



Master of Science in Computer Engineering

Master Degree Thesis

Optimizations and Analysis in Firewall Anomaly Resolution

Supervisors

prof. Riccardo Sisto

prof. Fulvio Valenza

dott. Lucia Seno

dott. Daniele Bringhenti

Candidate

Ilaria SCHIO

ACADEMIC YEAR 2021-2022

Acknowledgements

Pause today and notice something
you have worked hard on and
recognize yourself for it.
Acknowledge your effort.

Kristin Armstrong

This project would not have been possible without the help and assistance of professor Fulvio Valenza and engineer Lucia Seno. I am also extremely grateful to my friends and family who supported me through all these years.

Thank you.

Contents

List of Figures	7
List of Tables	9
Listings	9
1 Introduction	11
1.1 Thesis Introduction	11
1.2 Thesis Description	12
2 Firewalls	14
2.1 Firewall: General	14
2.2 Access Control Criteria	16
2.3 Types of Firewalls	16
2.3.1 Packet Filtering Firewalls	17
2.3.2 Circuit-Level Gateways	18
2.3.3 Application-Level Gateways (Proxy Firewalls)	19
2.3.4 Stateful Inspection Firewalls	20
2.3.5 Next-Generation Firewalls (NGFW)	21
2.4 Final	22
3 Conflict Analysis	23
3.1 Rule Relations	23
3.2 Anomalies	25
3.2.1 Sub-Optimization Anomalies	25
3.2.2 Conflict Anomalies	27
3.3 Other Solutions	27
3.3.1 Al-Shaer, Hamed, Boutaba, and Hasan	28
3.3.2 Hu and Ahn	29
3.3.3 Cheminod, Durante, Seno and Valenzano	33

4	Thesis Objective	35
5	Problem Definition and Methodology	37
5.1	Which First: Simplify or Filter	37
5.2	Conflict Types or Configurations	38
5.3	Priority or not Priority	39
5.4	Order	39
5.5	Considerations	40
5.6	Analysis and Considerations on Rule Relationships	42
6	Case Study Analysis	50
7	The Proposed Approach	99
7.1	Find the Best Rule	99
7.2	How to Query the Administrator	102
7.3	Algorithm	110
8	Implementation and Validation	112
8.1	Setup and Environment	112
8.2	Main Classes and Data Objects	115
8.3	Important Functions	117
8.4	Example with Test Case and Validation	125
9	Conclusion	131
	Bibliography	133
	Appendices	136
	Appendix A	137
a	Initial List	137
b	Final List	142
	Appendix B	143
a	Method intersect	143
b	Methods generate and helper	144
c	Method findMinPriorityRule	146

d	Method removeRuleCouple	146
e	Method query	147
f	Method getRuleByID	148
g	Method invertPriority	149
h	Method findLastRule	149
i	Method algorithm	150

List of Figures

2.1	Example of a Firewall	15
2.2	Packet Filtering Firewall	17
2.3	Circuit-Level Gateway	18
2.4	Application-Level Gateway (Proxy Firewall)	19
2.5	Stateful Inspection Firewall	20
2.6	Next-Generation Firewall (NGFW)	21
3.1	Example of Ordered List of Rules	24
3.2	Policy Tree for the Firewall in Figure 3.1	28
3.3	Architecture of FAME	30
3.4	Policy anomaly management framework of FAME	31
3.5	Example of Firewall and Firewall Sequences	33
5.1	Mirror cases of n.19	46
5.2	Mirror cases of n.95	47
5.3	Fifth-Level Considerations Cases	49
6.1	Case n.7	50
6.2	Case n.19	53
6.3	Case n.27	56
6.4	Case n.32	58
6.5	Case n.36	60
6.6	Case n.39	62
6.7	Case n.42	64
6.8	Case n.44	66
6.9	Case n.47	69
6.10	Case n.49	71

6.11	Case n.57	73
6.12	Case n.59	75
6.13	Case n.69	77
6.14	Case n.82	79
6.15	Case n.84	81
6.16	Case n.85	83
6.17	Case n.89	85
6.18	Case n.92	87
6.19	Case n.93	90
6.20	Case n.94	92
6.21	Case n.95	95
6.22	Case n.97	97
7.1	Case with 4 Rules - Configuration	104
7.2	Case with 4 Rules - Graph	105
8.1	Eclipse Logo	112
8.2	Welcome Screen of Eclipse 4.12	113
8.3	XML Schema	114
8.4	Graphical Representation of the Clusters	126
8.5	Solution of Cluster 1	126

List of Tables

8.1	Example of rules inside a Firewall	125
8.2	RuleCouples generated in Cluster1	127
8.3	RuleCouples with Relation Values in Cluster1	127
8.4	Rules in the Cluster1 After Querying R1	129
8.5	RuleCouples in the Cluster1 After Querying R1	129
8.6	RuleCouples in the Cluster1 After Querying R5	130
8.7	RuleCouples in the Cluster1 After Querying R6	130

Listings

8.1	checkValueRelationFunction method	118
8.2	findBestRuleToQuery method	119
8.3	resolveConflicts method	122
1	intersect method	143
2	generation method	144
3	helper method	145
4	findMinPriorityRule method	146
5	removeRuleCouple method	147
6	query method	147
7	getRuleByID method	148
8	invertPriority method	149
9	findLastRule method	149
10	algorithm method	150

Chapter 1

Introduction

1.1 Thesis Introduction

Network security policies can be seen as a set of rules that describe the company's security controls. The purpose is to delineate the rules for allowing or denying access to the network and also how policies are applied. Its intent is to block malicious users from entering in the internal network and at the same time it tries to reduce the risk of malevolent users from within. This is in the form of a very long and complex document that has to be passed through a committee in order to be put in place.

The security policy should also determine how policies are enforced by imposing a hierarchy of access permissions, following the cybersecurity principle of 'need to know': a user should only have access to some information that they are required to know.

The network security policies are then implemented inside the company by properly configure and manage the firewall and the network security controls. This means that policies and rules are inserted inside the firewall in a certain order by the network administrator and this implies a certain order of priority between the rules.

The problem is that rules and policies might change over time: some rules may have been inserted but then forgotten, others may have become obsolete, or maybe because of some external changes inside the company the policies also need to be updated. In our study we will only consider the case in which the administrator can add a new rule or policy over time, but they cannot modify or re-write an already existing one inside the list of rules. Also we assume that the administrator never make mistakes when they write rules, but they only insert them in the wrong order.

Everytime we need to enforce the security policies inside the company we look up to this list, but if there is some confusion in what is the action is to take, then we need to query the administrator that will solve our doubts.

In literature there are some works that address the argument of security policies, but not with the optimization point of view of the workload of the administrator. For example, Al-Shaer et al. developed a tool that implement a firewall anomaly discovery algorithm and the distributed firewall policy editor: it just detects and report the anomlies but it does not solve them; Hu and Ahn's approach tries to solve the anomalies but they use a greedy algorithm, so it is only sub-optimal.

The purpose of this thesis is to find an algorithm that allows us to query the least number of times the administrator while also solving all the problems that are present inside the list of rules. In order to define the algorithm we had to first study how to approach the problem from a theoretical point of view, therefore an extensive analysis of how rules interact with each other in general.

Then, we figured out a semantic to query the administrator in an efficient way, while solving the conflicts, which is asking for the correctness of a specific rule. This query can be translated into priority constraints that based on the response from the administrator can be accepted and forced or have to be negated. From these analysis we were able to design an algorithm that complied with our goals.

Before moving to the next cycle we perform some checks to verify if inside the cluster are still present some conflicts, if we reach to a solution we can move on to the next cluster; if instead it is not possible, we keep query the administrator. The algorithm can be optimized with the help of a SAT solver, that would minimize in some cases the number of queries required to reach the solution for some clusters.

The implementation that we propose in this project is done in Java language, while we leave for future works the solution with the help of a SAT solver. Nevertheless, the algorithm is able to first determine clusters of rules inside a firewall that interact with each other, detect the potential conflicts within them and solve the conflicts in an efficient way.

1.2 Thesis Description

Excluding this first chapter, which aims to give an overview of the overall objective, the thesis will follow the structure described below:

- **Chapter 2 - Firewalls:** in this chapter we give a background knowledge of what is a firewall and which are the most common implementations of it nowadays.
- **Chapter 3 - Conflict Analysis:** in this section it will be presented which are the definitions that will be used throughout the whole thesis about relations and conflicts, also it will show some solutions for conflict analysis taken from literature.

- **Chapter 4 - Thesis Objective** : here, we outline in better details the thesis purpose explaining the sub-problems that we will encounter.
- **Chapter 5 - Problem Definition and Methodology**: in this chapter we will describe the different approaches and methods that we considered at the beginning and the winning resolution that we used to solve the problem. Then given a simple problem of three rules interacting with each other, we analyse all the possible outcomes. The analysis is divided in two: in this first phase we perform a ‘filtering’ in which we remove the non-interesting cases.
- **Chapter 6 - Case Study Analysis**: following the previous chapter analysis, here we present the second part of the analysis, in which we study the remaining cases one by one.
- **Chapter 7 - The Proposed Approach**: after the analysis we sum up the considerations in order to extract useful information.
- **Chapter 8 - Implementation and Validation**: here we describe the most important functions that are used inside the implementation of the code and we show some meaningful results.
- **Chapter 9 - Conclusion**: finally we reach to a conclusion to our project, we discuss the overall solution and what can still be achieved in this field.
- **Appendix A**: we display the initial and final lists that are used in the rules’ relationship analysis.
- **Appendix B**: in this part we present the additional code for the useful functions that are used in the algorithm.

Chapter 2

Firewalls

2.1 Firewall: General

Firewalls are a cybersecurity devices that are used to filter the incoming and outgoing traffic. They are network security controls that regulate the traversing of packets by the definition of:

- an ordered set of rules, in which each rule R is composed by a set of conditions C and one action a : $R = (C, a)$
- a default action, it is the action to apply if the packet does not meet all the conditions of any rule
- a resolution strategy, it describes which rule's action should be applied if more than one rule matches the packet.

The firewall actions can be ALLOW, it forwards the packet, or DENY, it discards the packet. The packet is allowed or denied by a specific rule if the packet header matches all the conditions of this rule.

There are different types of resolution strategy:

- First Matching Rule (FMR) = selects the first applicable rule in the ordered list.
- Allow Takes Precedence (ATP) = when more rules are activated and they have contradicting actions, you enforce the ALLOW rule over the DENY one.
- Deny Takes Precedence (DTP) = when more rules are activated and they have contradicting actions, you enforce the DENY rule over the ALLOW one.
- Most Specific Takes Precedence (MSTP) = when there are two conflicting rules that are activated, you have to select the most specific rule.

- Least Specific Takes Precedence (LSTP) = when there are two conflicting rules that are activated, you have to select the least specific rule.

For instance, let's imagine a firewall with the following set of rules:

Protocol	Source		Destination		Action
	Address	Port	Address	Port	
1: tcp,	140.192.37.20,	any,	*.*.*.*,	80,	deny
2: tcp,	140.192.37.*,	any,	*.*.*.*,	80,	accept
3: tcp,	*.*.*.*,	any,	161.120.33.40,	80,	accept
4: tcp,	140.192.37.*,	any,	161.120.33.40,	80,	deny
5: tcp,	140.192.37.30,	any,	*.*.*.*,	21,	deny
6: tcp,	140.192.37.*,	any,	*.*.*.*,	21,	accept
7: tcp,	140.192.37.*,	any,	161.120.33.40,	21,	accept
8: tcp,	*.*.*.*,	any,	*.*.*.*,	any,	deny
9: udp,	140.192.37.*,	any,	161.120.33.40,	53,	accept
10: udp,	*.*.*.*,	any,	161.120.33.40,	53,	accept
11: udp,	140.192.38.*,	any,	161.120.35.*,	any,	accept
12: udp,	*.*.*.*,	any,	*.*.*.*,	any,	deny

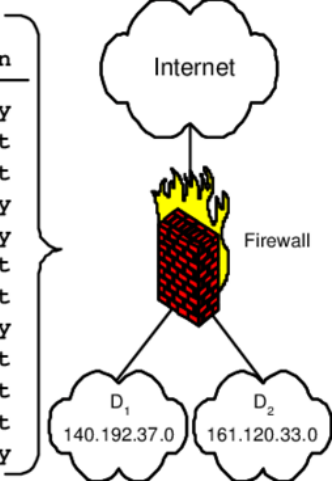


Figure 2.1: Example of a Firewall

A packet arrives to it with these characteristics:

$$P = (TCP, 140.192.37.18, any, 161.120.33.40, 80) \quad (2.1)$$

Upon its arrival, it will activate the rules: $R2$, $R3$, $R4$ and $R8$. Depending on the strategy that you implemented the rule that will be applied can be different. If you choose:

- FMR \implies then, $R2$ is selected
- ATP \implies then, $R2$ is selected
- DTP \implies then, $R4$ is selected
- MSTP \implies then, $R4$ is selected
- LSTP \implies then, $R8$ is selected

Usually the FMR strategy is the most used, but it still depends on the situations and necessities of the organization network.

2.2 Access Control Criteria

The rules inside the firewall are applied also following two possible methods:

- *whitelisting* or default-allow, in general it is a mechanism which allows some entities to access a particular service or privilege while everything else is denied. it is a list of things that are allowed while everything that is not on the list is denied.
- *blacklisting* or default-deny, in general it is a mechanism that allows the access to everything except those that are explicitly mentioned in the list.

Normally firewalls apply the blacklisting criteria since it is safer, more secure and it allows you to have a better precision on the rule's definition than the whitelisting method.

2.3 Types of Firewalls

Firewalls can be either implemented in hardware, software or in a mixed solution with both hardware and software. It is best practice to try to achieve the maximum level of protection and safety, while not losing too much of efficiency, time and money. Depending on the level of security that you need from the firewall you can have different types of firewalls. Each type has its own advantages and disadvantages, the choice is up to the organization or client based on their personal requirements and necessities.

An hardware firewall is a dedicated physical device that it inserted between a computer and a gateway. On the other hand, a software firewall is a particular program that runs on a computer or on a server, as any other software application.

Firewalls can be also divided in two main categories [10]:

- Host-based Firewalls
- Network-based Firewalls

The former type is installed on each network node, it controls each incoming and outgoing packet. It is usually a software application that is part of the operating system, it protects each host from unauthorized access and dangerous attacks.

The latter is typically a dedicated system with proprietary software installed. Network firewalls employ two or more network interface cards and work on the network level.

There are many types of network-based firewall, let's see a few of them and their characteristics.

2.3.1 Packet Filtering Firewalls

A Packet Filtering firewall is the most basic, simplest and easiest type of firewall.

This type of firewall performs a simple check of the data packets coming through the router, inspecting information in the packet header without inspecting its contents.

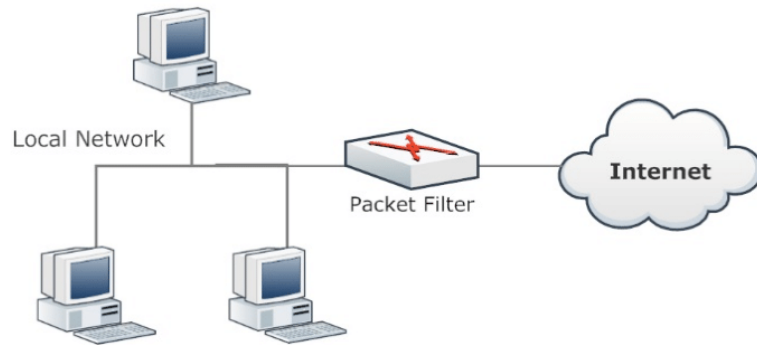


Figure 2.2: Packet Filtering Firewall

If the packet does not pass the check, then it is dropped. It is also known as a *static firewall*.

Advantages:

- it is not very resource-intensive, it does not have a big impact on system and network performance
- it is relatively simple and inexpensive
- all the traffic could be filtered by a single device (theoretically)
- it is extremely fast and efficient in scanning the traffic, since it is not looking in the content

Disadvantages:

- it is subject to IP spoofing attack, it does not check the payload
- since it is based on IP address and/or port information, it lacks of the “big picture”
- the access control list can be difficult to manage and set up
- in some cases it might not be secure enough, you may need more security

2.3.2 Circuit-Level Gateways

A Circuit-Level Gateway is a firewall that provides User Datagram Protocol (UDP) and Transmission Control Protocol (TCP) connection security and typically operate at the session-level.

It monitors the handshakes initiation messages that are exchanged across the network between the local and remote hosts to determine whether the session that is being set up is actually legitimate. The gateway is put in between the two hosts and exchanges TCP segments with the two without changing the contents.

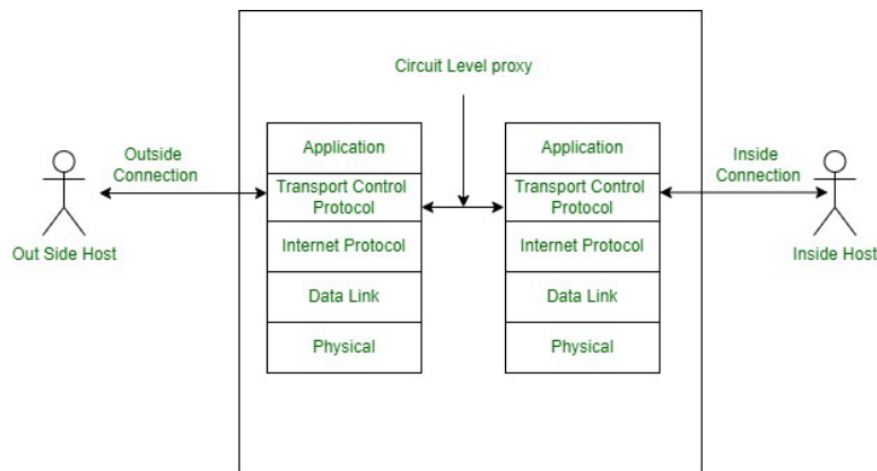


Figure 2.3: Circuit-Level Gateway

However, as we have said, it does not look into the content of the packets, but it just verifies the transmission control protocol (TCP) handshake.

This means that, if a packet is dangerous or contains a malware, but it has performed the right TCP handshake headers, it would pass right through the firewall undetected.

Advantages:

- only processes requested transactions, all other traffic is rejected
- it is easy to set up and manage
- it is fast at scanning, since it does not look inside the packet contents
- it is rather cheap and inexpensive

Disadvantages:

- it does not provide protection against data leakage from devices within the firewall

- no application layer monitoring
- it requires frequent updates to keep rules up-to-date
- it provides a higher level of security than packet filtering firewalls, but this could still not be enough for certain systems that require higher security

2.3.3 Application-Level Gateways (Proxy Firewalls)

The Application-Level Gateways can examine application layer information (i.e. HTTP, POP, DNS, etc). It operates at the application layer as an intermediate device to filter incoming traffic between two end systems, this is why they are also called *proxy firewalls*.

They act as proxy, transferring the requests from the clients pretending to be original clients on the web-server. This allows to protect the client's identity, keeping the network safe from potential attacks. Once the connection is established, it inspects data packets coming from the source.

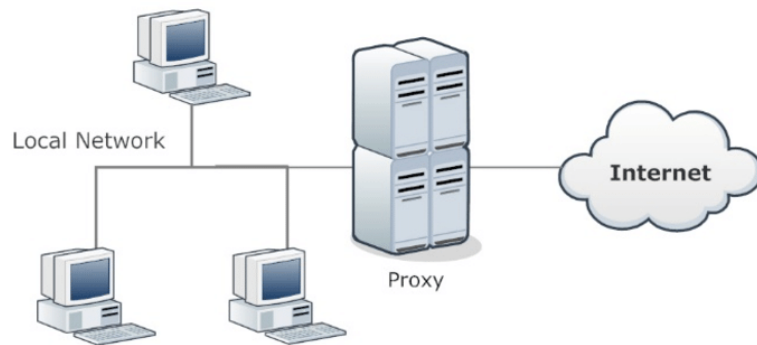


Figure 2.4: Application-Level Gateway (Proxy Firewall)

The proxy firewalls perform a check that looks at both the packet and at the TCP handshake protocol, if everything is correct then the packet is forwarded, otherwise the packet is discarded.

This type of firewalls may also perform deep-layer packet inspection, checking the actual contents of the information packet to verify that it contains no malware.

Advantages:

- it examines also at the content of the packet, not only the headers
- it provides more specific security controls
- it assures user anonymity

Disadvantages:

- they can affect network performance and can be difficult to manage
- it is more expensive than the other type of firewalls
- it can create significant slowdown and delays in the communication

2.3.4 Stateful Inspection Firewalls

Stateful Inspection Firewalls are able to inspect each packet and they can monitor and check if a packet is included or not in an established network session, for instance a TCP session.

This solution has the advantage of being more secure than previous firewall types as the packet filter, but of course this device demands greater costs and expenses from the network performance.

It only allows the communication if and only if, the session is perfectly established between two hosts, otherwise it will block the connection. This type of firewall is also called *dynamic packet filtering*.

