2 associated with FE_2 are identical and the two firewall expressions FE_1 and FE_2 are equivalent. □

Algorithm 8

Input: Two firewall expressions FE_1 and FE_2

Output: A determination of whether FE_1 and FE_2 are equivalent

Steps: Use Algorithm 7 twice to determine: (1) whether FE_1 implies FE_2 and (2) whether FE_2 implies FE_1. From Theorem 5, if FE_1 implies FE_2 and FE_2 implies FE_1, then FE_1 and FE_2 are equivalent. Otherwise, also from Theorem 5, FE_1 and FE_2 are not equivalent.

Time Complexity: The time complexity of Algorithm 8 is twice the time complexity of Algorithm 7.

End

5.6 Chapter Summary

The main contribution of this chapter is to present a generalization of firewalls called firewall expressions. Each firewall expression is specified using one or more firewalls and the three firewall operators "not", "and", and "or". We showed that each firewall expression can be represented by a set of slices called a base of the firewall expression. We also showed that the bases of given firewall expressions can be used to determine whether the given firewall expressions satisfy some desired properties of adequacy, implication, and equivalence. Finally, we showed that firewall expressions can be utilized to support bottom-up methods for designing firewalls.

A concrete running example has been presented in [53] to illustrate the utility of some of the algorithms presented in this chapter.

The authors in [38, 41] investigated a novel representation of firewalls as finite automata rather than as sequences of rules. They show later in [40], how to use the automata representation of a given firewall to determine whether the given firewall satisfies some desired properties of adequacy, implication, and equivalence. They also showed in a recent work [39] that a firewall expression can also be represented as finite automata.

It has been shown in [18] that the problems of determining whether given firewalls satisfy some desired properties of adequacy, implication, and equivalence are all NP-hard. From this fact and the fact that each (complete) firewall is also a firewall expression, it follows that the problems of determining

whether given firewall expressions satisfy some desired properties of adequacy, implication, and equivalence are also NP-hard. Indeed, the time complexities of Algorithms 6, 7, and 8 that can be used to determine whether given firewall expressions satisfy some desired properties of adequacy, implication, and equivalence are all exponential.

There are two main approaches to face the NP-hardness of determining whether given firewall expressions satisfy some desired properties of adequacy, implication, and equivalence. The first approach is to use SAT solvers, for example as discussed in [33], [77], and [5], to determine whether given firewall expressions satisfy some desired properties of adequacy, implication, and equivalence. Note that the time complexity of using SAT solvers is polynomial in most practical situations.

The second approach is to use probabilistic algorithms [1]. Note that the time complexities of probabilistic algorithms are always polynomial but unfortunately these algorithms can yield wrong determinations in rare cases.

Chapter 6

Outsourcing of Firewall Expressions

6.1 Introduction

In Chapter 5, we presented a generalization of firewalls called firewall expressions. A firewall expression is specified using one or more firewalls and the three firewall operators: "not", "and", and "or". An example of a firewall expression FE is as follows:

$$FE = (G \text{ and } \text{not}(H)) \text{ or } \text{not}(G)$$

In this example, G and H are two firewalls, called the component firewalls of FE, "not", "and", and "or" are firewall operators. This firewall expression accepts a packet p iff firewall G accepts p and firewall H rejects p or firewall G rejects p. In the rest of this chapter, when we mention 'firewall expression FE', we mean the firewall expression in the above example and when we mention 'component firewalls of FE', we mean firewalls G and H mentioned above.

We now introduce a generalization of firewall systems, called *expression systems*, whose underlying firewall is a firewall expression. Like a firewall system, an expression system can be used as a packet filter when placed at the entry point of an enterprise network to examine the packets that attempt to enter the network and decide based on an underlying firewall expression

whether to accept or reject each of these packets. If the expression system determines that packet p is to be rejected, then the system discards packet p. Otherwise, the expression system determines that packet p is to be accepted and in this case the system forwards p to the enterprise network.

When part of the tasks that need to be executed to implement and manage an expression system are executed by public clouds, then the resulting system is called an *outsourced expression system*. Our goal in this chapter is to design an outsourced expression system such that the resulting system prevents the attacks that can be caused by public clouds that are used to implement the system.

6.2 Expression Systems

For firewall expression FE, we can define an *Expression System* as a system that takes as input any packet p and determines whether packet p is accepted or rejected according to the firewall expression FE. In this case, we call FE the underlying firewall expression of the expression system.

The firewall expression FE has two component firewalls G and H. Figure 6.1 presents the architecture of the expression system whose underlying firewall expression is FE. This expression system has 3 components: one firewall system for G , one firewall system for H, and one decision unit which we call an overall decision unit.

When a packet p passes this expression system, p is first forwarded to

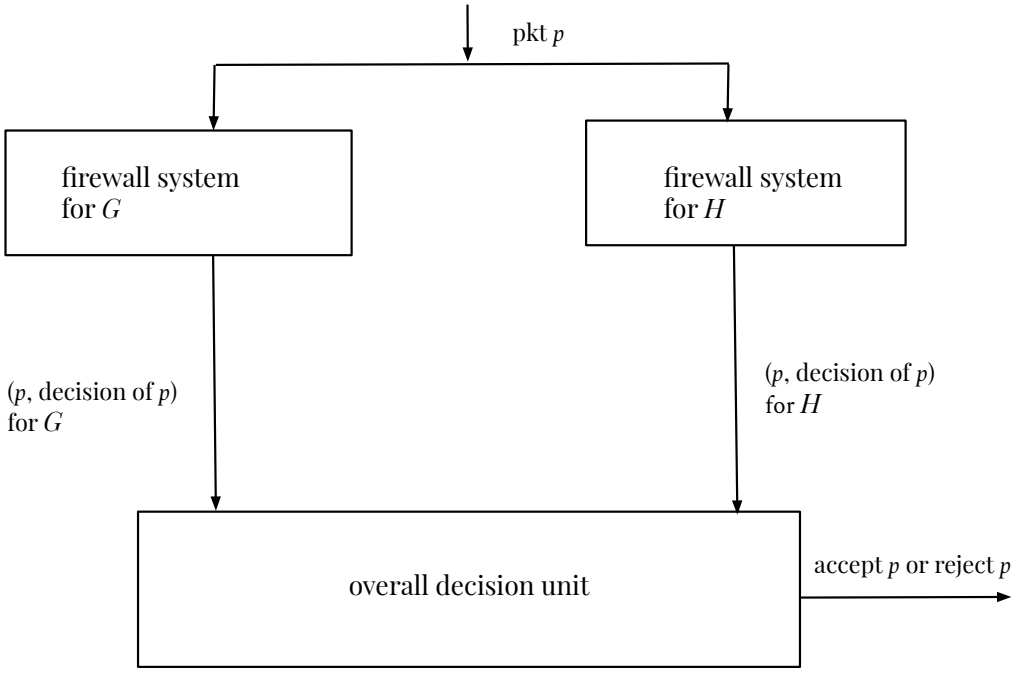

Figure 6.1: Expression system for firewall expression FE which has two component firewalls G and H

each of the two component firewall systems. The architecture of each component firewall system is same as that of a regular firewall system presented in Figure 1.1.