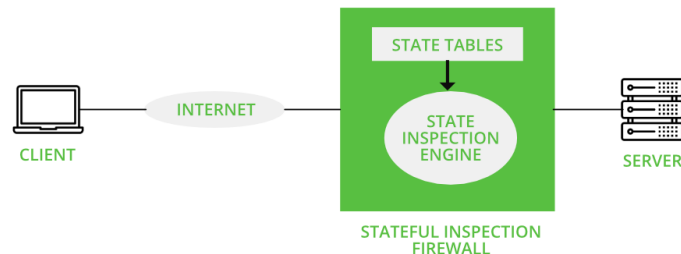


Figure 2.5: Stateful Inspection Firewall

When a user establishes a connection, this type of firewall creates a sort of database in which it stores everything called *state table*. Usually inside the state table you store for each session all connection-related information, i.e. source and destination IP addresses and port number, etc.

These types of firewalls implement more checks and are considered more secure than the stateless firewalls (the ones that do not keep track of the connection information).

Advantages:

- it keeps track of the entire session, at the same time it checks IP addresses and payloads

- it provides a greater level of control over the traffic
- it is more secure than packet filtering and circuit-level gateways
- it does not need to open a big number of ports to allow traffic in or out
- it helps to prevent some attacks for example DoS

Disadvantages:

- it slows down the transfer of legitimate packets through the network compared to other solutions
- it increases the load and puts more pressure on computing resources
- it is more expensive
- it does not provide authentication capabilities to validate traffic sources

2.3.5 Next-Generation Firewalls (NGFW)

The Next-Generation Firewalls are also called *intelligent firewalls*: they can achieve all the functions that are implemented by the previous solutions, plus some supplementary features like application awareness and control, malware filtering, integrated intrusion prevention, etc.

Unlike some of the previous types, NGFWs monitor and inspect the entire transaction of data, so the whole packet (headers and contents), and sources. Next-generation firewalls may include other technologies as well, such as intrusion prevention systems (IPSs) that work to automatically stop attacks against your network.

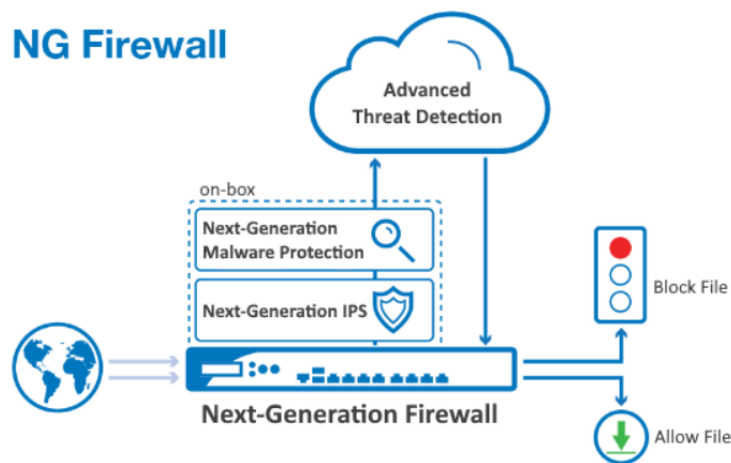


Figure 2.6: Next-Generation Firewall (NGFW)

In general, there is no single definition for this type of firewalls. An ordinary NGFW combines packet with stateful inspection and it also incorporates some variety of deep packet inspection (DPI), but also other network security systems, for instance IDS and IPS and malware filtering. NGFWs have a better performance when they are incorporated with other security systems.

Advantages:

- it combines different type of controls and check in order to provide an excellent level of filtering the traffic
- it records all traffic from the network to the application layer
- it can be automatically updated to give the up-to-date context

Disadvantages:

- since NGFW performs better when integrated with other systems, it may be very difficult and complex to manage
- it is more expensive than the other solutions

2.4 Final

However, in this project we will not need to use or to consider very high performing firewalls or very specific requirements. We will be content enough with a simple packet filter packet filter with deny as default action and the First Matching Rule (FMR) as the resolution strategy.

Therefore, this firewall can be thought as an ordered list of rules, in which each rule inside the firewall will be in the form of:

$$R_x = (IP_{source}, Port_{source}, IP_{dest}, Port_{dest}, Protocol, Priority, Action) \quad (2.2)$$

where the

$$Action = ALLOW \vee DENY. \quad (2.3)$$

This is the simplest and easiest choice that we can make in order to simplify our arguments.

Chapter 3

Conflict Analysis

In this chapter we describe the basic definitions of rules, relations, anomalies and conflicts that will be used throughout the whole thesis. The mathematical interpretations and definitions of these objects are taken from Valenza and Cheminod's work [1].

3.1 Rule Relations

In order to analyse, detect and then solve anomalies you need to have a model for specifying the relations between rules. Thus, in our model let's consider four type of relations between the condition fields:

- equivalence ($f_x = f_y$): two condition fields f_x and f_y are equivalent if they have the same value (or range of values)
- dominance ($f_x \succ f_y$): a condition field f_x dominates another one f_y if it is a generalization of the second one
- correlation ($f_x \sim f_y$): two condition fields f_x and f_y are correlated if they share some values, but neither of them dominates the other one.
- disjointness ($f_x \perp f_y$): two condition fields f_x and f_y are disjoint if they do not share any value

Given the definition of the relationships between the fields, we can advance with the correspondent relation among two condition sets, C_x and C_y . Considering two conditions C_x and C_y , only one of the following relations holds:

- equivalence: two conditions C_x and C_y are equivalent if each condition field f_x^i in C_x is equivalent to the corresponding condition element f_y^i in C_y so that they exactly match the same packets: $C_x \equiv C_y \Leftrightarrow f_x^i = f_y^i \forall i$

- dominance: a condition C_x dominates C_y ($C_x \succ C_y$) if it is a generalization of the latter. This means that it is true, if the first condition set matches all the packet matched by the second, and some more: $C_x \succ C_y \Leftrightarrow C_x \not\equiv C_y \wedge f_x^i \succeq f_y^i \forall i$
- correlation: two conditions C_x and C_y are correlated ($C_x \sim C_y$) if they match some common packets, but none of them includes (or dominates) the other one: $C_x \sim C_y \Leftrightarrow C_x \not\subseteq C_y \wedge C_x \not\supseteq C_y \wedge \forall i | f_x^i \not\subseteq f_y^i$ where $C_x \not\subseteq C_y$ stands for $C_x \not\supseteq C_y \wedge C_x \neq C_y$
- disjointness: two conditions are disjoint if they do not match any common packet: $C_x \perp C_y \Leftrightarrow \exists i | f_x^i \perp f_y^i$. Note that $C_x \not\subseteq C_y$ means that C_x and C_y are either equivalent, correlated or one dominant the other.

We also need to introduce the concept of priority of a rule, which is described with the function $\pi(r)$, with r being a rule. To be more precise, the function $\pi(r)$, returns the position of r in the ordered rule set. We place at the top of the list the rules with the highest priority. Thus, between any two rules r_x and r_y it exists either the relation $\pi(r_x) > \pi(r_y)$ or the opposite $\pi(r_x) < \pi(r_y)$.

Let's see an example to better understand it. Given this set of ordered rules, let's try to determine the relations between these rules:

	priority	IPsrc	Psrc	IPdst	Pdst	Proto	Action
r_1	1	130.162.0.1	*	130.162.0.2	80	TCP	ALLOW
r_2	2	130.162.1.3	*	130.162.2.2	*	UDP	ALLOW
r_3	3	130.162.0.1	*	130.162.0.2	80	TCP	DENY
r_4	4	130.162.1.0/24	*	130.162.2.0/24	*	*	ALLOW
r_5	5	130.162.1.0/24	*	130.162.2.0/24	80	*	ALLOW
r_6	6	130.162.3.0/24	*	130.162.0.0/24	*	*	DENY
r_7	7	130.162.3.1	*	130.162.0.1	80	TCP	ALLOW
r_8	8	130.162.1.1	*	130.162.2.1	22	TCP	ALLOW
r_9	9	130.162.1.4	*	130.162.2.5	0-1024	*	DENY
r_{10}	10	130.162.3.0/24	*	130.162.0.0/24	*	*	DENY
r_{11}	11	130.163.3.0/24	*	130.162.0.0/24	*	*	DENY
...							
default	∞	*	*	*	*	*	DENY

Figure 3.1: Example of Ordered List of Rules

- equivalence \Rightarrow (R1, R3) and (R6, R10)
- dominance \Rightarrow (R2, R4), (R2, R5), (R4, R5), (R4, R8), (R6, R7) and (R7, R10)
- correlation \Rightarrow (R1, R6), (R1, R10), (R1, R11), (R3, R6), (R3, R10), (R3, R11), (R4, R9), (R5, R8), (R5, R9), (R7, R11) and (R10, R11)

- disjointness \implies (R1, R2), (R1, R4), (R1, R5), (R1, R7), (R1, R8), (R1, R9), (R2, R3), (R2, R6), (R2, R7), (R2, R8), (R2, R9), (R2, R10), (R2, R11), (R3, R4), (R3, R5), (R3, R7), (R3, R8), (R3, R9), (R4, R6), (R4, R7), (R4, R10), (R4, R11), (R5, R6), (R5, R7), (R5, R10), (R5, R11), (R6, R8), (R6, R9), (R7, R8), (R7, R9), (R8, R9), (R8, R10), (R8, R11), (R9, R10) and (R9, R11)

3.2 Anomalies

The word *anomaly* is a sort of an “umbrella-term”, that includes errors and mistakes that may happen inside a firewall. There are various reasons why these anomalies may happen: rules become obsolete, policies change over time, etc.

Either way, you need to consider that policies rules can be defined by more than one person, therefore this can become a huge problem and a source of many anomalies.

In general you can distinguish two types of firewall policy anomaly:

- intra-policy (intra-firewall), if the anomaly is between two rules in the same firewall.
- inter-policy (inter-firewall), if the anomaly is between two rules in two different firewalls.

In this project, we will only consider the intra-firewall policy anomalies. Furthermore, in this type of anomalies you can do another distinction between: conflict anomalies and sub-optimization anomalies.

Formally, given two rules $r_x = (C_x, a_x), r_y = (C_y, a_y)$, with some relation (they are not disjoint), you have a conflict when $a_x \neq a_y$, otherwise you have a sub-optimization.

The sub-optimization anomalies can be always automatically solved with the help of a specific resolution strategy. However, the conflict anomalies require the aid of the network administrator in order to be solved, you cannot adopt an automatic resolution strategy for this type, without changing the behaviour of the security policies.

Each of these categories has their own sub-types, let's see them also with some examples.

3.2.1 Sub-Optimization Anomalies

- *Irrelevance* = A policy rule r_x is irrelevant if it does not match any packet that might arrive to the firewall. This happens when either the source nor the destination address of the rule does not match with the subnet protected by

the firewall. This type of anomaly will not be considered in further analysis since it requires external rules that state which are the subnets protected by the firewall.

- *Duplication Anomaly* = A policy rule r_x duplicates rule r_y and vice versa if they specify the same action and match the same packets:

$$\mathcal{A}_{dup}(r_x, r_y) := C_x \equiv C_y \wedge a_x = a_y \quad (3.1)$$

If you remove the rule with the lowest priority between r_x and r_y , it will not change the policy behaviour.

- *Shadow Redundancy Anomaly* = A policy rule r_y is shadowed by a rule r_x if $\pi(r_x) > \pi(r_y)$, and all packets matched by r_y are also matched by r_x , also they both specify the same action:

$$\mathcal{A}_{shadowed}(r_x, r_y) := \pi(r_x) > \pi(r_y) \wedge a_x = a_y \wedge C_x \supset C_y \wedge a_x = a_y \quad (3.2)$$

If you remove r_y (the rule with the lower priority), then you will not change the overall policy behaviour.

- *Unnecessary* = A policy rule r_x is unnecessary with respect to rule r_y when r_x and r_y specify the same action, $\pi(r_x) > \pi(r_y)$, all packets that are matched by r_x also matched by r_y and it does not exist a rule $r_z \not\supset r_x$ with a priority such that $\pi(r_x) > \pi(r_z) > \pi(r_y)$ and with opposite action:

$$\begin{aligned} \mathcal{A}_{unn}(r_x, r_y) := & \pi(r_x) > \pi(r_y) \wedge C_y \supset C_x \wedge a_x = a_y \wedge \\ & \nexists r_z | \pi(r_x) > \pi(r_z) > \pi(r_y) \wedge C_z \not\supset C_y \wedge a_z \neq a_y \end{aligned} \quad (3.3)$$

In this case, if you remove r_x , this will not change the policy behaviour.

Considering the list of rules in the Figure 3.1, then we can determine the following sub-optimization anomalies:

- duplication anomaly \implies (R6, R10), since they are equivalent and they have the same action
- shadow redundancy anomaly \implies (R4, R5), since R4 includes R5 and they share the same action and (R4, R8), since R4 includes R8 and the both have the same action
- unnecessary \implies (R2, R4) and (R2, R5), in both R2 has a higher priority than R4 and R5, all packets that match R2 also match R4 and R5, they all specify the same action, plus it does not exist an intermediate rule between these rules that have a different action.

3.2.2 Conflict Anomalies

- *Contradiction Anomaly* = Two rules r_x and r_y are in contradiction if they match the same packets but they have different actions:

$$\mathcal{A}_{contr}(r_x, r_y) := C_x \equiv C_y \wedge a_x \neq a_y \quad (3.4)$$

To solve this conflict the administrator has to choose which one has to be removed from the list.

- *Shadowing Conflict Anomaly* = A policy rule r_y is shadowed by rule r_x if $\pi(r_x) > \pi(r_y)$, and all packets matched by r_y are also matched by r_x , also the two rules have different actions:

$$\mathcal{A}_{shdwconfl}(r_x, r_y) := \pi(r_x) > \pi(r_y) \wedge C_x \succ C_y \wedge a_x \neq a_y \quad (3.5)$$

In this case, the administrator can decide to: remove r_y (the rule with lower priority) or put r_x just after r_y ('de-prioritization')

- *Correlation Anomaly* = Two policy rules are correlated when some packets that are matched by r_x are also matched by r_y but there are other packets that are either matched by r_x only or by r_y only, and the two rules have different actions:

$$\mathcal{A}_{corr}(r_x, r_y) := C_x \sim C_y \wedge a_x \neq a_y \quad (3.6)$$

Here, the network administrator has three choices: leave everything as it is, re-write r_y and r_x so that they are no longer in conflict or re-order the two rules such that the rule with higher priority is now after the lower priority one.

Considering the list of rules in the Figure 3.1, then then we can determine the following conflicts:

- contradiction anomaly \implies (R1,R3), since they are equivalent and they have different actions
- shadow conflict anomaly \implies (R6, R7), since R6 includes R7 and the have different actions and (R7, R10), since R10 includes R7 and they have different actions
- correlation anomaly \implies (R1, R6), (R1, R10), (R1, R11), (R4, R9), (R5, R9) and (R7, R11), they are all correlated and they specify different actions

3.3 Other Solutions

These are some of the solutions that some of the resarchers in this field produced in order to solve the issue about conflicts inside a network.

3.3.1 Al-Shaer, Hamed, Boutaba, and Hasan

In their paper, they proposed a software tool called ‘Firewall Policy Advisor’ (FPA) [2]. This tool implements the firewall anomaly discovery algorithms (both intra and inter) and a distributed firewall policy editor.

An *interfirewall anomaly* can occur if two firewalls on the network path specify different actions for the same traffic. They developed an algorithm that is able to find the rule relations and also to discover for any two or more rules all the possible anomalies between these firewalls: it is called *Interfirewall Anomaly Discovery Algorithm*.

Since on the network path that connects two subdomains there might be more than two firewalls, you need to run this algorithm for all the firewalls on this path in order to look for all the possible anomalies.

Once you have determined all the firewalls on the path you need to run in each one the *Intrafirewall Anomaly Discovery Algorithm*, in this way you can be sure that every single firewall is cleared from any possible intrafirewall anomaly. As we have seen before, an *intrafirewall anomaly* can occur when two rules specify different actions for the same traffic inside the same firewall.

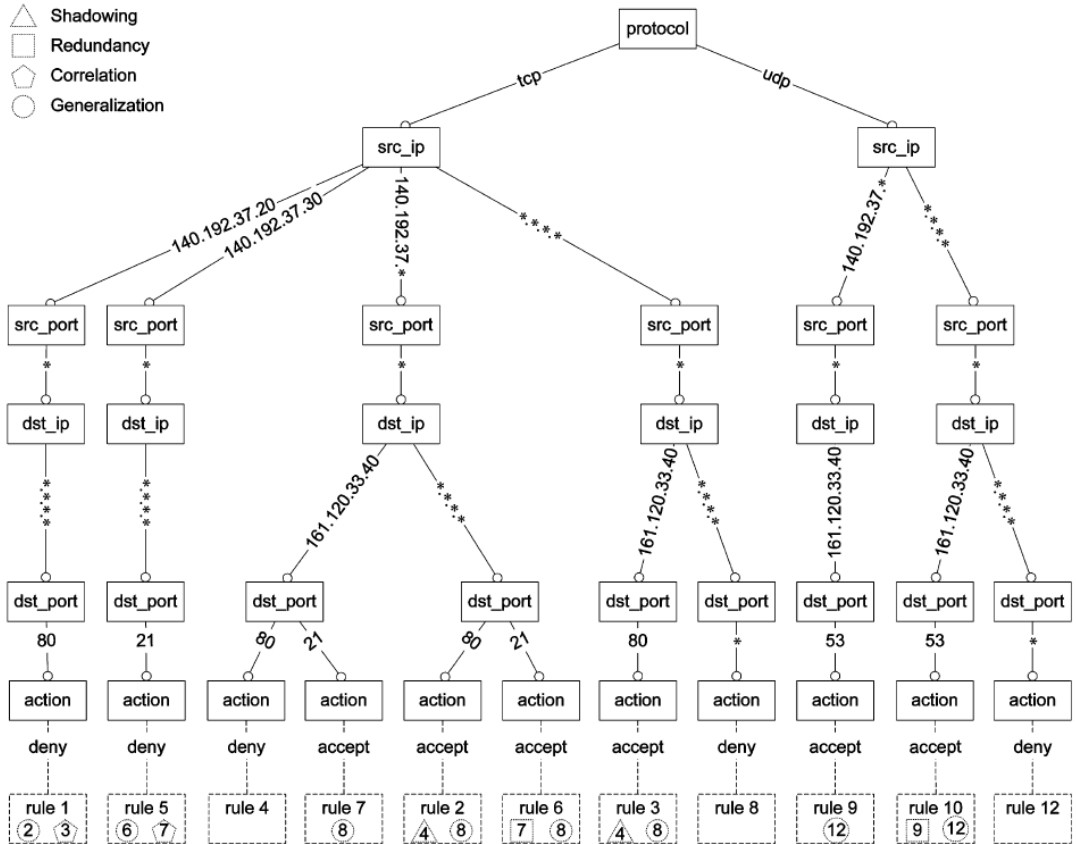


Figure 3.2: Policy Tree for the Firewall in Figure 3.1

Afterwards, starting from the most upstream firewall, you build the policy tree and you add to it the rules of all the following firewalls in the path. Once you have performed this process on the whole path, then the rules that could potentially create an anomaly are reported. Also if some rules are left unmarked, then they are declared as an irrelevant anomaly.

The algorithm continues inserting in the aggregate policy tree each rule from the successive downstream firewalls. This is done based on the field values, so that the current rule is inserted in the policy tree by matching the previously inserted rules.

If the rule is disjoint or correlated, it is inserted into a new branch. If the rule is a superset match, the rule is inserted into the branches of all the subset rules. In all the other cases, the rule is inserted in the first branch of a rule that is an exact or superset match.

When the process terminates, the anomaly is discovered depending on the relation and on the action of the currently inserted rule and the rule in the policy tree. If an anomaly is found, the affected rules are marked and the anomaly is reported.

The *policy editor* facilitate the work of the user in checking which is the correct firewall in which you should insert a new rule in order to avoid interfirewall anomalies. It also helps to define the appropriate rule order within this chosen firewall so that you can bypass intrafirewall anomalies.

3.3.2 Hu and Ahn

In their research, they developed a framework called Firewall Anomaly Management Environment (FAME) [3], which can be represented with a two level architecture.

The upper layer is the visualization layer, which displays the results of the analysis to system administrators; it is based on two visualization interfaces: policy conflict viewer and policy redundancy viewer, which are aimed to manage policy conflicts and redundancies.

Meanwhile, the lower level supports the basic functionalities addressed in this framework and the relevant resources.

To better identify policy anomalies and then to be more effective in their resolution, they decided to propose a *rule-based segmentation technique*: it uses a binary decision diagram-based data structure to describe the rules, to perform many set of operations and also to convert a list of rules into a set of disjoint network packet spaces.

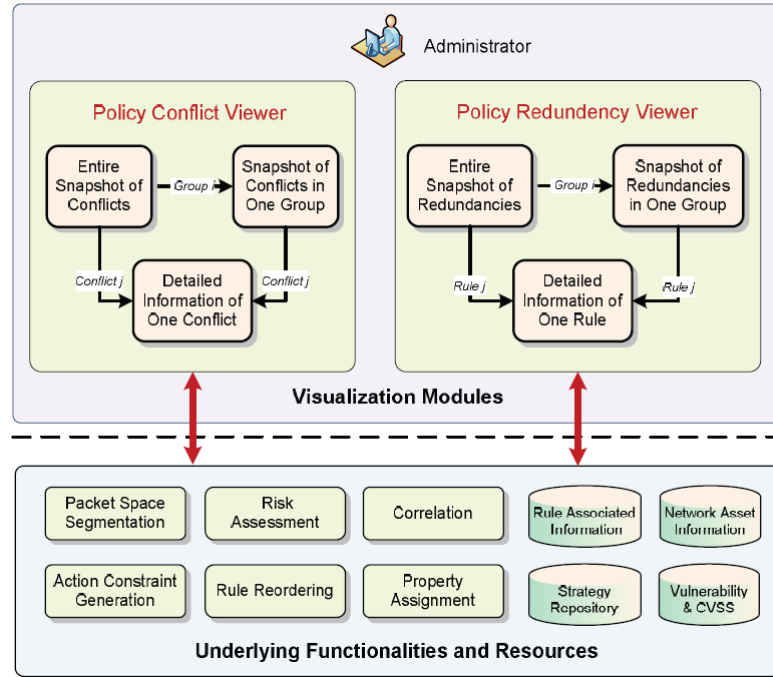


Figure 3.3: Architecture of FAME

In this way, you partition the whole space into a set of pair-wise disjoint segments, then you can classify the policy segments in: non-overlapping and overlapping segment. The overlapping segment can be additionally divided into conflicting-overlapping and non-conflicting-overlapping.

A *non-overlapping* segment is associated with one unique rule. An *overlapping* segment is associated with a set of rules, which might conflict with each other (*conflicting overlapping*) or have the same action (*non-conflicting overlapping*).

When you consider a list of rules that are interacting, it may happen that:

- one overlapping segment may be associated with several rules
- one rule may overlap with multiple other rules and can be involved in a couple overlapping segments

In the *conflict detection and resolution functionality*, the first thing to do is to identify the conflicting segments, each one of them is linked with a policy conflict and a list of conflicting rules. You also identify the correlation relationships between the conflicting segments and you derive the conflict correlation groups.

Then, for each conflicting segment you generate an action constraint by examining the characteristics of the specific conflicting segment using a strategy-based method. The last step uses a reordering algorithm to discover a near-optimal conflict resolution for policy conflicts. The algorithm that is used is a combination of a permutation and a greedy algorithm.

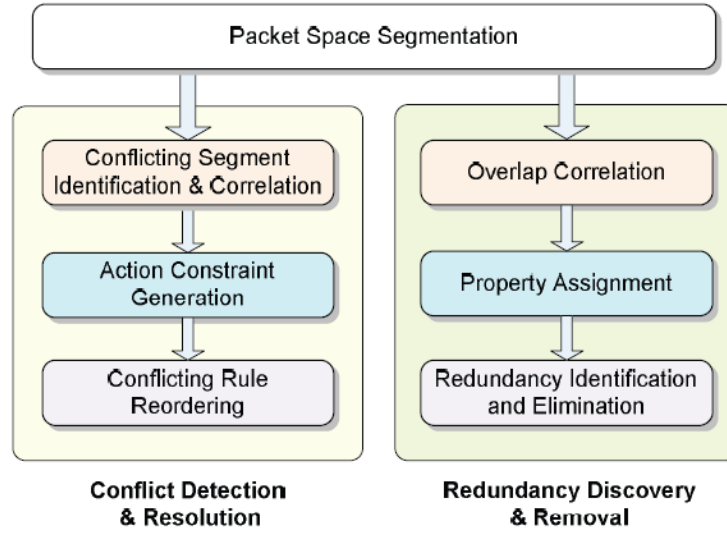


Figure 3.4: Policy anomaly management framework of FAME

In the *redundancy discovery and removal functionality*, you have to identify the segment correlation groups first, next for each rule's subspace you implement the process of property assignment. Finally, you identify and remove the redundant rules.

In order to resolve policy anomalies in an efficient way you need to identify the dependency relationships between packet segments: this is because one rule may be associated in various policy anomalies, thus isolating anomalies and solving them might cause abrupt behaviours. Therefore, you need to introduce the concept of *correlation group*, which is a set of segments with a dependency relationship.

Generating the correlation groups for anomaly analysis is convenient because the anomalies can be examined within each group independently, since all correlation groups are independent between each other.

The generation of action constraints for conflict segments relies on a strategy-based conflict resolution method. The process includes automated and manual strategy selections.

When the conflicts are found and the conflict correlation groups are identified, you have to perform the risk assessment for conflicts, which is done using both manual and automated strategies.

Therefore, they reached the conclusion to employ a greedy algorithm to try to solve the conflicts that contain a larger number of correlated conflicting rules. The only issue with this is that since it is a greedy algorithm, it is sub-optimal and not optimal, it makes the locally optimal choice, assuming that it will bring to the global optimum.

The algorithm proceeds like this:

- calculate a resolving score for each conflicting rule, individually
- select the rule with the highest resolving score, in order to solve the conflicts
- identify a position range with the best conflict resolution for this rule
- move the selected rule to the new position (local optimal)

Repeat this cycle till all the rules in the correlation group are processed. The final order of the correlated rules is a possible solution.

A critical part of this algorithm is the computation of the resolving score for the conflicting rule, every time you move a rule you might change the conflicts, thus you need to recompute the score.

Since you might want to increase the efficiency of this algorithm, they proposed a combination of permutation and greedy algorithm: they introduce a threshold N , if the number of conflicting rules is less than this threshold, then the permutation algorithm is used, if not you use the greedy algorithm.

During *redundancy elimination* part, every rule subspace that is marked by a policy segment is assigned with a property. They defined 4 property values:

- Removable = a rule subspace is removable. If you remove it, it would not make any effect on the original packet space.
- Strong Irremovable = a rule subspace cannot be removed since the action of the corresponding policy segment can be determined only by this rule.
- Weak Irremovable = if a rule inside the same subspace has a strong irremovable property.
- Correlated = if the action of the segment can be determined by any of the rule inside the subspace.

FAME offers two policy views to display the outputs of the analysis. Each viewer then provides two interfaces: one to show an entire snapshot of all the anomalies, the other to show a partial snapshot of the anomalies that are contained inside a specific correlation group.

3.3.3 Cheminod, Durante, Seno and Valenzano

They considered the problem of an overloaded firewall and how to reduce the cardinality of this firewall, while balancing the workload to other firewalls within the protected network. Thus, they presented a technique to split and migrate the security policies on sequential firewalls, whilst not modifying the overall security policy [4].

They describe a single firewall fw as:

$$fw = (r_1, r_2, \dots, r_{|fw|}) \quad (3.7)$$

where r_1, r_2, \dots are the rules inside the firewall and $|fw|$ is the cardinality of the firewall.

If at least one packet matches at least one rule in fw , then the firewall fw is *complete*. Every firewall defines a *filtering function*, which associate a packet if it is forwarded in the same packet space or in the null space if it is instead discarded.

A firewall sequence is a list of firewalls:

$$fws = (fw_1, fw_2, \dots, fw_{|fws|}) \quad (3.8)$$

Each firewall can be seen as a firewall sequence made of one single element. Also a firewall sequence defines a filtering function: which is the inverted order composition of the filtering functions associated to the individual firewalls that form the sequence.

The trivial firewall is a firewall that does not block any traffic:

$$fw_{trivial} = (\langle (*, *, \dots, *), ALLOW \rangle) \quad (3.9)$$

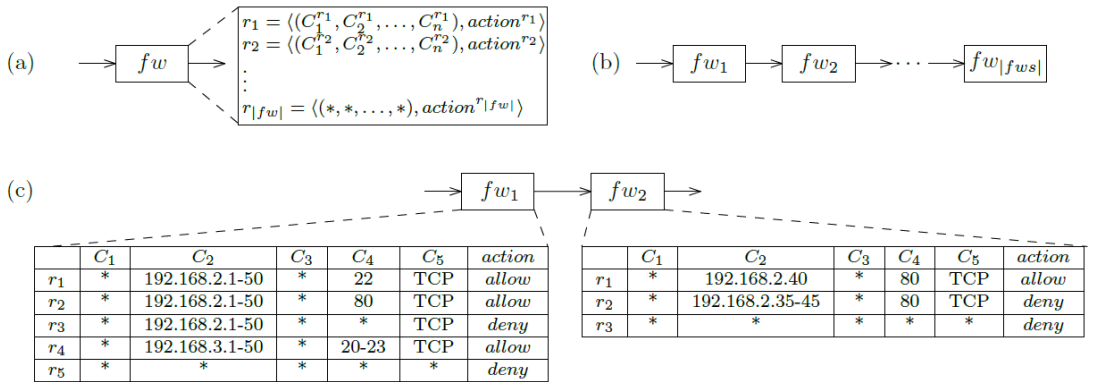


Figure 3.5: Example of Firewall and Firewall Sequences

Two firewall sequences are said to be equivalent if for every packet in the packet space the filtering functions of the firewalls return the same result.

Therefore, the problem can be expressed as given a firewall sequence made of two firewall, $fw_s = (fw_1, fw_2)$, the issue is how to compute a firewall $\overline{fw_2}$, so that the sequence $\overline{fw_s} = (\overline{fw_1}, \overline{fw_2})$ satisfies the property:

$$fw_s = \overline{fw_s} \quad (3.10)$$

where $\overline{fw_1}$ is the trivial firewall.

Their solution aims to delegate to fw_2 the workload of security policies that was originally done by fw_1 , this is done in order to have a better equilibrium of resources between the free devices.

This problem only consider the *downstream policy migration*, in which the overloaded firewall is the one at the beginning of the traffic path.

In order to solve the problem with the technique and algorithm that was proposed in [4] as a solution, they made an observation: if a packet has a matching rule in fw_1 that is a deny rule, independently of its matching rule in fw_2 , this packet is dropped by fw_s and, for the property to hold, should be also dropped by $\overline{fw_s}$.

So, the idea is to compute a set of deny rules, which eliminates all and only packets originally eliminated by fw_1 . At this point, you can move these rules before those in fw_2 to obtain $\overline{fw_2}$.

The remaining issue how to compute the set of deny rules: it is expressed as a union of sets, one for each deny rule, which is the result from sequences of set subtractions, where the result of a subtraction is the minuend of the next one.

Nonetheless to improve the performance of this algorithm you might prefer to split the security policy of fw_1 before defining the deny rules, in this way you minimize the number of rules that you have to add to fw_2 .

Chapter 4

Thesis Objective

This chapter outline the main objective of this thesis work and the sub-problems that we encountered while carrying out the thesis.

As we have noted in the Chapter 2, firewalls have been designed to monitor and control the incoming and outgoing network traffic based on some predefined rules. They are a barricade or a hurdle in order to stop malicious traffic to reach the trusted network.

However, these predefined rules can be wrong, or just inserted in the wrong order or even outdated. If this happens malevolent traffic can enter or users can access malicious sites for example, and this is not preferable in a secure network.

The main problem is rather easy to explain:

When there are too many anomalies inside the firewall, how can we ease the workload of the administrator, in terms of number of queries that we have to perform?

From this premise, many other small problems arise that we need to take care of.

First of all, as we have seen in the previous chapters, the term *anomaly* is umbrella-term for errors or mistakes inside a firewall: to be more specific a sub-optimization anomaly can be automatically solved by applying a precise strategy method, instead a conflict anomaly requires the help of the network administrator to be solved.

Therefore one issue to address is whether or not is advantageous to solve some conflicts before others, or better to say if there is a distinction in terms of performance between solving first conflicts or sub-optimization anomalies.

Hence, it is required a detailed anaysis on how the different system configurations might change or not if different types of conflicts are solved in a different

order. This is done taking into an account a simple problem of few rules interacting with each other and then considering all the possible values of relationship between these rules, how they can generate conflicts and which types.

Then, we need to focus on another obstacle: the interaction with the administrator. Thus, we need to define a syntax and a semantic to query the administrator in such a way that they are always able to give us an answer which can be interpreted univocally. There might be some types of questions that the administrator is not able to answer, so they return an ambiguous response (i.e. “Do not know.”), we cannot afford to receive these answers since we cannot extract any information.

From all of these studies and analysis we extract some suggestions and considerations that are useful to design the algorithm and the methods that we need.

Finally, we generate a possible resolution algorithm that is able to achieve what we wanted: solving conflicts inside a firewall in an optimized way, with the least number of queries to the administrator, in order to not overwork the administrator workload.

As said before, in this project we designed a theoretical algorithm that employs the help of a SAT solver to enforce constraints and exit beforehand from the querying cycle. However, in the practical implementation of the algorithm, we developed it using only the Java language due to limited time. In the latter design we accept some limitations because of the lack of intelligence of the algorithm, it is missing its “brain”, the SAT solver, that would allow to exit the loop beforehand.

Nevertheless, both the algorithms are able to first determine clusters of rules inside a firewall that interact with each other, detect the potential conflicts within them and solve the conflicts in an efficient way.

Chapter 5

Problem Definition and Methodology

As we have seen in the previous chapters, there are some solutions in literature that try to solve or at least to reduce the conflicts.

Al-Shaer's solution [2] allows you to discover anomalies inside firewalls but in the end it does not solve conflicts, it just shows them where they are inside the policy tree.

Hu's solution [3] instead proposes a way to solve conflicts but it first divides the regions in disjoint sub-spaces and then applies a greedy recursive algorithm.

And finally in the last solution that we presented, Cheminod's one [4] you are moving the problem from one firewall to another.

All of these methods are great, but still not quite what we are looking for. We need to step further into these concepts in order to define our final algorithm. Let's consider the sub-problems, explaining all their possible approaches and try to solve them one by one.

5.1 Which First: Simplify or Filter

With the term *simplify*, we mean rewriting the rules such that the IP address intervals and the ports intervals of the rules are either disjoint or equal between each other. Hence, it may happen that you have to add new rules for which it is correct just one single action (be it ALLOW or DENY) without a doubt.

In case *First Simplify Then Filter*: we perform a simplification without a previously classification or analysis of the rules, this may overwork the admin instead of helping them, since you are giving them a huge new number of rules that now they have to process. Also you need to consider that some of the original rules

may not need a rewriting operation, so you may perform a lot of work that takes time for nothing. This is the basic concept behind the *Atomic Flows* used in Hu's solution [3], the intervals are so small that the rules are either equivalent or disjoint.

This method is not encouraged since it is very complex to determine atomic flows, also the rules inside the firewall increases by a lot and if the firewall is a linear one, it is not convenient.

Alternately, you may decide to *First Filter Then Simplify*: which means that you perform some kind of checks before modifying the rules. A possible filter that can be implemented is "sub-optimized or not". The reasoning is rather simple: sub-optimization anomalies do not require the intervention of the administrator, therefore if you remove those anomalies earlier in the analysis they will not impact the number of calls to the administrator.

So, let's keep in mind that removing early on the sub-optimization anomalies can be an advantage.

After this, the simplification of the rules is still too much work, therefore we should abandon the idea of rewriting rules and consider administrators that are able to remove and in the worst case reorder rules, but never rewrite rules and adding new ones.

This means that now, the correlation conflict can be solved either by re-ordering the rules, in case the winning rule had lower priority, or leaving everything as it was, in case the winning rule had higher priority.

5.2 Conflict Types or Configurations

You may decide to solve the conflicts by types or by groups (or configurations) of rules.

By *types of conflicts*, it means that you solve them based on their relationship value: so i.e. all of type A, then all of type B, etc., this implies that there is an order of precedence between types of conflicts: that one type of conflict is more important (and therefore it has to be solved first) than another.

By *configuration*, it means that you solve all the conflicts that there are inside a group of rules that interact with each other in some way.

In their paper [1], they suggest a possible algorithm to solve this problem in a semi-automatic way, that follows the first method. It can be summed up in these few steps:

- Analyse the rules (divide them into lists)
- Cancel the irrelevance ones
- Cancel the duplicate ones
- Cancel the shadow redundancy one
- Ask the admin for the contradiction ones
- Ask the admin for the shadow conflict ones
- Modify the correlation ones (eventually)
- Re-start from the beginning
- Cancel the unnecessary ones

The first few steps are used to remove the sub-optimization anomalies, then you query the administrator about the conflicts by type.

In the other method, you first consider groups of rules that interact with each other, let's call them clusters, and then for each cluster you solve all the conflicts that are present inside by querying appropriately the administrator.

5.3 Priority or not Priority

The concept of *priority* is very important and it may have different connotations depending the resolution method that you decide to use.

The *rule's priority* allows you to decide and to understand whether to keep or not a certain rule when placed in a conflict, based on the type of conflict the solution to may differ. Pay attention that if you use a resolution by configuration (or by groups), the rule's priority may not coincide with the order in which you query the administrator (the first rule that you query may not be the rule with the highest priority). In this case you need to find a way to decide the order on which to query the rules.

The *conflict's priority* is instead used to decide and to understand which conflict has to be considered first inside the same type of conflicts. This concept is used in a resolution by type of conflicts.

5.4 Order

In case we consider a solution by type of conflicts, order means the *order of conflict solution*: find the order of precedence between the different type of conflicts. This require a theoretical analysis between all the possible types of conflicts.

- **CONTRADICTION VS CORRELATION:** the contradiction is always solved with the removal of one of the two rules that are in conflict, instead the correlation is solved either re-ordering the rules or leaving everything as it is. This means that it is convenient to always first solve the contradictions because in the best case scenario you can remove the rule that it is also part of the correlation conflict, thus you do not have to solve the correlation.
- **CONTRADICTION VS SHADOW CONFLICT:** the contradiction is always solved with the removal of one of the two rules that are in conflict, instead the shadow conflict is solved either re-ordering the rules or removing one of the rules. This means that it is convenient to always first solve the contradictions because in the best case scenario you can remove the rule that it is also part of the shadow conflict, thus you do not have to solve the shadow conflict. Of course there might be some cases in which if you solve the shadow conflict first and you remove the rule that is also part of the contradiction, therefore you do not have to solve the contradiction, but it is less likely.
- **CORRELATION VS SHADOW CONFLICT:** the shadow conflict is solved either with re-ordering the rules or with the removal of one of the two rules that are in conflict, instead the correlation is solved either re-ordering the rules or leaving everything as it is. This means that it is convenient to always first solve the shadow conflicts because in the best case scenario you can remove the rule that it is also part of the correlation conflict, thus you do not have to solve the correlation.

So the final order in which you should solve the conflicts is: contradictions, then shadow conflicts and finally correlations. This explains the order that is used in Valenza's algorithm [1].

In case we consider a solution by configuration, order means *order of querying rules*: find the order in which you have to query the rules, so that it minimizes the number of calls to the administrator. This requires an analysis on all the possible cases the rules can interact with each other (see Chapter 6), and from this we can extract some suggestions on how the rules should be ordered.

5.5 Considerations

In order to insert the rules in the firewall it is required human interaction with the system. This means that one could insert the sub-optimized (duplicated, irrelevant, unnecessary) rules in some specific positions like either at the beginning or at the end of the list, i.e you place a duplicated rule right after the original one. Based on this assumption, in order to optimize the rules' analysis, you might decide to read the list either from the top or bottom.

The problem is that this is a mere assumption of the typical human and you always fall in the best case or in the worst case scenario, you are not able to get

the average case and draw some conclusions.

Also we need to change the definition of Shadow Redundancy Anomaly, if we use the Valenza's definition, then it's not possible to remove the sub-optimization properly. Thus, we introduce a new definition: a policy rule r_y is shadowed by a rule r_x if $\pi(r_x) > \pi(r_y)$, all packets matched by r_y are also matched by r_x , also they both specify the same action and it does not exist a rule r_z such that $\pi(r_x) > \pi(r_z) > \pi(r_y)$, that has an opposite action and C_z is not disjoint:

$$\begin{aligned} \mathcal{A}_{shadowed}(r_x, r_y) &:= \pi(r_x) > \pi(r_y) \wedge a_x = a_y \wedge \\ \nexists r_z | \pi(r_x) > \pi(r_z) > \pi(r_y) \wedge C_z \not\subset C_y \wedge a_z \neq a_y \end{aligned} \quad (5.1)$$

In this way, the removal of r_y is possible without changing the overall policy behaviour and now we actually removed the suboptimization.

As we said previously, in this work we decided to consider an administrator that only removes or re-orders rules, but they will not re-write them or add new ones in replacement. This means that we will not use neither the *First Simplify Then Filter* nor the *First Filter Then Simplify*. But we will perform some kind of filtering operation in order to remove the sub-optimization anomalies.

Then we decided to focus on a configuration solution for few reasons: first and foremost it is about the issue on how to query the administrator (see Chapter 6), if we used the *by type* method, it would required to ask about the single conflict which is not an easy task, instead using the configuration one allows us to facilitate this process.

Also there might be some cases in which it is important to consider more than two interacting rules, if we just consider conflicts you will never see the bigger picture, since you only acknowledge couple of rules, but with configurations you always have the whole image on how all the rules interact with each other and also the conflict couples.

Since we chose to study the *configuration* method, we need to study all the possible cases in order to find the best order to query the rules to the administrator.

Throughout the analysis and the exploration of the graph for all the possible outcomes, we need a SAT solver in order to analyse the responses from the administrator, given a set of rules and their conflicts. Even though we used theoretically the SAT solver to verify the correctness of the solution, we will not implement it in practice, but just use the Java language.