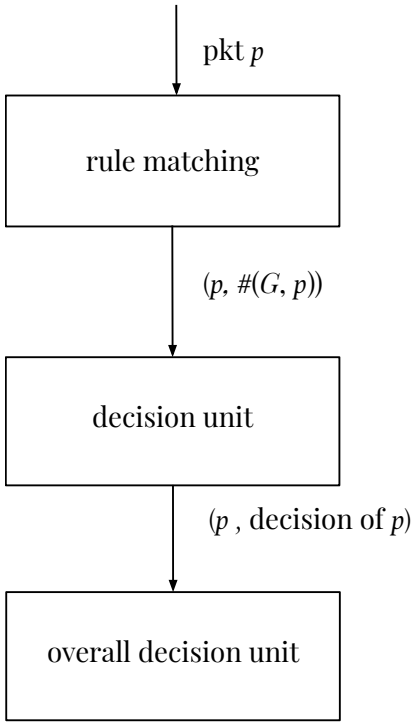

Figure 6.2: Firewall system for component firewall G

Figure 6.2 presents the architecture of the firewall system for G which consists of two units: a rule matching unit and a decision unit. When p enters this firewall system, p is first forwarded to the rule matching unit. Then, the rule matching unit uses the underlying firewall G to compute the sequence number $\#(G, p)$ of the first match rule in G for p. Next, the rule matching unit forwards the pair $(p, \#(G, p))$ to the decision unit. Finally, the decision

unit takes as input packet p and the sequence number $\#(G, p)$ received from the rule matching unit and uses firewall G to compute the decision ("accept" or "reject") of the rule whose sequence number is $\#(G, p)$. After computing the decision for packet p, the decision unit sends the pair (p, decision of p) to the overall decision unit.

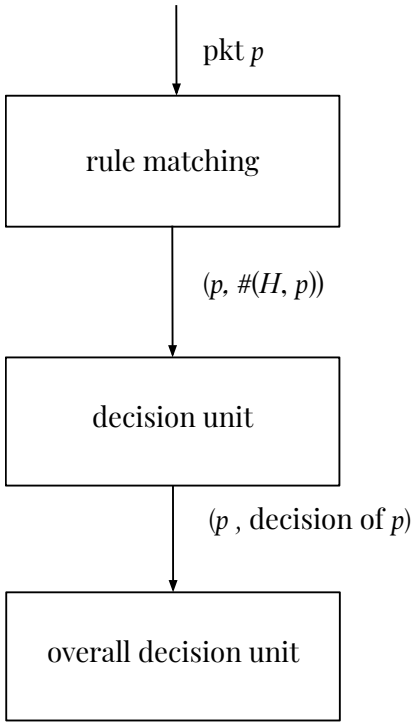

Figure 6.3: Firewall system for component firewall H

Figure 6.3 presents the architecture of the firewall system for H which consists of two units: a rule matching unit and a decision unit. Similar to the system in Figure 6.2, this system computes the decision for packet p according to the underlying firewall H and sends the pair (p, decision of p) to the overall

decision unit

The task of the overall decision unit is to take as input two pairs of (p, decision of p), one from the firewall system for G and one from the firewall system for H, and compute a decision for packet p according to the underlying firewall expression FE. If the overall decision unit determines that the decision for packet p is "accept", then the decision unit forwards p to the enterprise network. Otherwise, the overall decision unit determines that the decision for packet p is "reject", then the decision unit discards packet p and prevents it from entering the network.

6.3 Outsourced Expression Systems

In this section, we design an outsourced expression system whose underlying firewall expression FE has two component firewalls G and H. Figure 6.4 shows the architecture of the outsourced expression system for the firewall expression FE. (Extending the discussion to designing an outsourced expression system whose underlying firewall expression has any number of component firewalls is straight forward.)

Our outsourced expression system is obtained from the expression system in Figure 6.1 by replacing the firewall system for component firewall G with a private system for G and by replacing the firewall system for component firewall H with a private system for H. (Recall that the private system for any underlying firewall F has been presented in Section 3.5 in Chapter 3.)