5.6 Analysis and Considerations on Rule Relationships

In order to understand the general cases, let's start with a simple problem and consider only 3 rules R1, R2 and R3 that interact with each other. Then we have 3 possible couples (R1,R2), (R2,R3), (R1,R3) that represent the relationship between two rules. For each couple there are 4 possible values for a relation: equivalence (E), dominance (DM), correlation (C), disjointness (DJ).

Since the dominance is a non-symmetric relation we can consider it with a double value. So that, $(R_x, R_y) = DM$, it means that R_x includes R_y , but you also need to consider when R_y includes R_x , because it is a different relation, therefore the double value. In the latter case we invert the rules inside the brackets (R_y, R_x) , so that we maintain the syntax.

To get the total number of cases we need to use the formula for disposition with repetition:

$$D = N^k \quad (5.2)$$

In which N is the number of possible values that the couple can assume, in our case it is equal to 5, and k is the number of elements, in our case the couples are 3. Thus, the total number is equal to $5^3 = 125$. All these 125 cases may or may not generate anomalies and eventually conflicts, depending on how the rules interact.

Nothing actually limits the number of rules that you can consider, you can also do the same discussion with four or more rules, but pay attention to the number of all possible cases.

Here is some calculations.

Given the number of rules N , then the number of possible couples of rules is given by combination equation:

$$C = \frac{N!}{2!(N-2)!} \quad (5.3)$$

For each couple you can have 5 possible values of relations (equivalence, two dominance, correlation and disjointness), so $totalcases = 5^C$, which means:

$$totalcases = 5^{\frac{N!}{2!(N-2)!}} \quad (5.4)$$

In case we consider 'just' four rules ($N = 4$), then $totalcases = 15625$ cases, from all these cases we need to perform the analysis and study, removing all the logically impossible configurations and then just focusing on the ones that actually create conflicts.

This means that it is not impossible, it is just a lot of work and time, it is better to concentrate on just three rules in this moment, which are still a total of 125 cases.

The whole initial list given 3 rules can be seen in Appendix [a](#).

First-Level Considerations

By first looking at the whole list we can do some basic considerations and with some logic we can remove some of those cases that are either logically impossible or that are not interesting for our study case.

In case 1: equivalent rules means that they consider the same area of packets, but the actions may be different. If they all have the same action, either allow or deny, it means that those three rules are coincident to just one rule, therefore there is no anomaly. Instead if one rule has an opposite action, for example: $R1 = \text{ALLOW}$, $R2 = \text{DENY}$, $R3 = \text{ALLOW}$ (actually this works for every combination of actions), then $(R1, R2)$ and $(R2, R3)$ are conflict anomalies, specifically contradictions. But since the third relation says that $R1 = R3$ and as hypothesis $R1$ and $R3$ have the same action, then it means that $R1$ and $R3$ are actually the same rule, therefore there is only one contradiction. So there is no problem in studying the order of conflict anomalies since there is just one conflict.

In case there are two couples with relation value = E, independently of which they are, and the third couple has a different relation value (not equivalence): then this case is impossible to occur. Logically the three rules are equal for the transitive property, then this denies the value of the third couple (that is different from equivalence). Therefore, cases 2, 3, 4, 5, 6, 11, 13, 21, 26, 51, 76 and 101 shall not be studied.

In case there is just one couple with relation value equal to E, meanwhile the other two have different relation values like DM-C, DM-DJ, C-DJ, etc (consider all possible dispositions): let's consider for example the case of $(R1, R2) = E$, $(R2, R3) = \text{DM}$, $(R3, R1) = \text{DM}$. Since $R1$ and $R2$ intercept the same group of packets, then you can write instead of $R2$, $R1$, but the relation $(R1, R3)$ can either be DM or DM2, so $R1$ can include or be included $R3$ but not both at the same time, thus this configuration is not possible logically. This argument is valid for all similar cases, the only exceptions are cases 36 and 56, in which this 'rule' does not apply. Therefore, cases 8, 9, 10, 12, 14, 15, 17, 18, 20, 22, 23, 24, 28, 29, 30, 41, 46, 52, 54, 55, 66, 71, 77, 78, 80, 81, 86, 96, 102, 103, 104, 106, 111 and 116 shall not be studied.

In case there are two couples with relation value equal to DJ: independently of the relation value of the third couple there is only one potentially anomaly, so there is no problem in studying the order of conflict anomalies since there is just one conflict. Therefore, cases 25, 50, 75, 100, 105, 110, 115, 120, 121, 122, 123 and 124 shall not be studied.

In case 125: since the three rules are disjoint there will never be an anomaly, there is no intersection between them.

At this point we have $125 - 60 = 65$ cases.

Second-Level Considerations

In case 31: if R1 and R3 both intercept the same group of packets and R1 includes R2, then also R3 includes R2 ((R3, R2)=DM), therefore this case is not possible.

In case 33: if R1 includes R2 and R2 includes R3, then R1 also includes R3, it means that R3 is completely included inside R2 which is also completely included inside R1, therefore is not possible that R1 is at the same time included inside R3.

In case 34: if R1 includes R2 and R2 includes R3, then R1 also includes R3, it means that R3 is completely included inside R2 which is also completely included inside R1, therefore is not possible that R1 is at the same time correlated to R3.

In case 35: if R1 includes R2 and R2 includes R3, then R1 also includes R3, it means that R3 is completely included inside R2 which is also completely included inside R1, therefore is not possible that R1 is at the same time disjoint from R3.

In case 40: if R1 includes R2 and also R3 includes R2, then it means that R1 and R3 have a non empty intersection, so they cannot be disjoint. Therefore this case is not possible.

In case 43: if R2 and R3 are correlated and R1 includes R2, then there has to be a part of R1 that is not intersected with R3, thus R3 does not include R1 and therefore this case is not possible.

In case 45: if R1 includes R2 and R2 and R3 are correlated, it means that R1 and R3 have a non empty intersection, so they cannot be disjoint. Therefore this case is not possible.

In case 48: if R1 includes R2 and R3 includes R1, then R2 and R3 have a non empty intersection, so they cannot be disjoint. Therefore this case is not possible.

In case 61: if R1 and R3 both intercept the same group of packets and R2 includes R1, then it is not possible that R3 includes R2. Therefore this case is not possible.

In case 62: if R2 includes R1 and R3 includes R2, then R3 also includes R1 ((R3, R1) = DM) and not the opposite, therefore this case is not possible.

In case 64: if R2 includes R1 and R3 includes R2, then R3 also includes R1 ((R3, R1) = DM), thus they cannot be correlated. Therefore this case is not possible.

In case 65: if R2 includes R1 and R3 includes R2, then R3 also includes R1 ((R3, R1) = DM), thus they have a non empty intersection. Therefore this case

is not possible.

In case 67: if R2 includes R1 and R1 includes R3, then R3 is completely included inside R2, so they cannot be in a correlated relation. Therefore this case is not possible.

In case 72: if R2 includes R1 and R1 includes R3, then R2 and R3 have a non empty intersection, so they cannot be disjoint. Therefore this case is not possible.

In case 73: if R2 and R3 both include R1, then it means that R2 and R3 share in common at least R1 so they cannot be disjoint. Therefore this case is not possible.

In case 74: if R2 includes R1 and R1 is correlated to R3, then R2 and R3 have a non empty intersection, so they cannot be disjoint. Therefore this case is not possible.

In case 83: if R2 includes R3 and R3 includes R1, then R1 is completely included inside R2, so they cannot be in a correlated relation. Therefore this case is not possible.

In case 87: if R3 includes R2 and R1 includes R3, then R2 is completely included inside R1, so they cannot be in a correlated relation. Therefore this case is not possible.

In case 90: if R1 and R2 are correlated and R3 includes R2, it means that R1 and R3 have a non empty intersection, so they cannot be disjoint. Therefore this case is not possible.

In case 98: if R3 includes R1 and R1 and R2 are correlated, it means that R2 and R3 have a non empty intersection, so they cannot be disjoint. Therefore this case is not possible.

In case 107: if R1 and R2 both include R3, then it means that R1 and R2 share in common at least R3 so they cannot be disjoint. Therefore this case is not possible.

In case 108: if R2 includes R3 and R3 includes R1, then R1 and R2 have a non empty intersection, so they cannot be disjoint. Therefore this case is not possible.

In case 109: if R2 includes R3 and R1 and R3 are correlated, then R1 and R2 have a non empty intersection, so they cannot be disjoint. Therefore this case is not possible.

In case 112: if R1 includes R3 and R3 includes R2, then R2 is completely included inside R1, so R1 and R2 have a non empty intersection, so they cannot be disjoint. Therefore this case is not possible.

In case 117: if R1 includes R3 and R2 and R3 are correlated, then R1 and R2 have a non empty intersection, so they cannot be disjoint. Therefore this case is not possible.

The remaining cases are $65-25=40$ cases.

Third-Level Considerations

In case 32 and 37: they determine the same identical scenario, therefore one of the two case can be dropped. Let's remove case 37.

In case 36 and 56: they determine the same identical scenario, therefore one of the two case can be dropped. Let's remove case 56.

At this point we have $40-2 = 38$ cases.

Fourth-Level Considerations

Given these characteristics then we can notice that when the relation between the rules is not dominance then it is possible to extract a sort of 'class leader' to represent the group with the same configuration.

This can be done only when there is no dominance relation between the rules because it does not matter the rule priority, therefore it is equivalent to study one case instead of another.

To be clear, we will not remove them actually, but we will not study them since they are just the same case but mirrored.

In cases 19, 79 and 91: they can be simplified by just considering case 19.

In cases 95, 99 and 119: they can be simplified by just considering case 95.

At this point we have: $38-4$ cases = 34 cases.

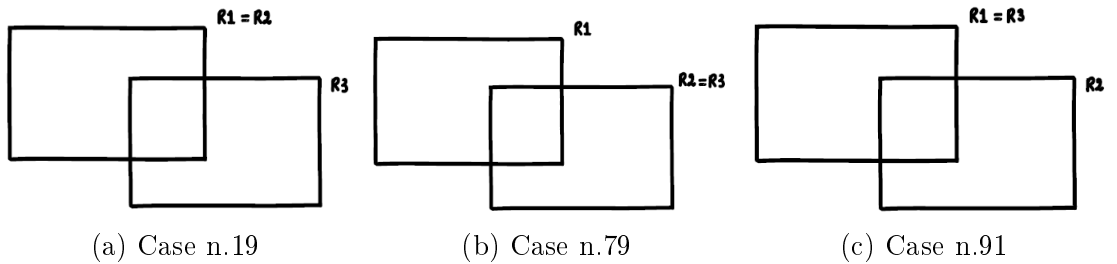


Figure 5.1: Mirror cases of n.19

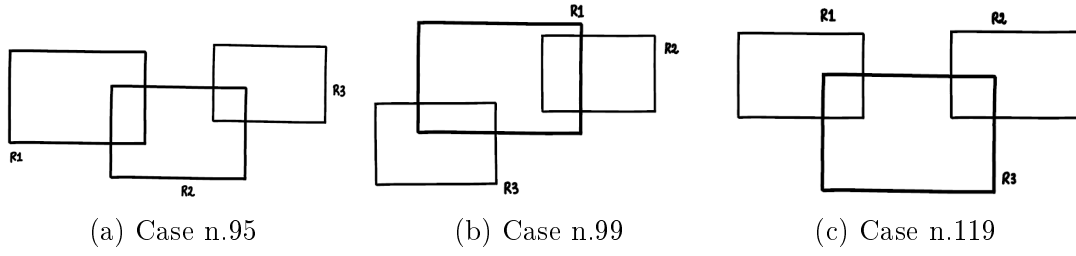


Figure 5.2: Mirror cases of n.95

Fifth-Level Considerations

In case 13: there is just one potential contradiction conflict (R1,R2). Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

In case 38: considering all possible action values that the rules can assume, you always can get at most one shadow conflict (R1,R2) that still needs to be solved. Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

In case 53: there is just one potential contradiction conflict (R1,R2). Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

In case 58: considering all possible action values that the rules can assume, you always can get at most one shadow conflict (R2,R3) that still needs to be solved. Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

In case 60: considering all possible action values that the rules can assume, you always can get at most one shadow conflict (R2,R3) that still needs to be solved. Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

In case 63: this configuration does not generate any anomalies, in fact it is the correct way to configure a firewall. Thus, there is no need to study it.

In case 68: considering all possible action values that the rules can assume, you always can get at most one correlation conflict (R2,R3) that still needs to be solved. Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

In case 70: considering all possible action values that the rules can assume, you always can get at most one correlation conflict (R2,R3) that still needs to be solved. Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

In case 88: considering all possible action values that the rules can assume, you always can get at most one correlation conflict (R1,R2) that still needs to be solved. Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

In case 113: this configuration does not generate any anomalies. Thus there is no need to study it.

In case 114: considering all possible action values that the rules can assume, you always can get at most one correlation conflict (R1,R3) that still needs to be solved. Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

In case 118: considering all possible action values that the rules can assume, you always can get at most one correlation conflict (R2,R3) that still needs to be solved. Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

At this point we have: $34 - 12 = 22$ cases.

Final List

The final list of all the cases that we need to study one by one, considering also the actions of each rule and how they interact with each other, can be seen in Appendix [b](#).

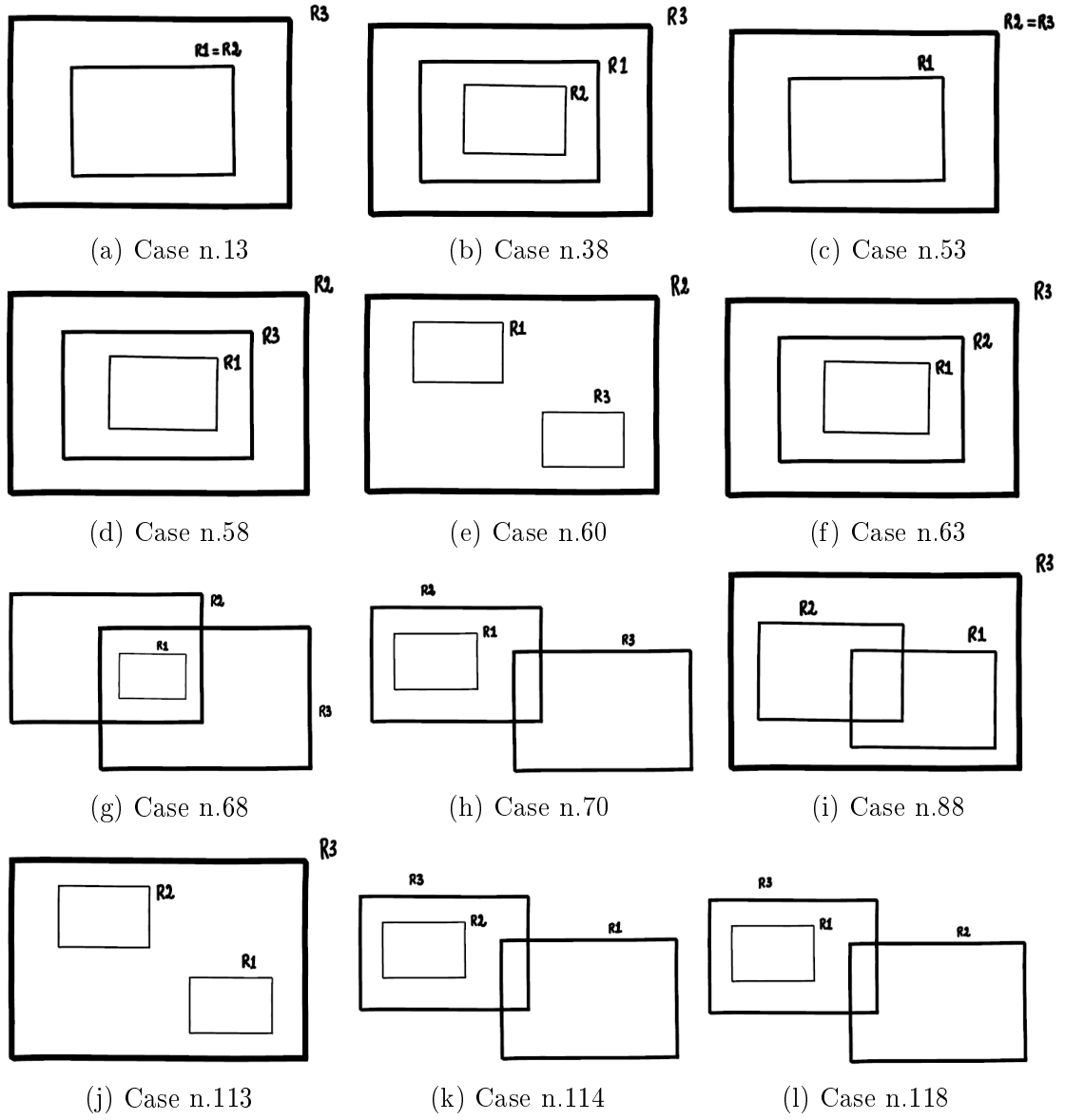


Figure 5.3: Fifth-Level Considerations Cases

Chapter 6

Case Study Analysis

Given a single configuration, you have to imagine which was the final idea of the administrator when they were designing it. When we consider the actions and how their value affect the presence of conflicts, we will not study the mirror version of the same case, since it is the same case but just mirrored and it does not create any additional interesting value.

CASE N.7: $(R1, R2) = E \text{ --- } (R2, R3) = DM \text{ --- } (R1, R3) = DM$

Let's call the internal area the zone defined by the intercepted packets by the rule R3.

Let's call external area the zone defined by the intercepted packets by the rules R1 and R2 and not R3.

Given these two areas the administrator may want:

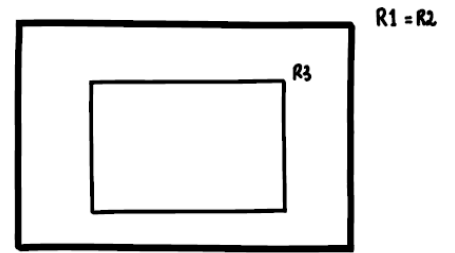
- A) concordant sign, ALLOW or DENY, it does not matter.
- B) discordant sign: external zone = ALLOW/-DENY, internal zone = DENY/ALLOW. Then it means that:

- R1 and R2: it has to stay the one which has a different action from R3
- R3: has to 'win' against R1/R2 in order to stay 'over' them

Let's consider the actions rules and how they can generate or not conflicts.

$R1 = A, R2 = A, R3 = D$: since you remove one between R1 and R2 because of duplication, there is just one conflict of type shadow conflict, in case you removed R2 then $(R1, R3)$. Thus there is no problem in studying the order of conflict

Figure 6.1: Case n.7



anomalies since there is just one conflict.

$R1 = A, R2 = D, R3 = A$: there is a contradiction $(R1, R2)$ and a shadow conflict $(R2, R3)$. In order to obtain:

- case A, then it means that R1 needs to win the contradiction, you remove R2 and R3 is also removed because of shadow redundancy (case A - only R1). Or it R2 needs to win the contradiction, you can remove R1, and R3 needs to lose the shadow conflict, you remove R3 (case A - only R2).
- case B, then it means that R2 needs to win the contradiction, you remove R1 and R3 wins the shadow conflict, you reorder the rules such that $R3 > R2$ (case B - outsideDENY insideALLOW).

$R1 = A, R2 = D, R3 = D$: there is a contradiction $(R1, R2)$ and a shadow conflict $(R1, R3)$. In order to obtain:

- case A, then it means that R2 needs to win the contradiction, you remove R1 and R3 is also removed because of shadow redundancy (case A - only R2). Or it R1 wins the contradiction, you can remove R2, and R3 needs to lose the shadow conflict, you remove R3 (case A - only R1).
- case B, then it means that R1 needs to win the contradiction, you remove R2 and R3 wins the shadow conflict, you reorder the rules such that $R3 > R1$ (case B - outsideALLOW insideDENY).

Let's query the admin in order to see with how many questions we can reach all of these scenarios.

At the first step you can either ask about R1 or R2 in order to understand which one of the two you have to keep, we choose R1, if we would have chosen R2 the arguments would be symmetric.

1) Q: "is R1 ok?"

If answer is YES: then all the area that is inside R1 has action which has the same action value of R1, you can remove R2 and R3, either because it loses the shadow conflict or because it is redundant (case A - only R1).

If answer is NO: then it can be that between the contradiction $(R1, R2)$ R2 wins and all the area has the same action value of R2 or it could happen that the two zones have different action values. Thus, we need to ask more questions.

2) Q: "is R2 ok?"

If answer is YES: then all the area that is inside R2 has action which has the same action value of R2, you can remove R1 and R3, either because it loses the shadow conflict or because it is redundant (case A - only R2).

If answer is NO: then it means that the internal area has a different action value from the external area (case B). You also know that the last rule is necessarily correct, therefore the contradiction (R1,R2) is won by the rule that has action value opposite from R3.

\implies Total number of steps: Best Case = 1 Worst Case = 2

In case of querying the admin from the internal area we would not get any advantages, since we cannot get any deductions from a smaller group. So you would have to query twice the admin, first asking for R3 and then either R1 or R2 in order to get to a solution. In this case best and worst case are the same and they are equal to 2.

Instead starting from the 'outside', there is a chance in which you can stop with just one query.

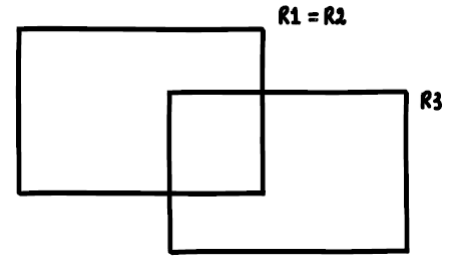
CASE N.19: $(R1, R2) = E \text{ — } (R2, R3) = C \text{ — } (R1, R3) = C$

Let's call the central area the zone defined by the intercepted packets by the rules R1 (or R2) and R3.

Let's call the higher area the zone defined by the intercepted packets only by the rules R1 and R2.

Let's call the lower area the zone defined by the intercepted packets only by the rule R3.

Figure 6.2: Case n.19



Given these three areas the administrator may want:

- A) all three areas with concordant sign, ALLOW or DENY, it does not matter.
- B) discordant sign: central area has the same action value with the higher area. Then it means that:
 - R1 and R2: they either coincide or just one of them can stay and it has to stay the one which has a different action value from R3
 - R3: has to 'lose' against R1/R2
- C) discordant sign: central area has the same action value with the lower area. Then it means that:
 - R1 and R2: they either coincide or just one of them can stay and it has to stay the one which has a different action value from R3
 - R3: has to 'win' against R1/R2 in order to stay 'over' them

Let's consider the actions rules and how they can generate or not conflicts.

$R1 = A, R2 = A, R3 = D$: since you remove one between R1 and R2 because of duplication, there is just one conflict of type correlation, in case you removed R2 then $(R1, R3)$. Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

$R1 = A, R2 = D, R3 = A$: there is a contradiction $(R1, R2)$ and a correlation $(R2, R3)$. In order to obtain:

- case A, then it means that R1 needs to win the contradiction, you remove R2 (case A).
- case B, then it means that R2 needs to win the contradiction, you remove R1 and R3 loses the correlation (case B).

- case C, then it means that R2 needs to win the contradiction, you remove R1 and R3 wins the correlation, you need to reorder the rules such that $R3 > R2$ (case C).

$R1 = A$, $R2 = D$, $R3 = D$: there is a contradiction (R1,R2) and a correlation (R1,R3). In order to obtain:

- case A, then it means that R2 needs to win the contradiction, you remove R1 (case A).
- case B, then it means that R1 needs to win the contradiction, you remove R2 and R3 loses the correlation (case B).
- case C, then it means that R1 needs to win the contradiction, you remove R2 and R3 wins the correlation, you need to reorder the rules such that $R3 > R1$ (case C).

Let's query the admin in order to see with how many questions we can reach all of these scenarios.

1) Q: "is R1 ok?"

If answer is YES + R3 has the same action value of R1: then it means that R1 wins the contradiction, you remove R2 and the lower area has the same action value of R1 (case A).

If answer is YES + R3 has a different action value from R1: then it means that R1 wins the contradiction, you remove R2 and the lower area has a different action value from R1 (case B)

If answer is NO: then it means that either between the contradiction (R1,R2) R2 wins and so all area has the same action of R2 (case A), or maybe it may be that the central area has a different sign from R1 (case B or C). Thus, we need to ask more questions.

2) Q: "is R2 ok?"

If answer is YES + R2 has the same action value of R3: then it means that R2 wins the contradiction, you remove R1 and the lower area has the same action value of R2 (case A).

If answer is YES + R2 has a different action value from R1: then it means that R2 wins the contradiction, you remove R1 and the lower area has a different action from R2 (case B).

If answer is NO: then it means that the central area has the same action value as the lower area (case C). You also know that the last rule is necessarily correct, therefore the contradiction (R1,R2) is won by the rule that has action value opposite from R3.

\implies Total number of steps: Best Case = 1 Worst Case = 2

Starting from R2 and then asking for R1 does not change the overall number of steps that need to be done, but only the intermediate arguments that you have to do.

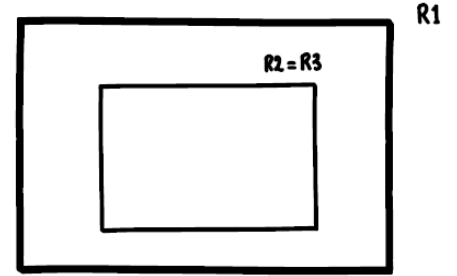
If you actually started from R3: you would solve immediately the correlation conflict, but you would not be able to say anything about the contradiction conflict between (R1,R2), therefore you need at least two steps, so that the best case and worst case are the same and equal to 2.

CASE N.27: $(R1, R2) = DM \text{ --- } (R2, R3) = E \text{ --- } (R1, R3) = DM$

Let's call the internal area the zone defined by the intercepted packets by the rules R2 and R3.

Let's call external area the zone defined by the intercepted packets by the rule R1 and not by R2 and R3.

Figure 6.3: Case n.27



Given these two areas the administrator may want:

- A) concordant sign, ALLOW or DENY, it does not matter.
- B) discordant sign: external zone = ALLOW/-DENY, internal zone = DENY/ALLOW. Then it means that:
 - R2 and R3: it has to stay the one which has a different action from R1
 - R1: has to 'lose' against R2/R3 in order to stay 'under' them

Let's consider the actions rules and how they can generate or not conflicts.

$R1 = A, R2 = A, R3 = D$: there is a contradiction $(R2, R3)$ and a shadow conflict $(R1, R3)$. In order to obtain:

- case A, then it means that R2 needs to win the contradiction, you remove R3 and R2 is also removed because of shadow redundancy (case A).
- case B, then it means that R3 needs to win the contradiction, you remove R2 and R3 wins the shadow conflict, you reorder the rules such that $R3 > R1$ (case B - outsideALLOW insideDENY).

$R1 = A, R2 = D, R3 = A$: there is a contradiction $(R2, R3)$ and a shadow conflict $(R1, R2)$. In order to obtain:

- case A, then it means that R3 needs to win the contradiction, you remove R2 and R3 is also removed because of shadow redundancy (case A).
- case B, then it means that R2 needs to win the contradiction, you remove R3 and R2 wins the shadow conflict, you reorder the rules such that $R2 > R1$ (case B - outsideALLOW insideDENY).

$R1 = A, R2 = D, R3 = D$: since you remove one between R2 and R3 because of duplication, there is just one conflict of type shadow conflict, in case you removed R3 then $(R1, R2)$. Thus there is no problem in studying the order of conflict

anomalies since there is just one conflict.

Let's query the admin in order to see with how many questions we can reach all of these scenarios.

1) Q: "is R1 ok?"

If answer is YES: then all the area that is inside R1 has action which has the same action value of R1, you can remove R2 (case A).

If answer is NO: then since in the external area there is no conflict but only in the internal one, this means that the internal area has a different action value from R1, therefore it is case B, so in the conflict (R2,R3) it wins the rule which has a different action value from R1 and then you reorder the rules.

⇒ Total number of steps: Best Case = 1 Worst Case = 1

If we started querying the admin from the internal area, even though it has more conflicts we cannot get any deductions on the external area. So if you queried "is R2 ok?" (same thing for R3), you would only be able to solve the contradiction conflict, but not the shadow conflict one, therefore you would need to ask also for R1, obtaining 2 steps at least and also at most (best and worst cases are equal to 2).

CASE N.32: $(R1, R2) = DM \text{ --- } (R2, R3) = DM \text{ --- } (R1, R3) = DM$

Let's call the external area the zone defined by the intercepted packets by the rule R1.

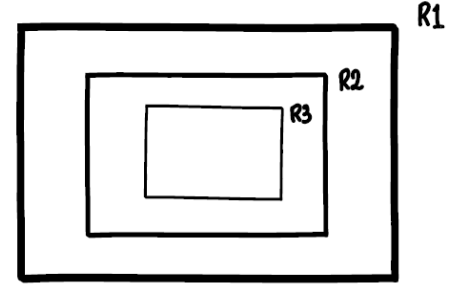
Let's call the central area the zone defined by the intercepted packets only by the rule R2.

Let's call the internal area the zone defined by the intercepted packets only by the rule R3.

Given these three areas the administrator may want:

- A) all three areas with the same action value, ALLOW or DENY, it does not matter. Then it means that R1 is imposing its action on all the region and you can remove the other two rules because of shadow redundancy.
- B) different action values: (external area + central area) and internal area have different action values
- C) different action values: external area and (central area + internal area) have different action values
- D) different action values: central area and (external area + internal area) have different action values

Figure 6.4: Case n.32



Let's consider the actions rules and how they can generate or not conflicts.

$R1 = A, R2 = A, R3 = D$: there are two shadow conflicts $(R1, R3)$ and $(R2, R3)$. In order to obtain:

- case A, then it means that R1 needs to win the shadow conflict $(R1, R3)$, you remove R3 and R2 is also removed because of shadow redundancy (case A).
- case B, then it means that R3 needs to win the shadow conflicts $(R1, R3)$ and $(R2, R3)$, you reorder the rules such that $R3 > R1$ and $R3 > R2$ (case B).

$R1 = A, R2 = D, R3 = A$: there are two shadow conflicts $(R1, R2)$ and $(R2, R3)$. In order to obtain:

- case A, then it means that R3 needs to win the shadow conflict $(R1, R2)$, you remove R2 and R3 is also removed because of shadow redundancy (case A).
- case C, then it means that R2 needs to win the shadow conflicts $(R1, R2)$ and $(R2, R3)$, you remove R3 and you reorder the rules such that $R2 > R1$ (case C).

- case D, then it means that R2 needs to win the shadow conflict (R1,R2), you reorder the rules such that $R2 > R1$ and R3 wins the shadow conflict (R2,R3), you reorder the rules such that $R3 > R2$ (case D).

$R1 = A$, $R2 = D$, $R3 = D$: there are two shadow conflicts (R1,R2) and (R1,R3). In order to obtain:

- case A, then it means that R1 needs to win the shadow conflicts (R1,R2) and (R1,R3), you remove R2 and R2 (case A).
- case B, then it means that R1 needs to win the shadow conflict (R1,R2), you remove R2 and R3 wins the shadow conflict (R1,R3), you reorder the rules such that $R3 > R1$ (case B).
- case C, then it means that R2 needs to win the shadow conflict (R1,R2), you reorder the rules such that $R2 > R1$ and R3 wins the shadow conflict (R1,R3), you reorder the rules such that $R3 > R1$ (case C).

Let's query the admin in order to see with how many questions we can reach all of these scenarios.

1) Q: "is R1 ok?"

If answer is YES: then it means that all the region has the same action value of R1, you can remove R2 and R3 either because of shadow redundancy or because they lost the shadow conflicts (case A).

If answer is NO: then it means that at least one of the two areas have a rule which has a different action value from R1. Thus, we need to ask more questions.

2) Q: "is R2 ok?"

If answer is YES: then it means that R2 has a different action value from R1 and the central area and the internal area have the same action value but this is different from the external area one (case C). You may remove R3 because of shadow redundancy.

If answer is NO + R2 has the same action value of R1: then it means that the internal area has a different action value from the rest (case B).

If answer is NO + R2 has a different action value from R1: then it means that the internal and the external area have the same action value, but this is different from the central area (case D).

\implies Total number of steps: Best Case = 1 Worst Case = 2

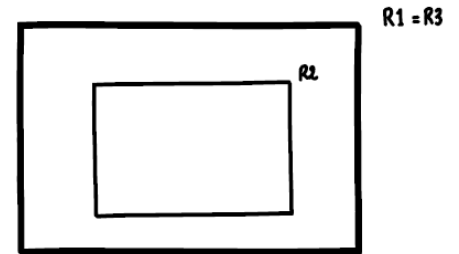
In case of we started querying the admin from the internal rule, we would not get any advantages, since we cannot get any deductions from a smaller group.

CASE N.36: $(R1, R2) = DM \text{ --- } (R3, R2) = DM \text{ --- } (R1, R3) = E$

Let's call the internal area the zone defined by the intercepted packets by the rules R2.

Let's call external area the zone defined by the intercepted packets by the rules R1 and R3 and not by R2.

Figure 6.5: Case n.36



Given these two areas the administrator may want:

- A) concordant sign, ALLOW or DENY, it does not matter.
- B) discordant sign: external zone = ALLOW/-DENY, internal zone = DENY/ALLOW. Then it means that:
 - R1 and R3: it has to stay the one which has a different action from R2
 - R2: has to 'win' against R1/R3 in order to stay 'over' them

Let's consider the actions rules and how they can generate or not conflicts.

$R1 = A, R2 = A, R3 = D$: there is just one conflict of type contradiction (R1,R3). Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

$R1 = A, R2 = D, R3 = A$: since you remove one between R1 and R3 because of duplication, there is just one conflict of type shadow conflict, in case you removed R3 then (R1,R2). Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

$R1 = A, R2 = D, R3 = D$: there is a contradiction (R1,R3) and a shadow conflict (R1,R2). In order to obtain:

- case A, then it means that R3 needs to win the contradiction, you remove R1 and R2 is also removed because of shadow redundancy (case A - only R3) Or R1 wins the contradiction, you remove R3 and R2 loses the shadow conflict and it is removed (case A - only R1).
- case B, then it means that R1 needs to win the contradiction, you remove R3 and R2 wins the shadow conflict, you reorder the rules such that $R2 > R1$ (case B - outsideALLOW insideDENY).

Let's query the admin in order to see with how many questions we can reach all of these scenarios.

1) Q: "is R1 ok?"

If answer is YES: then it means that all the area that is inside R1 has action which has the same action value of R1, you can remove R3 and R2 because of shadow conflict (case A - only R1).

If answer is NO: then it can be that between the contradiction (R1, R3) R3 wins and all the area has the same action value of R3 or it could happen that the two zones have different action values. Thus, we need to ask more questions.

2) Q: "is R3 ok?"

If answer is YES: then it means that all the area that is inside R3 has action which has the same action value of R1, you can remove R1 and R2 because of shadow redundancy (case A - only R3).

If answer is NO: then it means that the external and internal area have different action values, so you remove the rule that has the same action value of R2, in this case R3 and then reorder the rules such that $R2 > R1$ (case B).

\implies Total number of steps: Best Case = 1 Worst Case = 2

After the first query in which you ask for R1 and you find out that R1 is indeed wrong, it is better not to query the admin for R2. This is because you would be able only to solve the shadow conflict, but not the contradiction conflict: you would be able to know that the R2's action value is valid inside the internal area, but you do not know the value in the external one.

Also considering that $R1 = A$, $R2 = D$ and $R3 = D$, knowing that R1 is wrong, then R2 can only be correct, you would never get the response from the admin 'R2 = NO'. When R1 is wrong it can only be that either case A with only R3, which has the same action value as R2, so the answer from the admin would be still equal to YES or case B in which you need to reorder $R2 > R3$, but this would still return from the admin YES as an answer.

So, in this case querying for R2 is useless and just a waste of time, therefore you would need an additional step to reach all the possible cases, which mean that the best case is still 1 but the worst case is increased up to 3.

CASE N.39: $(R1, R2) = DM \text{ --- } (R3, R2) = DM \text{ --- } (R1, R3) = C$

Let's call the external area the zone defined by the intercepted packets by the rules R1 and R3.

Let's call the internal area the zone defined by the intercepted packets only by the rule R2.

Given these two areas the administrator may want:

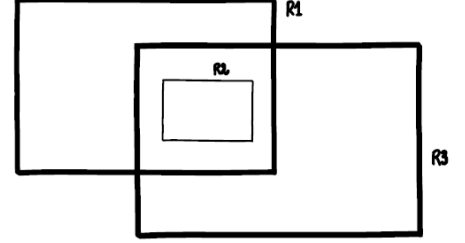


Figure 6.6: Case n.39

- A) both areas with the same action value, ALLOW or DENY, it does not matter.
- B) different action values: external zone = ALLOW/DENY, internal zone = DENY/ALLOW.
Then it means that:

- the values of the regions of only R1 and only R3 are not that interesting since they cannot have conflict, but keep in mind that their value determine the value of the external area
- the external area can have the same action value either with the region of only R1 or with the region of only R3.

Let's consider the actions rules and how they can generate or not conflicts.

$R1 = A, R2 = A, R3 = D$: there is a correlation $(R1, R3)$ and a shadow conflict $(R2, R3)$. In order to obtain:

- case A, then it means that R1 needs to win the correlation and R2 needs to win the shadow conflict, you remove R3. Or R3 wins the correlation, you reorder the rules such that $R3 > R1$ and R3 also wins the shadow conflict and you reorder the rules such that $R3 > R2$ (case A).
- case B, then it means that R1 needs to win the correlation and R2 loses the shadow conflict, you reorder the rules such that $R3 > R2$ (case B - outside-ALLOW insideDENY).

$R1 = A, R2 = D, R3 = A$: there are two shadow conflicts $(R1, R2)$ and $(R2, R3)$. In order to obtain:

- case A, then it means that R2 needs to lose the shadow conflict with R1, you remove R2 (case A).
- case B, then it means that R2 needs to win both the shadow conflicts, you reorder the rules such that $R2 > R1$ and $R2 > R3$ (case B - outsideALLOW insideDENY).

$R1 = A, R2 = D, R3 = D$: there is a correlation ($R1, R3$) and a shadow conflict ($R1, R2$). In order to obtain:

- case A, then it means that $R1$ needs to win both the correlation and the shadow conflict, you remove $R2$ (case A). Or $R3$ wins the correlation, you reorder the rules such that $R3 > R1$ and $R1$ loses the shadow conflict and you reorder the rules such that $R2 > R1$ (case A).
- case B, then it means that $R1$ needs to win the correlation and $R2$ wins the shadow conflict, you reorder the rules such that $R2 > R1$ (case B - outside-ALLOW insideDENY).

Let's query the admin in order to see with how many questions we can reach all of these scenarios.

1) Q: "is $R1$ ok?"

If answer is YES: then it means that both areas have the same action value (case A) in which either $R1$ wins the correlation conflict with $R3$ or $R1$ and $R3$ have the same action.

If answer is NO: then it means that either one of the two areas or both of them have a different action from $R1$. Thus, we need to ask more questions.

2) Q: "is $R3$ ok?"

If answer is YES: then it means that both areas have the same action value (case A) and $R3$ wins the correlation conflict with $R1$.

If answer is NO: then it means that the external and the internal area have different actions (case B), but the exact action value of each area depends on the rule's action.

\implies Total number of steps: Best Case = 1 Worst Case = 2

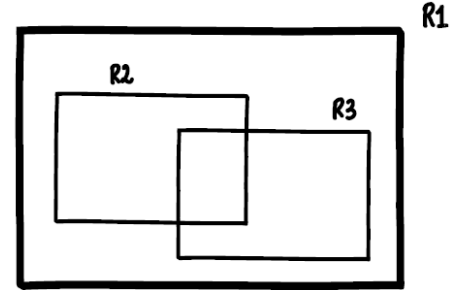
In this case as the second query we ask about $R3$, and not about $R2$ which is the second by priority but in this case is the lowest in terms of area of packets. This is done because asking about the smallest area does not give any additional information in order to get to the final scenarios, so it is better to first query about $R3$. We could also have started by querying with $R3$ and then proceed with $R1$.

CASE N.42: $(R1, R2) = DM$ — $(R2, R3) = C$ — $(R1, R3) = DM$

Let's call the left area the zone defined by the intercepted packets only by the rule R2.

Let's call the right area the zone defined by the intercepted packets only by the rule R3.

Let's call the central area the zone defined by the intercepted packets by the rules R2 and R3.



Given these three areas the administrator may want:

- A) all three areas with the same action value, ALLOW or DENY, it does not matter. If the three areas have the same action of R1 then you can remove the other two rules because of shadow redundancy.
- B) different action values: (right area + central area) and left area have different actions
- C) different action values: (left area + central area) and right area have different actions

Figure 6.7: Case n.42

Let's consider the actions rules and how they can generate or not conflicts.

$R1 = A$, $R2 = A$, $R3 = D$: there is a correlation $(R2, R3)$ and a shadow conflict $(R1, R3)$. In order to obtain:

- case A, then it means that R1 needs to win the shadow conflict, you remove R3 and also R2 because of shadow redundancy (case A).
- case B, then it means that R3 needs to win the correlation, you reorder the rules such that $R3 > R2$, and R1 loses the shadow conflict, you reorder the rules such that $R3 > R1$ (case B).
- case C, then it means that R2 needs to win the correlation and R3 wins the shadow conflict, you reorder the rules such that $R3 > R1$ (case C).

$R1 = A$, $R2 = D$, $R3 = A$: there is a correlation $(R2, R3)$ and a shadow conflict $(R1, R2)$. In order to obtain:

- case A, then it means that R2 needs to lose the shadow conflict, you remove R2 and also R3 because of shadow redundancy (case A).
- case B, then it means that R3 needs to win the correlation, you reorder the rules such that $R3 > R2$ and R2 wins the shadow conflict, you reorder the rules such that $R2 > R1$ (case B).

- case C, then it means that R2 wins both the correlation and the shadow conflict, you reorder the rules such that $R2 > R1$ (case C).