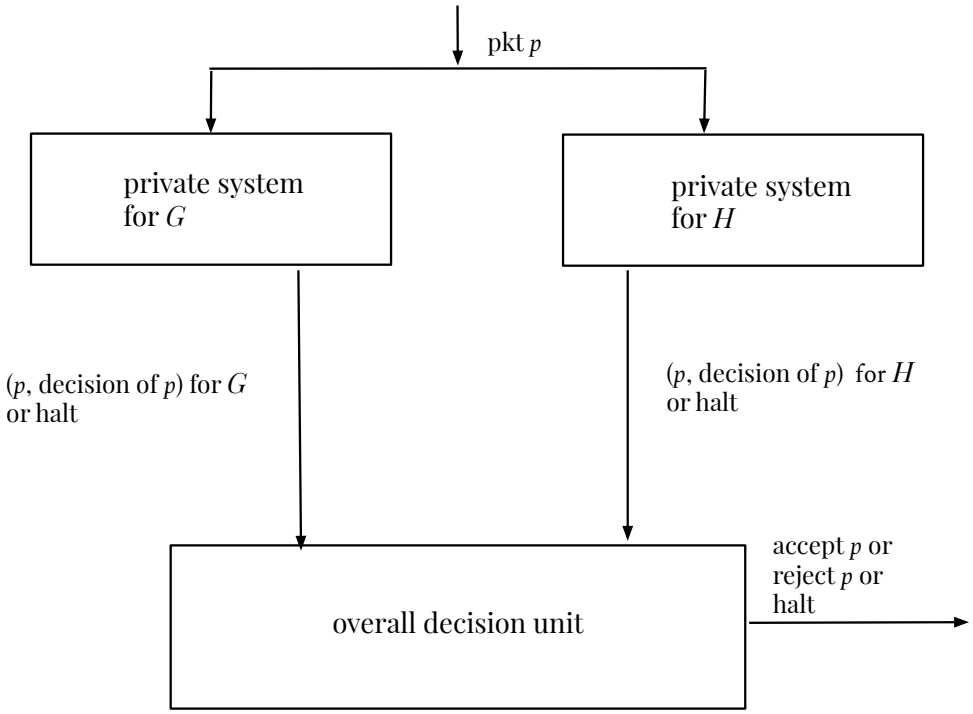

Figure 6.4: Outsourced expression system for firewall expression FE that has two component firewalls G and H

Figure 6.5 shows the private system for G that consists of two identical rule matching units and a verifiable decision unit. The two rule matching units are hosted in two public clouds C_1 and C_2. Each rule matching unit uses an incomplete version IG of the underlying firewall G. Note that the incomplete version of G is the same as G except that the decisions of all the rules in G are unspecified. The verifiable decision unit uses the underlying firewall G.

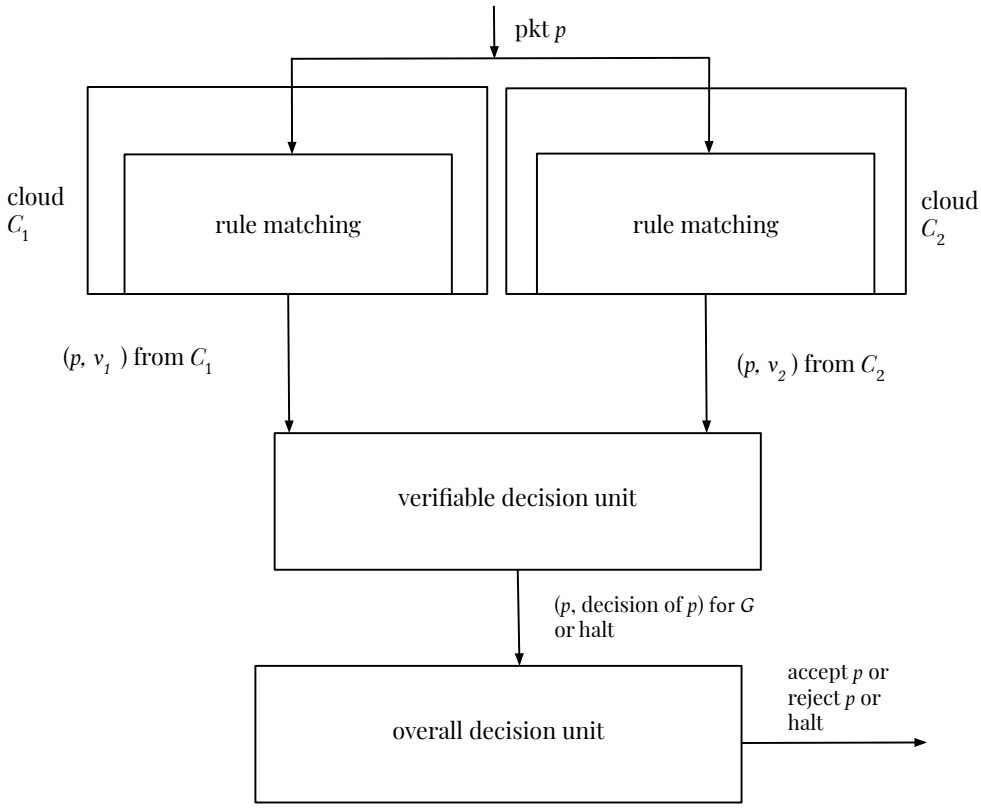

Figure 6.5: Private system for component firewall G

Figure 6.6 shows the private system for H that consists of two identical rule matching units and a verifiable decision unit. The two rule matching units

are hosted in two public clouds C_3 and C_4. Each rule matching unit uses an incomplete version IH of the underlying firewall H. Note that the incomplete version of H is the same as H except that the decisions of all the rules in H are unspecified. The verifiable decision unit uses the underlying firewall H.

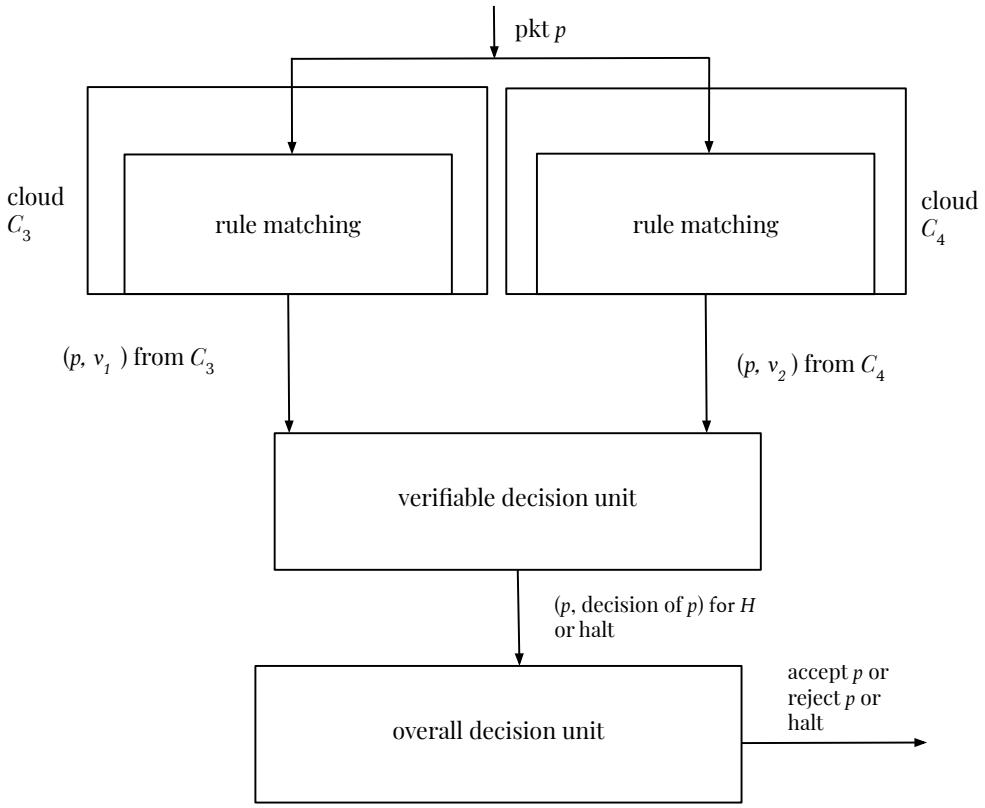

Figure 6.6: Private system for component firewall H

6.4 Execution of Outsourced Expression Systems

Assume that a packet p attempts to pass the expression system in Figure 6.4. Then packet p is directed to both the private system for G and

the private system for H.