$R1 = A, R2 = D, R3 = D$: there are two shadow conflicts $(R1, R2)$ and $(R1, R3)$. In order to obtain:

- case A, then it means that R2 and R3 need to win the shadow conflicts with R1, you reorder the rules such that $R2 > R1$ and $R3 > R1$ (case A). Or R1 needs to win both the shadow conflicts and you remove both R2 and R3 (case A).
- case B, then it means that R2 needs to lose the shadow conflict, you remove R2, and R3 wins the shadow conflict, you reorder the rules such that $R3 > R1$ (case B).
- case C, then it means that R3 needs to lose the shadow conflict, you remove R3, and R2 wins the shadow conflict, you reorder the rules such that $R2 > R1$ (case C).

Let's query the admin in order to see with how many questions we can reach all of these scenarios.

1) Q: "is R1 ok?"

If answer is YES: then it is the case in which all three regions have the same action (case A) and also the areas' action is the same as R1's.

If answer is NO: then it might be that the three area have different values (case B or C) or they are all the same (case A) but it has a different action value from with R1's. Thus, we need to ask more questions.

2) Q: "is R2 ok?"

If answer is YES + R2 has the same action value of R3: then it is the case in which all three area have the same action (case A) but the value is different from with R1's.

If answer is YES + R2 has a different action value from R3: then it is the case in which the central area has the same action value of the left area (case C).

If answer is NO: then R2 and R3 have different action values and the central area has the same action of the right area (case B).

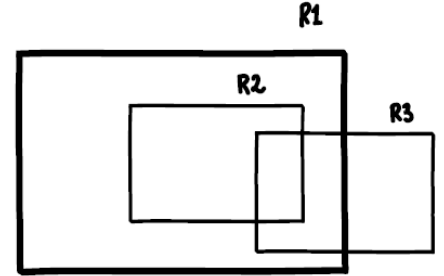
\Rightarrow Total number of steps: Best Case = 1 Worst Case = 2

CASE N.44: $(R1, R2) = DM$ — $(R2, R3) = C$ — $(R1, R3) = C$

Let's call the left area the zone defined by the intercepted packets only by the rule R2.

Let's call the right area the zone defined by the intercepted packets only by the rules R1 and R3.

Let's call the central area the zone defined by the intercepted packets by the rules R2 and R3.



Given these three areas the administrator may want:

Figure 6.8: Case n.44

- A) all three areas with the same action value, ALLOW or DENY, it does not matter. If the three areas have the same action as R1 then you can remove R2 because of shadow redundancy.
- B) different action value: (right area + central area) and left area have different actions
- C) different action value: (left area + central area) and right area have different actions

Let's consider the actions rules and how they can generate or not conflicts.

$R1 = A$, $R2 = A$, $R3 = D$: there are two correlations $(R1, R3)$ and $(R2, R3)$. In order to obtain:

- case A, then it means that R3 needs to lose both the correlations (case A).
- case B, then it means that R3 needs to win both the correlations, you reorder the rules such that $R3 > R2$ and $R3 > R1$ (case B).
- case C, then it means that R3 needs to win the correlation with R1, you reorder the rules such that $R3 > R1$, but R3 loses the correlation with R2 (case C).

$R1 = A$, $R2 = D$, $R3 = A$: there is a correlation $(R2, R3)$ and a shadow conflict $(R1, R2)$. In order to obtain:

- case A, then it means that R2 needs to lose the shadow conflict, you remove R2 (case A).
- case B, then it means that R2 needs to win the shadow conflict, you reorder the rules such that $R2 > R1$ and R3 wins the correlation, you reorder the rules such that $R3 > R2$ (case B).

- case C, then it means that R2 wins both the correlation and the shadow conflict, you reorder the rules such that $R2 > R1$ (case C).

$R1 = A$, $R2 = D$, $R3 = D$: there is a correlation ($R1, R3$) and a shadow conflict ($R1, R2$). In order to obtain:

- case A, then it means that R2 needs to win the shadow conflict with R1, you reorder the rules such that $R2 > R1$ and R3 wins the correlation, you reorder the rules such that $R3 > R1$ (case A). Or R1 needs to win the shadow conflict, you remove R2 and R1 also wins the correlation (case A).
- case B, then it means that R2 needs to lose the shadow conflict, you remove R2, and R3 wins the correlation, you reorder the rules such that $R3 > R1$ (case B).
- case C, then it means that R3 needs to lose the correlation and R2 wins the shadow conflict, you reorder the rules such that $R2 > R1$ (case C).

Let's query the admin in order to see with how many questions we can reach all of these scenarios.

1) Q: "is R1 ok?"

If answer is YES: then it is the case in which all three regions have the same action value (case A) and also the areas have the same action as R1.

If answer is NO: then it might be that the three area have different values (case B or C) or they have the same action (case A) but their value is a different action from R1's. Thus, we need to ask more questions.

2) Q: "is R3 ok?"

If answer is YES + R3 has a different action value from R2: then it is the case in which the central area has the same action value of the right area (case B).

If answer is YES + R3 has the same action value of R2: then it means that at least the central and the right areas have the same action value, but you are not sure about the left area. Thus, we need to ask more questions.

If answer is NO: then it means that it is the case in which the central area has the same action value of the left area (case C).

3) Q: "is R2 ok?"

If answer is YES: then it is the case in which all the three areas have the same action value but this value is different from R1's (case A).

If answer is NO: then it is the case in which the central area has the same action value as the right area (case B).

\implies Total number of steps: Best Case = 1 Worst Case = 3

If you started querying from R2, you would not obtain any information on the possible correlation (R1,R3). If you started querying from R3 you would not obtain any information on the possible shadow conflict (R1,R2). Both imply that you have to query at least twice the administrator, so the best case is equal to 2, while the worst case is still 3.

CASE N.47: $(R1, R2) = DM$ — $(R2, R3) = DJ$ — $(R1, R3) = DM$

Let's call the left area the zone defined by the intercepted packets only by the rule R2.

Let's call the right area the zone defined by the intercepted packets only by the rule R3.

Given these two areas the administrator may want:

- A) both areas with the same action value, ALLOW or DENY, it does not matter. If the two areas have also the same action as R1 then you can remove them because of shadow redundancy.
- B) different action value: the one with the same action as R1 is removed for shadow redundancy.

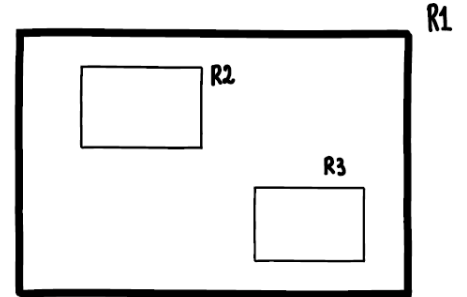


Figure 6.9: Case n.47

Let's consider the actions rules and how they can generate or not conflicts.

$R1 = A$, $R2 = A$, $R3 = D$: there is just one conflict of type shadow conflict $(R1, R3)$. Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

$R1 = A$, $R2 = D$, $R3 = A$: there is just one conflict of type shadow conflict $(R1, R2)$. Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

$R1 = A$, $R2 = D$, $R3 = D$: there are two shadow conflicts $(R1, R2)$ and $(R1, R3)$. In order to obtain:

- case A, then it means that R1 needs to win both the shadow conflicts, you remove R2 and R3 (case A). Or R2 and R3 need to win the shadow conflicts, you need to reorder the rules such that $R2 > R1$ and $R3 > R1$ (case A).
- case B, then it means that only one between R2 and R3 needs to lose the shadow conflict and the other one has to win it: either R2 loses it, you remove R2 and R3 wins it, you reorder the rules such that $R3 > R1$, or R2 wins it, you reorder the rules such that $R2 > R1$ and R3 loses it, you remove R3 (case B).

Let's query the admin in order to see with how many questions we can reach all of these scenarios.

1) Q: "is R1 ok?"

If answer is YES: then it means that both areas have the same action value (case

A) and they have the same action as R1 so you can remove them.

If answer is NO: then it means that either one of the two areas or both of them have a different actions from R1. Thus, we need to ask more questions.

2) Q: "is R2 ok?"

If answer is YES + R2 has a different action value from R3: then it means that only one area has a different action value from R1 (case B).

If answer is YES + R2 has the same action value of R3: then it means that both area have the same action values (case A), but the action is opposite from R1's one.

If answer is NO: then it means that R1 has imposed its sign on the left area, so the area with a different action is R3 (case B).

\implies Total number of steps: Best Case = 1 Worst Case = 2

If you start querying from R2 or R3 does not bring any advantage, so you would need at least 2 queries in the best case and in the worst case 3.

CASE N.49: $(R1, R2) = DM$ — $(R2, R3) = DJ$ — $(R1, R3) = C$

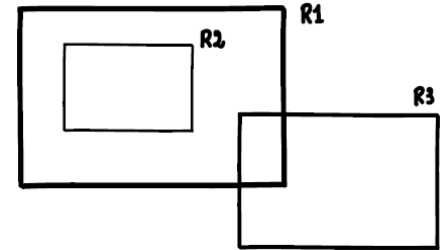
Let's call the left area the zone defined by the intercepted packets only by the rule R2.

Let's call the right area the zone defined by the intercepted packets by the rules R1 and R3.

Given these two areas the administrator may want:

- A) both areas with the same action value, ALLOW or DENY, it does not matter. If the left area has the same action as R1 then you can remove R2 because of shadow redundancy.
- B) different action values: if the left area has the same action of R1, then you can remove R2 for shadow redundancy.

Figure 6.10: Case n.49



Let's consider the actions rules and how they can generate or not conflicts.

$R1 = A, R2 = A, R3 = D$: there is just one conflict of type correlation $(R1, R3)$. Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

$R1 = A, R2 = D, R3 = A$: there is just one conflict of type shadow conflict $(R1, R2)$. Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

$R1 = A, R2 = D, R3 = D$: there is a shadow conflicts $(R1, R2)$ and a correlation $(R1, R3)$. In order to obtain:

- case A, then it means that R1 needs to win both the shadow conflict and the correlation, you remove R2 (case A). Or R2 needs to win the shadow conflict, you reorder the rules such that $R2 > R1$ and R3 wins the correlation, you reorder the rules such that $R3 > R1$ (case A).
- case B, then it means that only one between R2 and R3 needs to lose their conflict and the other one has to win it: either R2 loses it, you remove R2 and R3 wins it, you reorder the rules such that $R3 > R1$, or R2 wins it, you reorder the rules such that $R2 > R1$ and R3 loses it (case B).

Let's query the admin in order to see with how many questions we can reach all of these scenarios.

1) Q: "is R1 ok?"

If answer is YES: then it means that both the area have the action value (case A) and this has the same action value of R1.

If answer is NO: then it means that either the two areas have the same action value but this is opposite from R1's or they have different action values. Thus, we need to ask more questions.

2) Q: "is R2 ok?"

If answer is YES: then it means that R2 has won its conflict, but you do not know anything about R3's conflict, it could be either that both areas have different action value from R1 or it could be the case of areas with different actions. Thus, we need to ask more questions.

If answer is NO: then it means that R1 has imposed its action value on the left area, and since R1 is 'wrong', the right area has a different action value from (case B).

3) Q: "is R3 ok?"

If answer is YES: then it means that also R3 has won its conflict, therefore it is the case in which both areas have the same action value but this value is different from R1's (case A).

If answer is NO: then it means that R3 has lost its conflict, therefore the right area assumes R1's action value (case B).

\implies Total number of steps: Best Case = 1 Worst Case = 3

Starting querying from R2 or R3 does not bring any advantages since R2 and R3 are disjoint, therefore you would have to query at least twice the admin in the best case, meanwhile the worst case is still equal to 3. After asking for R1, it is equivalent, since they are disjoint, querying first R2 or R3.

CASE N.57: $(R2, R1) = DM \text{ --- } (R2, R3) = DM \text{ --- } (R1, R3) = DM$

Let's call the external area the zone defined by the intercepted packets by the rule R2.

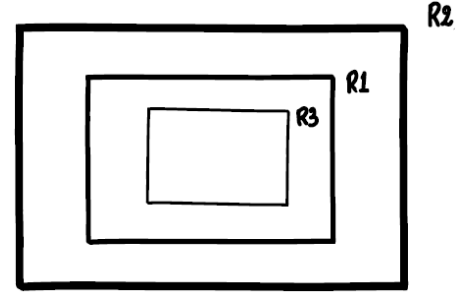
Let's call the central area the zone defined by the intercepted packets only by the rule R1.

Let's call the internal area the zone defined by the intercepted packets only by the rule R3.

Given these three areas the administrator may want:

- A) all three areas with the same action value, ALLOW or DENY, it does not matter. Then it means that R2 is imposing its action on all the region and you can remove the other two rules because of shadow redundancy.
- B) different action values: (external area + central area) and internal area have different action values
- C) different action values: external area and (central area + internal area) have different action values
- D) different action values: central area and (external area + internal area) have different action values

Figure 6.11: Case n.57



Let's consider the actions rules and how they can generate or not conflicts.

$R1 = A, R2 = A, R3 = D$: there are two shadow conflicts $(R1, R3)$ and $(R2, R3)$. In order to obtain:

- case A, then it means that R1 needs to win the shadow conflict $(R1, R3)$, you remove R3 (case A).
- case B, then it means that R3 needs to win both the shadow conflicts $(R1, R3)$ and $(R2, R3)$, you reorder the rules such that $R3 > R1$ and $R3 > R2$ (case B).

$R1 = A, R2 = D, R3 = A$: there is just one conflict of type shadow conflict $(R2, R3)$. Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

$R1 = A, R2 = D, R3 = D$: there is just one conflict of type shadow conflict $(R1, R3)$. Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

Let's query the admin in order to see with how many questions we can reach all of these scenarios.

1) Q: "is R2 ok?"

If answer is YES: then it means that all the region has the same action value of R2, you can remove R1 and R3 because of shadow redundancy (case A).

If answer is NO: then it means that one of the two shadow conflicts is won by R3 (case B), you can reorder $R3 > R1$ and $R3 > R2$.

\implies Total number of steps: Best Case = 1 Worst Case = 1

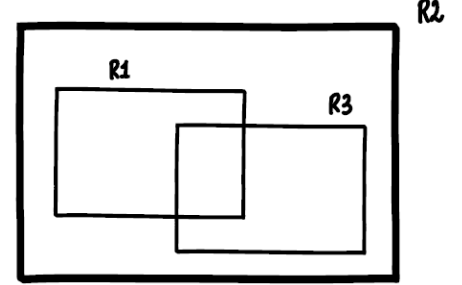
In case of we started querying the admin from the internal rule, we would not get any advantages, since we cannot get any deductions from a smaller group. So you would have to query at least twice the admin, which is already a worse strategy than this.

CASE N.59: $(R2, R1) = DM \text{ --- } (R2, R3) = DM \text{ --- } (R1, R3) = C$

Let's call the left area the zone defined by the intercepted packets only by the rule R1.

Let's call the right area the zone defined by the intercepted packets only by the rule R3.

Let's call the central area the zone defined by the intercepted packets by the rules R1 and R3.



Given these three areas the administrator may want:

- A) all three areas with the same action value, ALLOW or DENY, it does not matter. If the three areas have the same action of R2 then you can remove the other two rules because of shadow redundancy.
- B) different action values: (right area + central area) and left area have different actions
- C) different action values: (left area + central area) and right area have different actions

Figure 6.12: Case n.59

Let's consider the actions rules and how they can generate or not conflicts.

$R1 = A, R2 = A, R3 = D$: there is a correlation $(R1, R3)$ and a shadow conflict $(R2, R3)$. In order to obtain:

- case A, then it means that R2 needs to win the shadow conflict, you remove R3 (case A).
- case B, then it means that R3 needs to win the correlation, you reorder the rules such that $R3 > R1$, and R3 wins the shadow conflict, you reorder the rules such that $R3 > R2$ (case B).
- case C, then it means that R1 needs to win the correlation and R3 wins the shadow conflict, you reorder the rules such that $R3 > R2$ (case C).

$R1 = A, R2 = D, R3 = A$: there is just one conflict of type shadow conflict $(R2, R3)$. Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

$R1 = A, R2 = D, R3 = D$: there is just one conflict of type correlation $(R1, R3)$. Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

Let's query the admin in order to see with how many questions we can reach all of these scenarios.

1) Q: "is R2 ok?"

If answer is YES: then it is the case in which all three regions have the same action (case A) and also the areas' action is the same as R2's.

If answer is NO: then it might be that the three area have different values (case B or C). Thus, we need to ask more questions.

2) Q: "is R1 ok?"

If answer is YES: then it is the case in which the central area has the same action value of the left area (case C).

If answer is NO: then it means that central area has the same action of the right area (case B).

\implies Total number of steps: Best Case = 1 Worst Case = 2

CASE N.69: $(R2, R1) = DM$ — $(R2, R3) = C$ — $(R1, R3) = C$

Let's call the left area the zone defined by the intercepted packets only by the rule R1.

Let's call the right area the zone defined by the intercepted packets only by the rules R2 and R3.

Let's call the central area the zone defined by the intercepted packets by the rules R1 and R3.

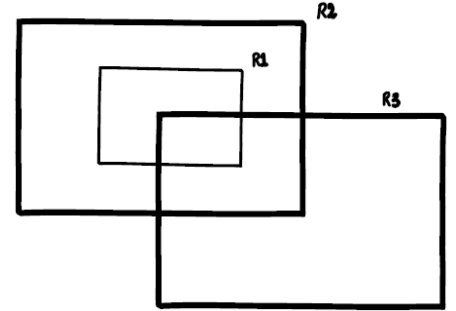


Figure 6.13: Case n.69

Given these three areas the administrator may want:

- A) all three areas with the same action value, ALLOW or DENY, it does not matter. If the three areas have the same action as R2 then you can remove R1 because of shadow redundancy.
- B) different action value: (right area + central area) and left area have different actions
- C) different action value: (left area + central area) and right area have different actions

Let's consider the actions rules and how they can generate or not conflicts.

$R1 = A$, $R2 = A$, $R3 = D$: there are two correlations $(R1, R3)$ and $(R2, R3)$. In order to obtain:

- case A, then it means that R3 needs to lose both the correlations (case A).
- case B, then it means that R3 needs to win both the correlations, you reorder the rules such that $R3 > R2$ and $R3 > R1$ (case B).
- case C, then it means that R3 needs to win the correlation with R2, you reorder the rules such that $R3 > R2$, but R3 loses the correlation with R1 (case C).

$R1 = A$, $R2 = D$, $R3 = A$: there is just one conflict of type correlation $(R2, R3)$. Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

$R1 = A$, $R2 = D$, $R3 = D$: there is just one conflict of type correlation $(R1, R3)$. Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

Let's query the admin in order to see with how many questions we can reach all of these scenarios.

1) Q: "is R2 ok?"

If answer is YES: then it is the case in which all three regions have the same action value (case A) and also the areas have the same action as R2.

If answer is NO: then it might be that the three area have different values (case B or C). Thus, we need to ask more questions.

2) Q: "is R1 ok?"

If answer is YES: then it is the case in which the central area has the same action value of the left area (case C).

If answer is NO: then it is the case in which the central area has the same action value of the right area (case B).

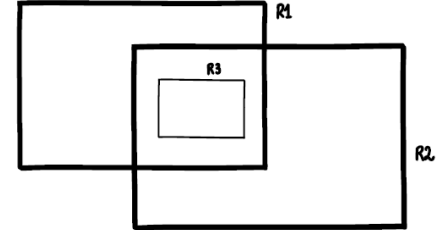
\implies Total number of steps: Best Case = 1 Worst Case = 2

If you started querying from R1, you would not obtain any information on the possible correlation (R2,R3). If you started querying from R3 you would still need in the best case one step and in the worst case three steps in order to reach all the possible cases.

CASE N.82: $(R1, R2) = C$ — $(R3, R2) = DM$ — $(R1, R3) = DM$

Let's call the external area the zone defined by the intercepted packets by the rules R1 and R2.

Let's call the internal area the zone defined by the intercepted packets only by the rule R3.



Given these two areas the administrator may want:

- A) both areas with the same action value, ALLOW or DENY, it does not matter.
- B) different action values: external zone = ALLOW/DENY, internal zone = DENY/ALLOW.
Then it means that:

Figure 6.14: Case n.82

- the values of the regions of only R1 and only R2 are not that interesting since they cannot have conflict, but keep in mind that their value determine the value of the external area
- the external area can have the same action value either with the region of only R1 or with the region of only R2.

Let's consider the actions rules and how they can generate or not conflicts.

$R1 = A$, $R2 = A$, $R3 = D$: there are two shadow conflicts $(R1, R3)$ and $(R2, R3)$. In order to obtain:

- case A, then it means that R3 needs to lose the shadow conflict either with R1 or with R2, you remove R3 (case A).
- case B, then it means that R3 needs to win both the shadow conflicts, you reorder the rules such that $R3 > R1$ and $R3 > R2$ (case B - outsideALLOW insideDENY).

$R1 = A$, $R2 = D$, $R3 = A$: there is a correlation $(R1, R2)$ and a shadow conflict $(R2, R3)$. In order to obtain:

- case A, then it means that R1 needs to win the correlation and R3 needs to win the shadow conflict, you reorder the rules such that $R3 > R2$. Or R2 wins the correlation, you reorder the rules such that $R2 > R1$ and R2 wins the shadow conflict and you remove R3 (case A).
- case B, then it means that R2 needs to win the correlation and R3 loses the shadow conflict, you reorder the rules such that $R3 > R2$ (case B - outsideDENY insideALLOW).