When packet p enters the private system for G, p is first directed to the rule matching unit hosted in cloud C_1 so that C_1 can compute a sequence number v_1 using IG and send the pair (p, v_1) to the verifiable decision unit. Also, packet p is directed to the rule matching unit hosted in cloud C_2 so that C_2 can compute a sequence number v_2 using IG and send the pair (p, v_2) to the verifiable decision unit.

Cloud C_1 computes v_1 as the sequence number $\#(IG, p)$ of the first match rule in IG for p. Similarly, cloud C_2 computes v_2 as the sequence number $\#(IG, p)$ of the first match rule in IG for p.

After the verifiable decision unit receives the two pairs (p, v_1) and (p, v_2), the decision unit checks whether the two sequence numbers v_1 and v_2 are equal. If v_1 and v_2 are equal, then the decision unit uses the underlying firewall G to determine the decision ("accept" or "reject") of the rule whose sequence number is v_1. After computing the decision for packet p, the verifiable decision unit sends the pair (p, decision of p) to the overall decision unit of the expression system.

On the other hand, if the two sequence numbers v_1 and v_2 are not equal, then the verifiable decision unit concludes that one of the two pairs (p, v_1) and (p, v_2) is corrupted as these pairs are being transferred from the rule matching unit to the verifiable decision unit. In this case, the verifiable decision unit "issues a halt" to the overall decision unit.

When the overall decision unit receives a "halt" command from the verifiable system of the private system for G or from the verifiable system of the private system for H, the overall decision unit puts the outsourced expression system into a halt so that no more incoming packet can enter this expression system. The private system for H works same as the private system for G.

If the overall decision unit receives (p, decision of p) from the private system for G and (p, decision of p) from the private system for H as shown in Figure 6.7, the overall decision unit computes the decision for packet p as follows.

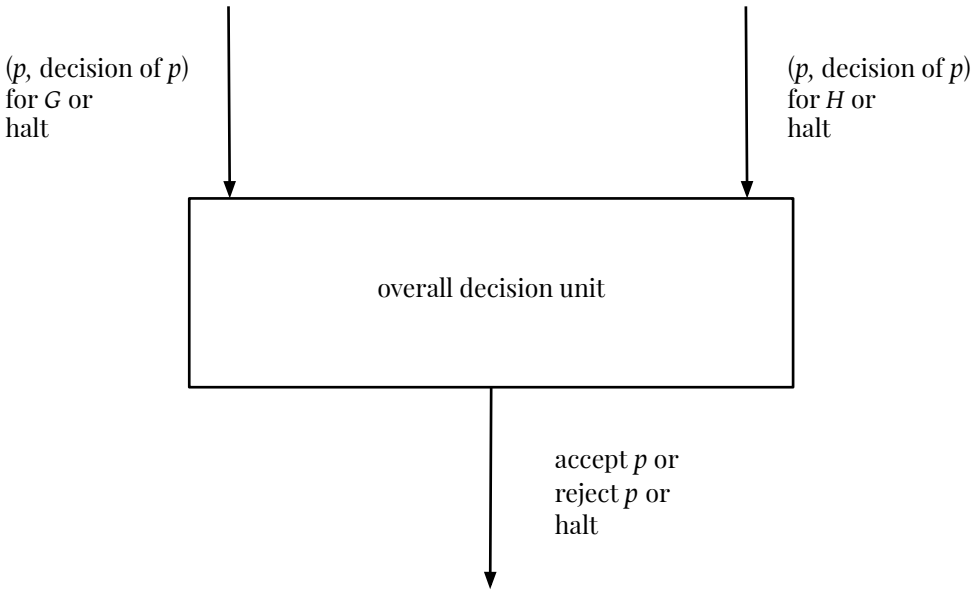

Figure 6.7: The overall decision unit

Let d_G denote the decision of p received by the overall decision unit

from the private system for G, and let d_H denote the decision of p received by the overall decision unit from the private system for H.

- The overall decision unit computes a Boolean expression BE for the pair (p, FE) as follows. If d_G is "accept", then every occurrence of G in FE is replaced by TRUE in BE. Otherwise every occurrence of G in FE is replaced by FALSE in BE. Similarly, if d_H is "accept", then every occurrence of H in FE is replaced by TRUE in BE. Otherwise every occurrence of H in FE is replaced by FALSE in BE. Moreover, the firewall operators "not", "and", and "or" in FE are replaced by the Boolean operators "$\neg$", "$\wedge$", and "$\vee$" respectively in BE.

- If BE evaluates to TRUE, then the decision for p is "accept". Otherwise, BE evaluates to FALSE and the decision for p is "reject".

If the computed decision for packet p is "reject", then the overall decision unit discards packet p. Otherwise, the decision for packet p is "accept" and in this case the decision unit forwards packet p to the enterprise network.

6.5 Security of Outsourced Expression Systems

In Section 3.3 in Chapter 3, we argued that an outsourced system for any underlying firewall F where the rule matching unit is hosted in a public cloud C is vulnerable to two types of security attacks: verifiability attacks and privacy attacks. Later in Section 3.4 and in Section 3.5 in Chapter 3, we

showed how to modify the outsourced system for the underlying firewall F to make sure that verifiability attacks and privacy attacks cannot occur in the modified system. This modified system is called the private system for firewall F.

The outsourced expression system presented in Section 6.3 in the current chapter uses two private systems for component firewalls G and H. Each private system uses two public clouds to outsource the two rule matching units. For example, the private system for G outsources the two rule matching units to two public clouds C_1 and C_2, where both clouds are sensible and they are non-colluding. By applying Theorem 3.4.1 to the private system for component firewall G, we conclude that verifiability attacks cannot occur in the private system for G.

Similarly, the private system for H outsources the two rule matching units to two public clouds C_3 and C_4 where both clouds are sensible and they are non-colluding. By applying Theorem 3.4.1 to the private system for component firewall H, we conclude that verifiability attacks cannot occur in the private system for H.

Since no verifiability attack can occur in each private system of the outsourced expression system, we conclude that no verifiability attack can occur in the outsourced expression system.

By Theorem 3.5.1 the private system for the component firewall G prevents privacy attacks from occurring because neither of the two clouds C_1

or C_2 knows the underlying firewall G. (Note that neither of the two clouds C_1 and C_2 knows the firewall expression FE and the underlying firewall H.)

Similarly, By Theorem 3.5.1 the private system for the component firewall G prevents privacy attack from occurring neither of the two clouds C_3 or C_4 knows the underlying firewall H. (Note that neither of the two clouds C_3 and C_4 knows the firewall expression FE and the underlying firewall G.)

Since no privacy attack can occur in each private system of the outsourced expression system, we conclude that no privacy attack can occur in the outsourced expression system.

6.6 Chapter Summary

The main contribution of this chapter is two-fold.

First, we showed that it is possible to design a packet filter for an enterprise network by choosing a generalized firewall model, namely firewall expression, which is a combination of multiple firewalls. For this model, we designed a generalized firewall system, called an expression system. An expression system takes as input a packet p and determines whether to accept or reject p according to an underlying firewall expression. We discussed the architecture of an expression system in Section 6.2.

Second, we designed an outsourced expression system using public clouds (presented in Section 6.3). We described the execution of our designed outsourced expression system in Section 6.4. In this system, a private system

is used for each component firewall of the underlying firewall expression. We discussed in Section 6.5 that verifiability and privacy attack cannot occur in our outsourced expression system.

Chapter 7

Conclusion and Future Work

In this dissertation, we identified a class of outsourced systems whose rule matching units are executed in public clouds. Since public clouds are usually unreliable, we further identified that the outsourced systems obtained by executing the rule matching units in public clouds are vulnerable to two types of security attacks: verifiability attacks and privacy attacks.

Prior outsourced systems that exist in literature either defend against verifiability attacks or defend against privacy attacks. But none of these system defends against both types of attacks. Our main contribution in this dissertation is to design several outsourced systems whose rule matching units are executed in public clouds such that the resulting systems prevent verifiability and privacy attacks from occurring.

Every outsourced system is built on top of an underlying firewall. We present the formal definition and example of an underlying firewall in Chapter 2. We have identified a special class of firewalls, called partially specified firewalls in Chapter 4. We have also identified a generalization of firewalls, called firewall expressions in Chapter 5. For each of these classes, we designed outsourced system that prevents both verifiability and privacy attacks from

occurring.

In Chapter 3, we first presented the architecture of an outsourced system for firewalls which has one rule matching unit and one decision unit, where the rule matching unit is executed in a public cloud. Then, we formally specified verifiability and privacy attacks that can occur in this outsourced system. Next, we modified this outsourced system to a system, called the private system. The design of the private system involved two identical rule matching units which are executed in two public clouds. The private system prevents both verifiability and privacy attacks under the assumption that the two public clouds used in this system are sensible and non-colluding.

We introduced partially specified firewalls in Chapter 4. In a partially specified firewall the decisions of some of the rules in the firewall are not specified. To design an outsourced system for partially specified firewall we used the private system designed in Chapter 3 for firewalls. In Chapter 4, we showed that every partially specified firewall has an equivalent (fully specified) firewall. Thus, we achieved an outsourced system for any partially specified firewall PF by first obtaining an equivalent firewall F from PF, and then designing a private system for firewall F.

In Chapter 5, we presented a generalization of firewalls called firewall expressions which is specified using one or more component firewalls and three firewall operators: "not", "and", and "or". For any underlying firewall expression FE, we defined an *Expression System* as a special class of firewall systems that takes as input any packet p and determines whether the under-

lying firewall expression FE accepts or rejects packet p.

We designed an outsourced expression system for any underlying firewall expression FE in Chapter 6. We achieved this outsourced expression system by using a private system, presented in Chapter 3, for each component firewall of FE and combining these private systems through an overall decision unit to determine whether any packet is accepted or rejected according to the underlying firewall expression FE.

Although we have made a number of contributions to design of outsourced systems, several avenues of future work still remain. Below we discuss some of the avenues for future work:

- The private system presented in Chapter 3 involves two public clouds system and this system can prevent both verifiability and privacy attacks under the assumption that the two public clouds are sensible and are non-colluding. An interesting open problem is to design a private system that can prevent both verifiability and privacy attacks when the two public clouds are sensible and *can be* colluding.

  A high-level idea of how to proceed to solve this problem is discussed below. One can proceed by designing two firewalls F_1 and F_2 from the underlying firewall F such that F_1 and F_2 are 'different' but they are functionally equivalent to F. One might use the concept of partially specified firewalls discussed in Chapter 4 to create two different but functionally equivalent firewalls from an underlying firewall.

Firewalls F_1 and F_2 are different in a way such that for any incoming packet p, the sequence number $\#(F_1, p)$ may not be same as the sequence number $\#(F_2, p)$. Moreover, the first match rule in F in p can be either the first match rule or the second match rule in F_1 for p. Similarly, the first match rule in F in p can be either the first match rule or the second match rule in F_2 for p. The mapping between F and F_1 and the mapping between F and F_2 should be pre-calculated and stored at the enterprise side. It will also be required that if the first match rule in F in p is mapped to the second match rule in firewall F_i for packet p, then the first match rule in F_i for p is not mapped to any rule in F.

After obtaining two different firewalls from F, one can then modify the private system presented in Chapter 3 as follows. One rule matching unit can be executed in a public cloud C_1 based on firewall F_1 and another rule matching unit can be executed in a public cloud C_2 based on firewall F_2. The two public clouds C_1 and C_2 are sensible but can be colluding. Each cloud C_i will be required to send the sequence numbers of both the first and second match rules for any incoming packet p based on the underlying firewall F_i. The decision unit will then use the pre-calculated mappings between F and F_1 and between F and F_2 to determine that each mapping resolves to the same rule in F, which is the first match rule in F for p. The high-level solution presented above merits further research.