$R1 = A, R2 = D, R3 = D$: there is a correlation $(R1, R2)$ and a shadow conflict $(R1, R3)$. In order to obtain:

- case A, then it means that $R2$ needs to win the correlation, you reorder the rules such that $R2 > R1$ and $R3$ wins the shadow conflict, you reorder the rules such that $R3 > R1$ (case A). Or $R1$ wins both the correlation and shadow conflict, you remove $R3$ (case A).
- case B, then it means that $R1$ needs to win the correlation and $R3$ wins the shadow conflict, you reorder the rules such that $R3 > R1$ (case B - outside-ALLOW insideDENY).

Let's query the admin in order to see with how many questions we can reach all of these scenarios.

1) Q: "is $R1$ ok?"

If answer is YES: then it means that both areas have the same action value (case A) in which either $R1$ wins the correlation conflict with $R2$ or $R1$ and $R2$ have the same action.

If answer is NO: then it means that either one of the two areas or both of them have a different action from $R1$. Thus, we need to ask more questions.

2) Q: "is $R2$ ok?"

If answer is YES: then it means that both areas have the same action value (case A).

If answer is NO: then it means that the external and the internal area have different actions (case B), but the exact action value of each area depends on the rule's action.

\implies Total number of steps: Best Case = 1 Worst Case = 2

CASE N.84: $(R2,R1) = C$ — $(R2,R3) = DM$ — $(R1,R3) = C$

Let's call the left area the zone defined by the intercepted packets only by the rules R1 and R2.

Let's call the right area the zone defined by the intercepted packets only by the rules R2 and R3.

Let's call the central area the zone defined by the intercepted packets by the rules R1 and R3.

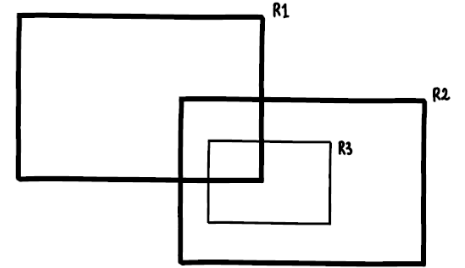


Figure 6.15: Case n.84

Given these three areas the administrator may want:

- A) all three areas with the same action value, ALLOW or DENY, it does not matter. If the three areas have the same action as R2 then you can remove R3 because of shadow redundancy.
- B) different action value: (right area + central area) and left area have different actions
- C) different action value: (left area + central area) and right area have different actions

Let's consider the actions rules and how they can generate or not conflicts.

$R1 = A$, $R2 = A$, $R3 = D$: there is a correlation $(R1,R3)$ and a shadow conflict $(R2,R3)$. In order to obtain:

- case A, then it means that R3 needs to lose the shadow conflict, you remove R3 (case A).
- case B, then it means that R3 needs to win both the correlation and the shadow conflict, you reorder the rules such that $R3 > R2$ and $R3 > R1$ (case B).
- case C, then it means that R3 needs to win the shadow conflict with R2, you reorder the rules such that $R3 > R2$, but R3 loses the correlation with R1 (case C).

$R1 = A$, $R2 = D$, $R3 = A$: there is a correlation $(R1,R2)$ and a shadow conflict $(R2,R3)$. In order to obtain:

- case A, then it means that R2 needs to lose the shadow conflict, you reorder the rules such that $R3 > R2$ and R2 loses the correlation with R1 (case A).

- case B, then it means that R3 needs to win the shadow conflict, you reorder the rules such that $R3 > R2$ and R2 wins the correlation, you reorder the rules such that $R2 > R1$ (case B).

$R1 = A, R2 = D, R3 = D$: there are two correlations ($R1, R2$) and ($R1, R3$). In order to obtain:

- case A, then it means that R1 needs to lose both the correlations, you reorder the rules such that $R3 > R1$ and $R2 > R1$ (case A).
- case B, then it means that R1 needs to lose the correlation with R3, you reorder the rules such that $R3 > R1$ but R1 wins the correlation with R2 (case B).
- case C, then it means that R1 needs to win both the correlations (case C).

Let's query the admin in order to see with how many questions we can reach all of these scenarios.

1) Q: "is R2 ok?"

If answer is YES: then it is the case in which all three regions have the same action value (case A) and also the areas have the same action as R2.

If answer is NO: then it might be that the three area have different values (case B or C). Thus, we need to ask more questions.

2) Q: "is R1 ok?"

If answer is YES + R1 has a different action value from R3: then it is the case in which the central area has the same action value of the left area (case C).

If answer is YES + R1 has the same action value of R3: then it means that at least the central and the left areas have the same action value, but you are not sure about the right area. Thus, we need to ask more questions.

If answer is NO: then it is the case in which the central area has the same action value of the right area (case B).

3) Q: "is R3 ok?"

If answer is YES: then it is the case in which the three areas have the same action but the value is different from R2's (case A).

If answer is NO: then it is the case in which the central area has the same action value of the left area (case C).

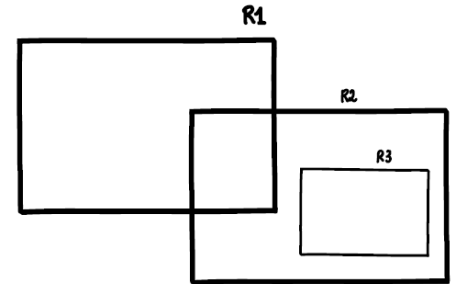
\Rightarrow Total number of steps: Best Case = 1 Worst Case = 3

CASE N.85: $(R1, R2) = C$ — $(R2, R3) = DM$ — $(R1, R3) = DJ$

Let's call the left area the zone defined by the intercepted packets by the rules R1 and R2.

Let's call the right area the zone defined by the intercepted packets only by the rule R3.

Figure 6.16: Case n.85



Given these two areas the administrator may want:

- A) both areas with the same action value, ALLOW or DENY, it does not matter. If the right area has the same action as R2 then you can remove R3 because of shadow redundancy.
- B) different action values: if the right area has the same action of R2, then you can remove R3 for shadow redundancy.

Let's consider the actions rules and how they can generate or not conflicts.

$R1 = A, R2 = A, R3 = D$: there is just one conflict of type shadow conflict (R2,R3). Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

$R1 = A, R2 = D, R3 = A$: there is a shadow conflict (R2,R3) and a correlation (R1,R2). In order to obtain:

- case A, then it means that R3 needs to win the shadow conflict with R2, you reorder the rules such that $R3 > R2$ and R1 wins the correlation (case A). Or R2 needs to win the shadow conflict, you remove R3 and R2 wins the correlation, you reorder the rules such that $R2 > R1$ (case A).
- case B, then it means that only one between R1 and R3 needs to win their conflict and the other one has to lose it: either R1 wins the correlation and R2 wins the shadow conflict, you remove R3, or R3 wins the shadow conflict, you reorder the rules such that $R3 > R2$ and R1 loses the correlation, you reorder the rules such that $R2 > R1$ (case B).

$R1 = A, R2 = D, R3 = D$: there is just one conflict of type correlation (R1,R2). Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

Let's query the admin in order to see with how many questions we can reach all of these scenarios.

1) Q: "is R2 ok?"

If answer is YES: then it means that both the area have the action value (case A) and this has the same action value of R2.

If answer is NO: then it means that either the two areas have the same action value but this is opposite from R2's or they have different action values. Thus, we need to ask more questions.

2) Q: "is R3 ok?"

If answer is YES: then it means that R3 has won its conflict, but you do not know anything about R1's conflict, it could be either that both areas have different action value from R2 or it could be the case of areas with different actions. Thus, we need to ask more questions.

If answer is NO: then it means that R2 has imposed its action value on the right area, and since R2 is 'wrong', the right area has a different action value from (case B). 3) Q: "is R1 ok?" If answer is YES: then it means that also R1 has won its conflict, therefore it is the case in which both areas have the same action value but this value is different from R2's (case A).

If answer is NO: then it means that R1 has lost its conflict, therefore the right area assumes R2's action value (case B).

\implies Total number of steps: Best Case = 1 Worst Case = 3

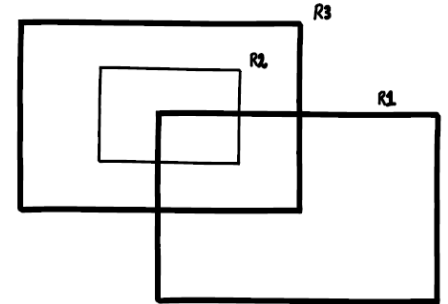
Starting querying from R1 or R3 does not bring any advantages since R1 and R3 are disjoint, therefore you would have to query at least twice the admin in the best case, meanwhile the worst case is still equal to 3. After asking for R2, it is equivalent, since they are disjoint, querying first R1 or R3.

CASE N.89: $(R2, R1) = C$ — $(R3, R2) = DM$ — $(R1, R3) = C$

Let's call the left area the zone defined by the intercepted packets only by the rules R2 and R3.

Let's call the right area the zone defined by the intercepted packets only by the rules R1 and R3.

Let's call the central area the zone defined by the intercepted packets by the rules R1 and R2.



Given these three areas the administrator may want:

Figure 6.17: Case n.89

- A) all three areas with the same action value, ALLOW or DENY, it does not matter. If the three areas have the same action as R3 then you can remove R2 because of shadow redundancy.
- B) different action value: (right area + central area) and left area have different actions
- C) different action value: (left area + central area) and right area have different actions

Let's consider the actions rules and how they can generate or not conflicts.

$R1 = A, R2 = A, R3 = D$: there is just one conflict of type correlation $(R1, R3)$. Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

$R1 = A, R2 = D, R3 = A$: there is just one conflict of type correlation $(R1, R2)$. Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

$R1 = A, R2 = D, R3 = D$: there are two correlations $(R1, R3)$ and $(R1, R2)$. In order to obtain:

- case A, then it means that R1 needs to lose both the correlations, you reorder the rules such that $R3 > R1$ and $R2 > R1$ (case A).
- case B, then it means that R1 needs to win both the correlations (case B).
- case C, then it means that R1 needs to win the correlation with R3, but R1 loses the correlation with R2, you reorder the rules such that $R2 > R1$ (case C).

Let's query the admin in order to see with how many questions we can reach all of these scenarios.

1) Q: "is R1 ok?"

If answer is YES: then it is the case in which the central area has the same action value of the right area (case B).

If answer is NO: then it might be that the three area have different values (case B or C). Thus, we need to ask more questions.

2) Q: "is R3 ok?"

If answer is YES: then it is the case in which all three regions have the same action value (case A) and also the areas have the same action as R3.

If answer is NO: then it is the case in which the central area has the same action value of the left area (case C).

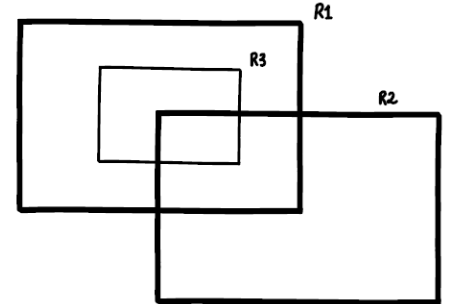
\implies Total number of steps: Best Case = 1 Worst Case = 2

CASE N.92: $(R1, R2) = C \text{ — } (R2, R3) = C \text{ — } (R1, R3) = DM$

Let's call the left area the zone defined by the intercepted packets only by the rules R1 and R3.

Let's call the right area the zone defined by the intercepted packets only by the rules R1 and R2.

Let's call the central area the zone defined by the intercepted packets by the rules R2 and R3.



Given these three areas the administrator may want: Figure 6.18: Case n.92

- A) all three areas with the same action value, ALLOW or DENY, it does not matter. If the three areas have the same action as R1 then you can remove R3 because of shadow redundancy.
- B) different action value: (right area + central area) and left area have different actions
- C) different action value: (left area + central area) and right area have different actions

Let's consider the actions rules and how they can generate or not conflicts.

$R1 = A, R2 = A, R3 = D$: there is a correlation $(R2, R3)$ and a shadow conflict $(R1, R3)$. In order to obtain:

- case A, then it means that R3 needs to lose the shadow conflict, you remove R3 (case A).
- case B, then it means that R3 needs to win the shadow conflict with R2, you reorder the rules such that $R3 > R2$, but R3 loses the correlation with R1 (case B).
- case C, then it means that R3 needs to win both the correlation and the shadow conflict, you reorder the rules such that $R3 > R2$ and $R3 > R1$ (case C).

$R1 = A, R2 = D, R3 = A$: there are two correlations $(R1, R2)$ and $(R2, R3)$. In order to obtain:

- case A, then it means that R2 needs to lose both the correlations, you reorder the rules such that $R3 > R2$ (case A).
- case B, then it means that R2 needs to win both the correlations, you reorder the rules such that $R2 > R1$ (case B).

- case C, then it means that R2 needs to win the correlation with R1, you reorder the rules such that $R2 > R1$ but R2 loses the correlation with R3, you reorder the rules such that $R3 > R2$ (case C).

$R1 = A$, $R2 = D$, $R3 = D$: there is a correlation (R1,R2) and a shadow conflict (R1,R3). In order to obtain:

- case A, then it means that R1 needs to lose the shadow conflict, you reorder the rules such that $R3 > R1$ and R1 loses the correlation, you reorder the rules such that $R2 > R1$ (case A).
- case B, then it means that R1 needs to win the shadow conflict, you remove R3 and R2 wins the correlation, you reorder the rules such that $R2 > R1$ (case B).
- case C, then it means that R1 needs to win the correlation and R1 loses the shadow conflict, you reorder the rules such that $R3 > R1$ (case C).

Let's query the admin in order to see with how many questions we can reach all of these scenarios.

1) Q: "is R1 ok?"

If answer is YES: then it is the case in which all three regions have the same action value (case A) and also the areas have the same action as R1.

If answer is NO: then it might be that the three areas have different values (case B or C). Thus, we need to ask more questions.

2) Q: "is R2 ok?"

If answer is YES + R2 has a different action value from R3: then it is the case in which the central area has the same action value of the right area (case B).

If answer is YES + R3 has the same action value as R2: then it is either the case that the central area has the same action value of the right area or all three areas have the same action value but it is different from R1's. Thus, we need to ask more questions.

If answer is NO: then it is the case in which the central area has the same action value of the left area (case C).

3) Q: "is R3 ok?"

If answer is YES: then it is the case in which the three areas have the same action but the value is different from R1's (case A).

If answer is NO: then it is the case in which the central area has the same action value of the right area (case B).

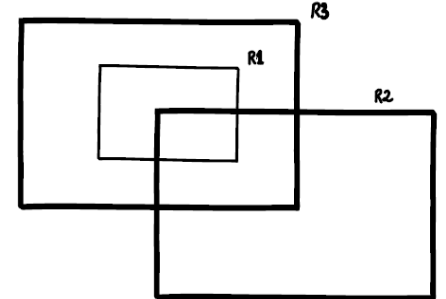
\Rightarrow Total number of steps: Best Case = 1 Worst Case = 3

CASE N.93: $(R1, R2) = C \text{ — } (R2, R3) = C \text{ — } (R3, R1) = DM$

Let's call the left area the zone defined by the intercepted packets only by the rules R1 and R3.

Let's call the right area the zone defined by the intercepted packets only by the rules R2 and R3.

Let's call the central area the zone defined by the intercepted packets by the rules R1 and R2.



Given these three areas the administrator may want: Figure 6.19: Case n.93

- A) all three areas with the same action value, ALLOW or DENY, it does not matter. If the three areas have the same action as R3 then you can remove R1 because of shadow redundancy.
- B) different action value: (right area + central area) and left area have different actions
- C) different action value: (left area + central area) and right area have different actions

Let's consider the actions rules and how they can generate or not conflicts.

$R1 = A, R2 = A, R3 = D$: there is just one conflict of type correlation $(R2, R3)$. Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

$R1 = A, R2 = D, R3 = A$: there are two correlations $(R1, R2)$ and $(R2, R3)$. In order to obtain:

- case A, then it means that R2 needs to lose both the correlations, you reorder the rules such that $R3 > R2$ (case A).
- case B, then it means that R2 needs to win both the correlations, you reorder the rules such that $R2 > R1$ (case B).
- case C, then it means that R2 needs to win the correlation with R3, but R2 loses the correlation with R1 (case C).

$R1 = A, R2 = D, R3 = D$: there is just one conflict of type correlation $(R1, R2)$. Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

Let's query the admin in order to see with how many questions we can reach all of these scenarios.

1) Q: "is R3 ok?"

If answer is YES: then it is the case in which all three regions have the same action value (case A) and also the areas have the same action as R3.

If answer is NO: then it might be that the three area have different values (case B or C). Thus, we need to ask more questions.

2) Q: "is R2 ok?"

If answer is YES: then it is the case in which the central area has the same action value of the right area (case B).

If answer is NO: then it is the case in which the central area has the same action value of the left area (case C).

\implies Total number of steps: Best Case = 1 Worst Case = 2

CASE N.94: $(R1, R2) = C \text{ — } (R2, R3) = C \text{ — } (R1, R3) = C$

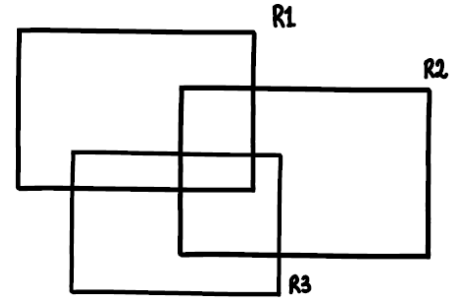
Let's call the R13 area the zone defined by the intercepted packets only by the rules R1 and R3.

Let's call the R12 area the zone defined by the intercepted packets only by the rules R1 and R2.

Let's call the R23 area the zone defined by the intercepted packets by the rules R2 and R3.

Let's call the R123 area the zone defined by the intercepted packets by the rules R1, R2 and R3.

Figure 6.20: Case n.94



Given these areas the administrator may want:

- A) all four areas with the same action value, ALLOW or DENY, it does not matter.
- B) discordant sign: three areas with the same action value and one with a different action value

Let's consider the actions rules and how they can generate or not conflicts.

$R1 = A, R2 = A, R3 = D$: there are two correlations $(R1, R3)$ and $(R2, R3)$. In order to obtain:

- case A, then it means that R1 needs to win the correlation with R3 and R2 wins the correlation with R3 (case A).
- case B, then it means that R3 needs to win both the correlations, you reorder the rules such that $R3 > R1$ and $R3 > R2$ (case B). Or R3 needs to lose the correlation with R1, but R3 wins the correlation with R2, you reorder the rules such that $R3 > R2$ (case B). Or R3 needs to win the correlation with R1, you reorder the rules such that $R3 > R1$, but R3 loses the correlation with R2 (case B).

$R1 = A, R2 = D, R3 = A$: there are two correlations $(R1, R2)$ and $(R2, R3)$. In order to obtain:

- case A, then it means that R1 needs to win the correlation with R2 and R3 wins the correlation with R2, you reorder the rules such that $R3 > R2$ (case A).
- case B, then it means that R1 needs to win the correlation with R2 and R2 wins the correlation with R3 (case B). Or R2 needs to win the correlation

with R1, you reorder the rules such that $R2 > R1$ and R3 wins the correlation with R2, you reorder the rules such that $R3 > R2$ (case B). Or R2 needs to win the correlation with R1, you reorder the rules such that $R2 > R1$ and R2 wins the correlation with R3 (case B).

$R1 = A$, $R2 = D$, $R3 = D$: there are two correlations ($R1, R2$) and ($R1, R3$). In order to obtain:

- case A, then it means that R2 needs to win the correlation with R1, you reorder the rules such that $R2 > R1$ and R3 wins the correlation with R1, you reorder the rules such that $R3 > R1$ (case A).
- case B, then it means that R1 needs to win both the correlations (case B). Or R2 needs to win the correlation with R1, you reorder the rules such that $R2 > R1$ and R3 loses the correlation with R1 (case B). Or R1 needs to win the correlation with R2 and R3 wins the correlation with R1, you reorder the rules such that $R3 > R1$ (case B).

Let's query the admin in order to see with how many questions we can reach all of these scenarios. In this case you can start from whichever one since all of them are in the same number of possible conflicts.

1) Q: "is R1 ok?"

If answer is YES + R2 has the same action value as R3: then it means that it is the case in which only three areas have the same action value (case B).

If answer is YES + R2 has a different action value from R3: then it means that at least three areas have the same action value, but you are not sure if it is the case of four areas with R1's action value or just three areas with the same action and one different. Thus, we need to ask more questions.

1.1) Q: "is R2 ok?"

If answer is YES + R1 has the same action value as R2: then it means that all four areas have the same action value (case A).

If answer is YES + R1 has a different action value from R2: then it means that it is the case in which only three areas have the same action value (case B).

If answer is NO + R1 has the same action value of R2: then it means that it is the case in which only three areas have the same action value (case B).

If answer is NO + R1 has a different action value from R2: then it means that all four areas have the same action value (case A).

If answer is NO: then you cannot say anything, you are still in doubt between the case B or case A. Thus, we need to ask more questions.

1.2) Q: "is R2 ok?"

If answer is YES + R2 has a different action value from R3: then it means that it is the case in which only three areas have the same action value (case B).

If answer is YES + R2 has the same action value from R3: then it means that you are still in doubt between the case B or case A. Thus, we need to ask more questions.

1.2.1) Q: "is R3 ok?"