- The firewall model considered in this dissertation is stateless. A firewall

F is called stateless when a packet is accepted or rejected by F only based on the rules in F. A model for designing stateful firewalls has been presented in [27]. In this model, each stateful firewall has a variable set called the state of the firewall, which is used to store some packets that the firewall has accepted previously and needs to remember in the near future. A packet is accepted or rejected by a stateful firewall F not only based on the rules in F but also based on the state of firewall F i.e, the packets that have been previously accepted by F.

One open problem is to extend the techniques presented in this dissertation to design outsourced systems for stateful firewalls such that the resulting systems can prevent both verifiability and privacy attacks from occurring.

The authors in [37] suggest that their outsourced firewall system that is designed for stateless firewall can be extended for stateful firewall by storing the state of the firewall in the clear in the cloud. However, their designed system defends only against privacy attacks. No outsourced system has been proposed yet for stateful firewalls that can defend against both verifiability and privacy attacks. To use the techniques presented in this dissertation to design outsourced systems for stateful firewalls, one must first find the answer of the following question. Where to store the state of the firewall such that the verifiability and privacy attacks are prevented and also purpose of outsourcing is achieved?

- In this dissertation, we developed methods to prevent both verifiability

and privacy attacks for firewall outsourcing. The problems of extending these techniques for outsourcing other middleboxes (such as Intrusion Detection System (IDS [52]), Network Address Translation (NAT [65])) that defend against both verifiability and privacy attacks require further research.

Several middlebox outsourcing techniques, for example [43, 62], and [75] have already been presented in the literature. However, none of these techniques defends against both verifiability and privacy attacks.

Bibliography

[1] Hrishikesh B Acharya and Mohamed G Gouda. Linear-time verification of firewalls. In *Proceedings of the 17th IEEE International Conference on Network Protocols (ICNP)*, pages 133–140. IEEE, 2009.

[2] Hrishikesh B Acharya and Mohamed G Gouda. Projection and division: Linear-space verification of firewalls. In *Proceedings of the 30th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 736–743. IEEE, 2010.

[3] Hrishikesh B Acharya and Mohamed G Gouda. Firewall verification and redundancy checking are equivalent. In *INFOCOM, 2011 Proceedings IEEE*, pages 2123–2128. IEEE, 2011.

[4] Hrishikesh B Acharya, Aditya Joshi, and Mohamed G Gouda. Firewall modules and modular firewalls. In *Proceedings of the 18th IEEE International Conference on Network Protocols (ICNP)*, pages 174–182. IEEE, 2010.

[5] Hrishikesh B Acharya, Satyam Kumar, Mohit Wadhwa, and Ayush Shah. Rules in play: On the complexity of routing tables and firewalls. In *Proceedings of the 24th IEEE International Conference on Network Protocols (ICNP)*. IEEE, 2016.

[6] Ehab Al-Shaer, Hazem Hamed, Raouf Boutaba, and Masum Hasan. Conflict classification and analysis of distributed firewall policies. *IEEE journal on selected areas in communications*, 23(10):2069–2084, 2005.

[7] Ehab S Al-Shaer and Hazem H Hamed. Discovery of policy anomalies in distributed firewalls. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 4, pages 2605–2616. IEEE, 2004.

[8] Ehab S Al-Shaer and Hazem H Hamed. Modeling and management of firewall policies. *IEEE Transactions on network and service management*, 1(1):2–10, 2004.

[9] Anne Anderson, Anthony Nadalin, B Parducci, D Engovatov, H Lockhart, M Kudo, P Humenn, S Godik, S Anderson, S Crocker, et al. Extensible access control markup language (xacml) version 1.0. *OASIS*, 2003.

[10] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of np. *Journal of the ACM (JACM)*, 45(1):70–122, 1998.

[11] Meryeme Ayache, Mohammed Erradi, and Bernd Freisleben. curlx: A middleware to enforce access control policies within a cloud environment. In *Proceedings of the 2015 IEEE Conference on Communications and Network Security (CNS 2015)*, pages 771–772. IEEE, 2015.

[12] Meryeme Ayache, Mohammed Erradi, Ahmed Khoumsi, and Bernd Freisleben. Analysis and verification of xacml policies in a medical cloud environment. *Scalable Computing: Practice and Experience*, 17(3):189–206, 2016.

[13] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[14] D Brent Chapman, Elizabeth D Zwicky, and Deborah Russell. *Building internet firewalls*. O'Reilly & Associates, Inc., 1995.

[15] Wen Ding, William Yurcik, and Xiaoxin Yin. Outsourcing internet security: Economic analysis of incentives for managed security service providers. In *International Workshop on Internet and Network Economics*, pages 947–958. Springer, 2005.

[16] Qunfeng Dong, Suman Banerjee, Jia Wang, Dheeraj Agrawal, and Ashutosh Shukla. Packet classifiers in ternary cams can be smaller. In *ACM SIGMETRICS Performance Evaluation Review*, volume 34, pages 311–322. ACM, 2006.

[17] Ehab S Elmallah, Hrishikesh B Acharya, and Mohamed G Gouda. Incremental verification of computing policies. In *Proceedings of the 16th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, volume 8756 of *Lecture Notes in Computer Science*, pages 226–236. Springer, 2014.

[18] Ehab S Elmallah and Mohamed G Gouda. Hardness of firewall analysis. In *Proccedings of the 2nd International Conference on NETworked sYStems (NETYS)*, volume 8593 of *Lecture Notes in Computer Science*, pages 153–168. Springer, 2014.

[19] David Eppstein and S Muthukrishnan. Internet packet filter management and rectangle geometry. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 827–835. Society for Industrial and Applied Mathematics, 2001.

[20] Seyed Kaveh Fayazbakhsh, Michael K Reiter, and Vyas Sekar. Verifiable network function outsourcing: Requirements, challenges, and roadmap. In *Proceedings of 2013 ACM Workshop on Hot Topics in Middleboxes and Network Function Virtualization (HotMiddlebox 2013)*, pages 25–30. ACM, 2013.

[21] MV Fernando, Paulo Esteves, Christian Esteve, et al. Software-defined networking: A comprehensive survey. *PROCEEDINGS OF THE IEEE*, 2015.

[22] Mike Frantzen, Florian Kerschbaum, E Eugene Schultz, and Sonia Fahmy. A framework for understanding vulnerabilities in firewalls using a dataflow model of firewall internals1. *Computers & Security*, 20(3):263–270, 2001.

[23] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Annual Cryptology Conference*, pages 465–482. Springer, 2010.

[24] Craig Gentry et al. Fully homomorphic encryption using ideal lattices. In *Stoc*, volume 9, pages 169–178, 2009.

[25] Glen Gibb, Hongyi Zeng, and Nick McKeown. Outsourcing network functionality. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 73–78. ACM, 2012.

[26] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186–208, 1989.

[27] Mohamed G Gouda and Alex X Liu. A model of stateful firewalls and its properties. In *Proceedings of the 2005 International Conference on Dependable Systems and Networks (DNS 2005)*, pages 128–137. IEEE, 2005.

[28] Mohamed G Gouda and Alex X Liu. Structured firewall design. *Computer Networks*, 51(4):1106–1120, 2007.

[29] Mohamed G Gouda and X-YA Liu. Firewall design: Consistency, completeness, and compactness. In *Proceedings of the 24th International Conference on Distributed Computing Systems*, pages 320–327. IEEE, 2004.

[30] Mohammad Hajjat, Xin Sun, Yu-Wei Eric Sung, David Maltz, Sanjay Rao, Kunwadee Sripanidkulchai, and Mohit Tawarmalani. Cloudward bound: planning for beneficial migration of enterprise applications to the

cloud. *ACM SIGCOMM Computer Communication Review*, 41(4):243–254, 2011.

[31] Adiseshu Hari, Subhash Suri, and Guru Parulkar. Detecting and resolving packet filter conflicts. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1203–1212. IEEE, 2000.

[32] Peng He, Gaogang Xie, Kavé Salamatian, and Laurent Mathy. Meta-algorithms for software-based packet classification. In *Network Protocols (ICNP), 2014 IEEE 22nd International Conference on*, pages 308–319. IEEE, 2014.

[33] Marijn JH Heule, Rezwana Reaz, Hrishikesh B Acharya, and Mohamed G Gouda. Analysis of computing policies using sat solvers (short paper). In *Proccedings of the 18th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 190–194. Springer, 2016.

[34] Daniel Hoffman and Kevin Yoo. Blowtorch: a framework for firewall test automation. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 96–103. ACM, 2005.

[35] Hongxin Hu, Gail-Joon Ahn, and Ketan Kulkarni. Detecting and resolving firewall policy anomalies. *IEEE Transactions on dependable and secure computing*, 9(3):318–331, 2012.

[36] Seny Kamara, Sonia Fahmy, Eugene Schultz, Florian Kerschbaum, and Michael Frantzen. Analysis of vulnerabilities in internet firewalls. *Computers & Security*, 22(3):214–232, 2003.

[37] Amir R Khakpour and Alex X Liu. First step toward cloud-based firewalling. In *Proceedings of the 31st IEEE International Symposium on Reliable Distributed Systems (SRDS 2012)*, pages 41–50. IEEE, 2012.

[38] Ahmed Khoumsi, Mohamed Erradi, Meryeme Ayache, and Wadie Krombi. An approach to resolve np-hard problems of firewalls. In *Proceedings of the 4th International Conference on NETworked sYStems (NETYS)*. Springer, 2016.

[39] Ahmed Khoumsi and Mohammed Erradi. Automata-based bottom-up design of conflict-free security policies specified as policy expressions. In *International Conference on Networked Systems*, pages 343–357. Springer, 2018.

[40] Ahmed Khoumsi, Wadie Krombi, and Mohammed Erradi. A formal approach to verify completeness and detect anomalies in firewall security policies. In *Proceedings of the 7th International Symposium on Foundations and Practice of Security*, pages 221–236. Springer, 2014.

[41] Wadie Krombi, Mohammed Erradi, and Ahmed Khoumsi. Automata-based approach to design and analyze security policies. In *Proceedings of the 12th Annual International Conference on Privacy, Security and Trust (PST)*, pages 306–313. IEEE, 2014.

[42] Tytus Kurek, Marcin Niemiec, and Artur Lason. Taking back control of privacy: A novel framework for preserving cloud-based firewall policy confidentiality. *International Journal of Information Security*, 15(3):235–250, 2016.

[43] Chang Lan, Justine Sherry, Raluca Ada Popa, Sylvia Ratnasamy, and Zhi Liu. Embark: Securely outsourcing middleboxes to the cloud. In *Proceedings of the 13th USENIX Symposium on Netwroked System Design and Implementation (NSDI 2016)*, volume 16, pages 255–273, 2016.

[44] Alex X Liu and Mohamed G Gouda. Diverse firewall design. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 19(9):1237–1251, 2008.

[45] Alex X Liu and Mohamed G Gouda. Complete redundancy removal for packet classifiers in tcams. *IEEE Transactions on Parallel and Distributed Systems*, 21(4):424–437, 2010.

[46] Alex X Liu and Mohamed G Gouda. Complete redundancy removal for packet classifiers in tcams. *IEEE Transactions on Parallel and Distributed Systems*, 21(4):424–437, 2010.

[47] Alex X Liu, Chad R Meiners, and Eric Torng. Tcam razor: A systematic approach towards minimizing packet classifiers in tcams. *IEEE/ACM Transactions on Networking (TON)*, 18(2):490–500, 2010.

[48] Alain Mayer, Avishai Wool, and Elisha Ziskind. Fang: A firewall analysis engine. In *IEEE Symposium on Security and Privacy*, pages 177–187. IEEE, 2000.

[49] Luca Melis, Hassan Jameel Asghar, Emiliano De Cristofaro, and Mohamed Ali Kaafar. Private processing of outsourced network functions: Feasibility and constructions. In *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, pages 39–44. ACM, 2016.

[50] Rolf Oppliger. Internet security: firewalls and beyond. *Communications of the ACM*, 40(5):92–102, 1997.

[51] Christos H. Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003.

[52] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer networks*, 31(23-24):2435–2463, 1999.

[53] Rezwana Reaz, H. B. Acharya, Ehab S. Elmallah, Jorge A. Cobb, and Mohamed G Gouda. Policy expres- sions and the bottom-up design of computing policies. Technical Report No. TR-17-01, Department of Computer Science, The Universisty of Texas at Austin, 2017.

[54] Rezwana Reaz, H. B. Acharya, Ehab S. Elmallah, Jorge A. Cobb, and Mohamed G. Gouda. Policy expressions and the bottom-up design of

computing policies. In *Proceedings of the 5th International Conference on Netwroked Systems (NETYS 2017)*, pages 151–165. Springer, 2017.

[55] Rezwana Reaz, H. B. Acharya, Ehab S. Elmallah, Jorge A. Cobb, and Mohamed G Gouda. Policy expressions and the bottom-up design of computing policies. *Computing*, 101(9):1307–1326, 2019.

[56] Rezwana Reaz, Muqeet Ali, Mohamed G Gouda, Marijn JH Heule, and Ehab S Elmallah. The implication problem of computing policies. In *Proceedings of the 17th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 109–123. Springer, 2015.

[57] Rezwana Reaz, Ehab S. Elmallah, and Mohamed G. Gouda. Executing firewalls in public clouds. In *Proceedings of the 10th international conference computing, communication and networking technologies (ICCCNT).* IEEE, 2019. (Accepted for publication).

[58] Ted Ritter. Network-based firewall: Extending the firewall into the cloud. `https://www.business.att.com/content/whitepaper/Nemertes_DN0496_Network-Based_Firewall_Services_May_2009.pdf`, 2009.

[59] Bruce Schneier. The case for outsourcing security. *Computer*, 35(4):supl20–supl21, 2002.

[60] Hualong Sheng, Lingbo Wei, Chi Zhang, and Xia Zhang. Privacy-preserving cloud-based firewall for iaas-based enterprise. In *Proceedings*

of 2016 International Conference on Networking and Network Applications (NaNA 2016), pages 206–209. IEEE, 2016.

[61] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. Making middleboxes someone else's problem: network processing as a cloud service. *ACM SIGCOMM Computer Communication Review*, 42(4):13–24, 2012.

[62] Junjie Shi, Yuan Zhang, and Sheng Zhong. Privacy-preserving network functionality outsourcing. *[online] arXiv:1502.00389*, 2015.

[63] Sumeet Singh, Florin Baboescu, George Varghese, and Jia Wang. Packet classification using multidimensional cutting. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 213–224. ACM, 2003.

[64] Venkatachary Srinivasan, Subhash Suri, and George Varghese. Packet classification using tuple space search. In *ACM SIGCOMM Computer Communication Review*, volume 29, pages 135–146. ACM, 1999.

[65] Pyda Srisuresh and Kjeld Egevang. Traditional ip network address translator (traditional nat), 2001.

[66] Nancy Sumrall and Manny Novoa. Trusted computing group (tcg) and the tpm 1.2 specification. In *Intel Developer Forum*, volume 32, 2003.

[67] Cong Wang, Xingliang Yuan, Yong Cui, and Kui Ren. Toward secure outsourced middlebox services: Practices, challenges, and beyond. *IEEE Network*, 32(1):166–171, 2018.

[68] Lingbo Wei, Chi Zhang, Yanmin Gong, Yuguang Fang, and Kefei Chen. A firewall of two clouds: Preserving outsourced firewall policy confidentiality with heterogeneity. In *Proceedings of 2016 IEEE Global Communications Conference (GLOBECOM 2016)*, pages 1–6. IEEE, 2016.

[69] Martin Whitworth. Outsourced security–the benefits and risks. *Network Security*, 2005(10):16–19, 2005.

[70] Avishai Wool. A quantitative study of firewall configuration errors. *Computer*, 37(6):62–67, 2004.

[71] Andrew C Yao. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pages 160–164. IEEE, 1982.

[72] Sorrachai Yingchareonthawornchai, James Daly, Alex X Liu, and Eric Torng. A sorted partitioning approach to high-speed and fast-update openflow classification. In *Network Protocols (ICNP), 2016 IEEE 24th International Conference on*, pages 1–10. IEEE, 2016.

[73] Xixun Yu, Zheng Yan, and Athanasios V Vasilakos. A survey of verifiable computation. *Mobile Networks and Applications*, 22(3):438–453, 2017.

[74] Lihua Yuan, Hao Chen, Jianning Mai, Chen-Nee Chuah, Zhendong Su, and Prasant Mohapatra. Fireman: A toolkit for firewall modeling and analysis. In *Security and Privacy, 2006 IEEE Symposium on*, pages 15–pp. IEEE, 2006.

[75] Xingliang Yuan, Huayi Duan, and Cong Wang. Bringing execution assurances of pattern matching in outsourced middleboxes. In *Proceedings of the 24th IEEE International Conference on Network Protocols (ICNP 2016)*, pages 1–10. IEEE, 2016.

[76] Xingliang Yuan, Huayi Duan, and Cong Wang. Assuring string pattern matching in outsourced middleboxes. *IEEE/ACM Transactions on Networking (TON)*, 26(3):1362–1375, 2018.

[77] Shuyuan Zhang, Abdulrahman Mahmoud, Sharad Malik, and Sanjai Narain. Verification and synthesis of firewalls using SAT and QBF. In *Proceedings of the 20th IEEE International Conference on Network Protocols (ICNP)*, pages 1–6. IEEE, 2012.

Vita

Rezwana Reaz is from Dhaka, Bangladesh. She completed her high school education in Dhaka. She started her undergraduate studies at Bangladesh University of Engineering and Technology (BUET) in January 2006 and earned B.Sc. in Computer Science and Engineering in February 2011. She enrolled in the masters program in the same university in April 2011 and graduated with a masters degree in Computer Science and Engineering in August 2013. During her masters, she also served as an ad-hoc Lecturer in the Department of Computer Science and Engineering in BUET. Her undergraduate and masters theses addressed interesting problems in the field of Computational Biology. She enrolled in the PhD program in the Department of Computer Science at the University of Texas at Austin in Fall 2013. During her PhD, she has worked under the supervision of Prof. Mohamed G. Gouda in network protocols and firewall security. Her research interest lies in the areas of formal methods, network firewalls, security protocols, and computational biology.

Email address: rimpi0505042@gmail.com

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.