If answer is YES: then it means that all four area have the same action value (case A).

If answer is NO: then it means that it is the case in which only three areas have the same action value (case B).

If answer is NO: then it means that it is the case in which only three areas have the same action value (case B).

⇒ Total number of steps: Best Case = 1 Worst Case = 3

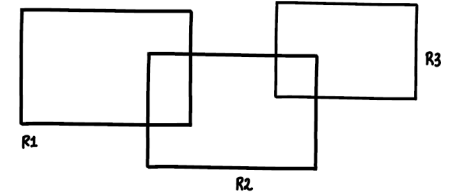
In this case there you can start querying the administrator from any rule that you want, since they are all linked together by correlation. You would always obtain the same number of steps in any order you ask the admin.

CASE N.95: $(R1, R2) = C \text{ — } (R2, R3) = C \text{ — } (R1, R3) = DJ$

Let's call the left area the zone defined by the intercepted packets by the rules R1 and R2.

Let's call the right area the zone defined by the intercepted packets by the rules R2 and R3.

Figure 6.21: Case n.95



Given these two areas the administrator may want:

- A) both areas with concordant sign, ALLOW or DENY, it does not matter.
- B) discordant sign

Let's consider the actions rules and how they can generate or not conflicts.

$R1 = A, R2 = A, R3 = D$: there is just one conflict of type correlation $(R2, R3)$. Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

$R1 = A, R2 = D, R3 = A$: there are two correlation conflicts $(R1, R2)$ and $(R2, R3)$. In order to obtain:

- case A, then it means that R2 needs to win both the correlations, you reorder the rules such that $R2 > R1$ (case A). Or R2 needs to lose both the correlations, you reorder the rules such that $R3 > R2$ (case A).
- case B, then it means that R2 wins only one between the two correlations: either R2 wins the correlation with R1 and loses the one with R3, you reorder the rules such that $R2 > R1$, or R2 wins the correlation with R3 and loses the one with R1 (case B).

$R1 = A, R2 = D, R3 = D$: there is just one conflict of type correlation $(R1, R2)$. Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

Let's query the admin in order to see with how many questions we can reach all of these scenarios.

1) Q: "is R2 ok?"

If answer is YES: then it means that both areas have the same action (case A), and it is the same value as R2.

If answer is NO: then it means that either the two areas have a different action value or they have the same action but it is different from R2's. Thus, we

need to ask more questions.

2) Q: "is R1 ok?"

If answer is YES: then it means that R2 has lost at least the conflict with R1, but you do not know anything about the conflict with R3. Thus, we need to ask more questions.

If answer is NO: then it means that the two areas have different action values (case B).

3) Q: "is R3 ok?"

If answer is YES: then it means that R2 has lost also the conflict with R3, therefore the two areas have the same action value but this is different from R2's (case A).

If answer is NO: then it means that the two areas have different action values (case B), so R2 has won the conflict with R3 but R2 has lost the conflict with R1.

\implies Total number of steps: Best Case = 1 Worst Case = 3

In this case it is more convinient starting querying the admin from the rule that is part of the most conflict: if you started from R1, you would get information only on the left area, same thing if you started from R3 you would only get information on the right area, but since you are querying the 'central' rule you can get hints on both the areas, thus there is a chance on terminating the process with just one step.

CASE N.97: $(R1, R2) = C \text{ --- } (R2, R3) = DJ \text{ --- } (R1, R3) = DM$

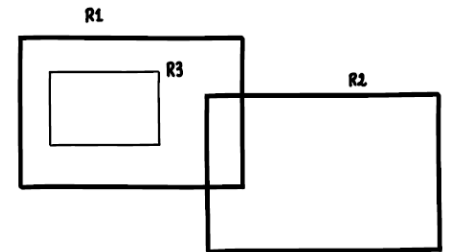
Let's call the left area the zone defined by the intercepted packets only by the rule R3.

Let's call the right area the zone defined by the intercepted packets by the rules R1 and R2.

Given these two areas the administrator may want:

- A) both areas with the same action value, ALLOW or DENY, it does not matter. If the two areas have also the same action as R1 then you can remove R3 because of shadow redundancy.
- B) different action values: if the left area has the same action of R1, then you can remove R3 for shadow redundancy.

Figure 6.22: Case n.97



Let's consider the actions rules and how they can generate or not conflicts.

$R1 = A, R2 = A, R3 = D$: there is just one conflict of type shadow conflict $(R1, R3)$. Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

$R1 = A, R2 = D, R3 = A$: there is just one conflict of type correlation $(R1, R2)$. Thus there is no problem in studying the order of conflict anomalies since there is just one conflict.

$R1 = A, R2 = D, R3 = D$: there is a shadow conflict $(R1, R3)$ and a correlation $(R1, R2)$. In order to obtain:

- case A, then it means that R1 needs to win both the shadow conflict and the correlation, you remove R3 (case A). Or R3 needs to win the shadow conflict, you reorder the rules such that $R3 > R1$ and R2 wins the correlation, you reorder the rules such that $R2 > R1$ (case A).
- case B, then it means that only one between R2 and R3 needs to lose their conflict and the other one has to win it: either R3 loses the shadow conflict, you remove R3 and R2 wins the correlation, you reorder the rules such that $R2 > R1$, or R2 loses the correlation and R3 wins the shadow conflict, you reorder the rules such that $R3 > R1$ (case B).

Let's query the admin in order to see with how many questions we can reach all of these scenarios.

1) Q: "is R1 ok?"

If answer is YES: then it means that both the area have the action value (case A) and this has the same action value of R1.

If answer is NO: then it means that either the two areas have the same action value but this is opposite from R1's or they have different action values. Thus, we need to ask more questions.

2) Q: "is R2 ok?"

If answer is YES: then it means that R2 has won its conflict, but you do not know anything about R3's conflict, it could be either that both areas have different action value from R1 or it could be the case of areas with different actions. Thus, we need to ask more questions.

If answer is NO: then it means that R1 has imposed its action value on the right area, and since R1 is 'wrong', the left area has a different action value from (case B). 3) Q: "is R3 ok?" If answer is YES: then it means that also R3 has won its conflict, therefore it is the case in which both areas have the same action value but this value is different from R1's (case A).

If answer is NO: then it means that R3 has lost its conflict, therefore the right area assumes R1's action value (case B).

\implies Total number of steps: Best Case = 1 Worst Case = 3

Starting querying from R2 or R3 does not bring any advantages since R2 and R3 are disjoint, therefore you would have to query at least twice the admin in the best case, meanwhile the worst case is still equal to 3. After asking for R1, it is equivalent, since they are disjoint, querying first R2 or R3.

Chapter 7

The Proposed Approach

Now that we have studied all the possible cases, let's then sum up the content, address the issue on how to query the admin and try to find some considerations or suggestions that may be helpful during the development of the algorithm.

7.1 Find the Best Rule

From the single case analysis we were able to extract some common patterns that allow us to choose in an optimized way which is the best rule to query first given a set of rules that are related. These are simply suggestions that are used in the function's code 'findBestRuleToQuery' to assign the scores to the rules:

- if there are rules that are related by the dominance relation: when you have to query the admin it is best practice to start from the rule that includes other rules, since therefore it is possible to deduce more information about the configuration.
- if there are rules that related by a single correlation relation then the order in which you query about them does not matter. If instead there is a rule that has more correlation relations with respect to the others then you need to query it first, since in the best case it may solve all of them at the same time.
- it is not recommended, since it is inefficient, start querying from the most punctual rules. This is because they enclose smaller areas of packets, thus even if you are lucky, their scope is limited and they do not allow to have the bigger picture of the conflicts inside the configuration.
- if there are two rules that are related by an equivalence relation, then it is enough to know that one of them is correct in order to remove the other one, both to solve the contradiction conflict and to solve the duplication anomaly. Since in this case it is indifferent which one you choose, it is better to rely on other considerations if present.

- if one rule is related to other rules in a conflicted and non-conflicted way, it is better to consider first those cases in which there are conflicts, since the non-conflicts are not interesting from the study point of view.

Before proceeding with the code, let's focus on a small issue. When we were considering only three rules that are related, then after querying about two of them, does not matter which ones, and receiving a negative response, it means that two out of the three rules are wrong, but this also means that the third one is obviously correct.

This is because of how the administrator works: they may have wrote some rules and put in an incorrect order the rules, but they cannot write all wrong rules, so at some point there must be a correct rule.

This is also why, after $N-1$ negative responses we can skip the last query and assume that the last one is going to be correct, since there must be at least one completely correct rule inside the configuration. So this is easy to check when you only have at most three rules.

When you have at least four rules inside a configuration that are related to each other then there might be some cases in which even knowing that there is one correct rule is not enough.

We need a trick to by-pass this problem. We have came up with three different solutions, each one has its own advantages and disadvantages. Let's compare them and choose one:

- Method A: when you have to solve at least two conflicts between different rules then, if those conflicts do not affect each other (so the rules are linked by a disjoint relation) then you can ask the administrator with a query of type "who wins (Rx,Ry) conflict?" for all the conflict that you have.
 - Advantages:
 - * the query type "who wins (Rx,Ry) conflict?" is very direct, easy to understand which rule wins the conflict.
 - Disadvantages:
 - * the query type "who wins (Rx,Ry) conflict?" is singular, punctual, so in case one rule has multiple conflicts with different rules you need to query multiple times the admin for the same rule.
 - * this method only works if and only if some rules are related by a disjoint relation. In case all the rules inside the configuration were related with each other this would not be possible, because solving one conflict may affect one or more rules and therefore one or more conflicts.

- Method B: once you receive from the administrator a positive response, add new constraints in such a way that the next query will not consider the correct rule (and all the previously correct rules)
 - Advantages:
 - * once you know that a rule is correct you ‘pretend’ is not there anymore, so if a bigger rule is still wrong it must be for some other rules.
 - Disadvantages:
 - * it is only you who ‘pretends’ that the rules is not there anymore, you do not actually remove it, therefore the admin still sees it.
 - * it requires you to enforce some additional constraints and to use a SAT solver.
- Method C: when in doubt, you query the administrator and you ask for the difference of areas of packets.
 - Advantages:
 - * querying only on the difference area then it is possible to solve easily the conflicts, since you know which rules win in that specific area.
 - Disadvantages:
 - * you need to compute and determine the subtraction of areas of packets which is a huge problem computationally. The packet areas are not simply rectangular areas, they are much more complex therefore the subtraction is a complex operation.

Method A only works on some very specific conditions so it is not very useful in a general case, therefore we discard it.

Method C requires a lot of computation and work in order to compute the areas to query, since it is not worth it in our opinion, we discard it.

Hence the winning choice is the method B, it does need a SAT solver in order to work, but it is an acceptable cost with respect to the method C. Therefore, theoretically we use to solve a SAT solver to by-pass the issues.

7.2 How to Query the Administrator

Considering that we are solving the conflicts by configurations means that given N rules that are connected to each other, we can establish a configuration between the rules that can eventually determine some conflicts. If that is the case then you solve this and then you move on to the next N rules.

The first issue that we have to solve is that we have to query to administrator in such a way that it is impossible to have an ambiguous answer, we also need to consider that the query "is R_x ok?" may solve in one shot, depending on the context, more than one conflict.

If instead, you asked "which one wins between (R_1, R_2) ?" or similar queries, the administrator may not know the answer, thus they are not able to provide a correct response and you cannot proceed. The latter type of query is only possible in case there is a third rule which is disjoint from one of the two, then the solution of the conflict does not add any more conflicts.

So, when the administrator responds in a certain way to one of our query, we are able to understand if the answer has solved or not a specific type of conflict and how to act based on that and based on the constraints that are enforced with each answered query.

When we are going to query the administrator we are going to ask them about the correctness of a rule, this means whether or not this rule is correct for all the packets that belong to that rule.

So, for example if R_x has action value ALLOW and we query the admin about this rule: this means asking if all the packets for which you can apply rule R_x have action value equal to ALLOW. If there is at least one packet for which this does not apply, then the administrator will respond with 'NO', otherwise they will respond with 'YES'.

Another way to interpret the query is to consider the rule's priority and how they are affected during the conflict resolution, in this way when you receive a response you may validate or not some constraints on the priorities.

Let's remember that when you are solving conflicts you may remove some rules or reorder them, but you never change nor modify the rules. We suppose that the administrator has correctly written the rules, but they may have been wrongly placed inside the firewall, in the wrong priority order.

Therefore, it is our duty to solve conflicts that may arise due to this wrong order by querying the admin about the correctness of the rules. That is to say that when you ask if R_x is correct you are actually asking (these conditions are in a logic AND):

- knowing that R_x 's priority is lower than the priority of all the other rules that you have asked about and for which you were sure about their positioning.
- if R_x 's priority is higher than the priority of all the other rules that you have not asked about yet.
- if R_x 's priority is higher than the priority of all the other rules that you have asked about but for which you were not sure about their positioning.

If the administrator responds with:

- YES: it means that the previous statements are all correct, so R_x 's priority comes right after the rules that you have already queried and you know that they have an higher priority, but it comes before the rules they you have queried yet and those rules thay you have queried already but for which you do not know the placement in the priority list.
- NO: it means that at least one of the statements is wrong, either there is one rule that you have not queried yet that has an higher priority, or one of the rules the you have already but for which you do not know the placement has an higher priority that R_x .

Let's see in detail an example of this syntax.

Given four rules: $R1 = \text{ALLOW}$, $R2 = \text{ALLOW}$, $R3 = \text{DENY}$, $R4 = \text{DENY}$. These four rules are related with each other as displayed in the Picture 7.2 (ALLOW = green, DENY = red); then the relations between the rules are the following:

- $(R1, R2) = \text{dominance without conflict}$
- $(R1, R3) = \text{dominance with conflict}$
- $(R1, R4) = \text{dominance with conflict}$
- $(R2, R3) = \text{correlation with conflict}$
- $(R2, R4) = \text{correlation with conflict}$
- $(R3, R4) = \text{disjoint}$

Thanks to a specific function, we are able to know which is the best rule to query first. Given a set of rules that are interacting with each other and that are generating some conflicts, there is a special order in which you can ask with the least number of queries the administrator, while at the same time solving all the conflicts inside the configuration.

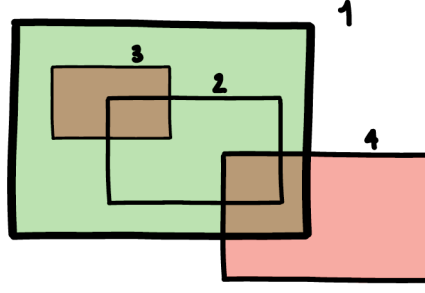


Figure 7.1: Case with 4 Rules - Configuration

This function is able to compute some scores for each rule given the relations with the other rules and their actions and it is able to say which is the *best* rule to query the administrator. In this case best means that it allows us to solve in the best case the greatest number of conflicts given this configuration.

Let's trust this function, thus it returns that the *best* rule to query first is the rule R1. With every query and response from the administrator you can obtain a graph (see Figure 7.2) that allows you to understand all the possible outcomes of this scenario, just by following the branches from one query to the leaves you can trace all the responses that you have received from the administrator and obtain the final and correct order of the rules' priority.

The first query is: "is R1 correct?", which can be translated considering the priorities as: $\pi(R1) > \pi(R2) \wedge \pi(R1) > \pi(R3) \wedge \pi(R1) > \pi(R4)$. Actually, when you are questioning the administrator about these constraints you are also asking "is R1 the first rule in the priority list?".

If the answer is YES, it means that all those constraints are valid, so: $\pi(R1) > \pi(R2) \wedge \pi(R1) > \pi(R3) \wedge \pi(R1) > \pi(R4)$. R1 wins all its conflicts:

- (R1, R3) = R1 wins and R3 is removed (all the relations that contain R3 are also removed).
- (R1, R4) = R1 wins and nothing changes since R4 was at a lower priority level than R1.

If those valid constraints and the results from the conflicts are inserted as input in a SAT solver, you would get as an answer the configuration picture in the top left. Finally, in this case, the correct priority list would be: $\pi(R1) > \pi(R2) > \pi(R4)$.

If the answer is NO, it means that one of those statement is false but you do not know which one, so you write them as: $\pi(R2) > \pi(R1) \vee \pi(R3) > \pi(R1) \vee \pi(R4) > \pi(R1)$. You need to further query the admin in order to find the correct priority order. We call again the previous function and this time, it tells us that the next best rule to query is R2.

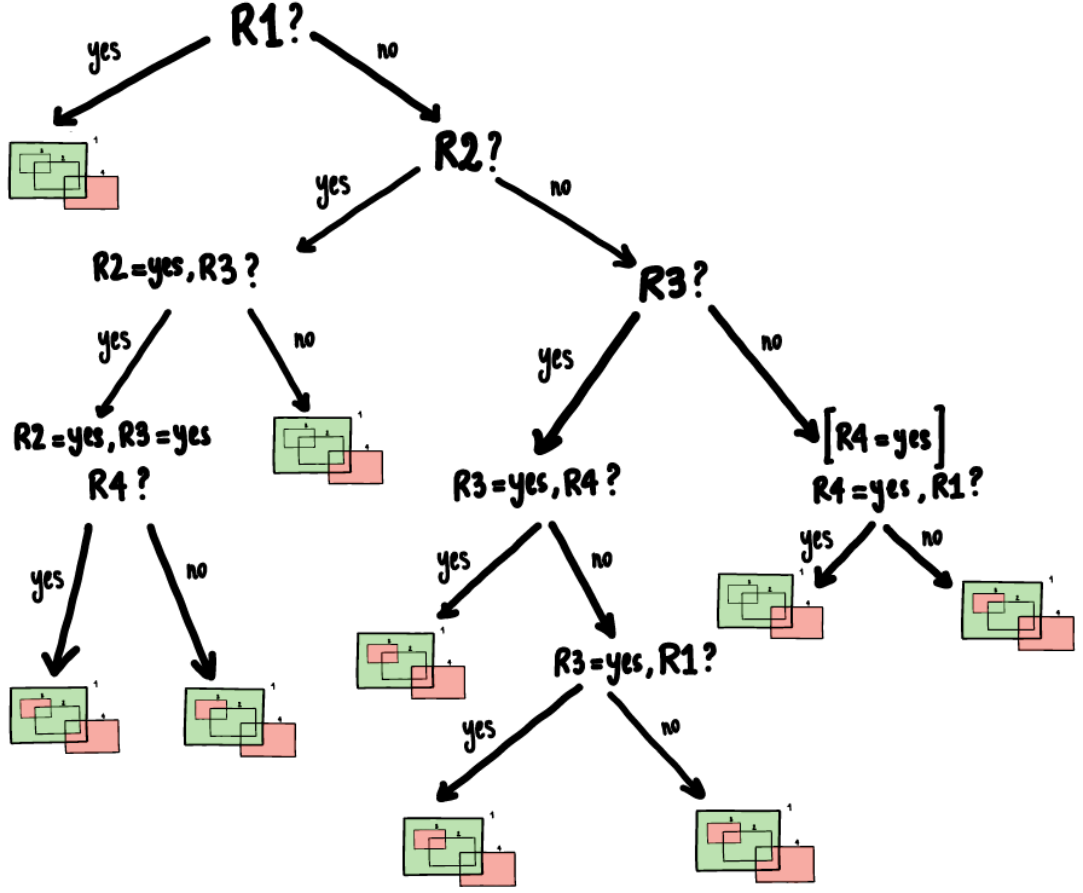


Figure 7.2: Case with 4 Rules - Graph

Then, the second query is: "is R2 correct?", which can be translated considering the priorities as: $\pi(R2) > \pi(R1) \wedge \pi(R2) > \pi(R3) \wedge \pi(R2) > \pi(R4)$. And also, you are asking "is R2 the first rule in the priority list?", since we know that R1 is not the first on the list, we check if R2 is the one.

If the answer is YES, it means that all those constraints are valid, so: $\pi(R2) > \pi(R1) \wedge \pi(R2) > \pi(R3) \wedge \pi(R2) > \pi(R4)$. R2 wins all its conflicts:

- (R2, R3) = R2 wins and nothing changes since R3 was at a lower priority level than R2.
- (R2, R4) = R2 wins and nothing changes since R4 was at a lower priority level than R2.

When you insert the constraints and the results of the conflicts in the SAT solver, it is not able to produce a final solution. We are still missing some information. Therefore we need to query again the administrator, but now we have an additional knowledge that R2 is the first in the list of priorities.

This time the function suggests us to interrogate the admin about R3. The query this time is slightly different since we have some insights: “is R3 correct?” is translated into constraints as:

- knowing that R2 is correct, so that: $\pi(R2) > \pi(R1) \wedge \pi(R2) > \pi(R3) \wedge \pi(R2) > \pi(R4)$ (= R2 is the first in the list)
- is it true that (these conditions are in \wedge):
 - R3’s priority is higher than the priority of all the rules that you have not asked about yet: $\pi(R3) > \pi(R4)$
 - R3’s priority is higher than the priority of all the rules that you have asked already but for which you are not sure the placement: $\pi(R3) > \pi(R1)$

If the answer is YES, it means that all those constraints are valid, so: $\pi(R3) > \pi(R1) \wedge \pi(R3) > \pi(R4)$. R3 wins all its conflicts (the remaining ones):

- (R1, R3) = R3 wins and you reorder the rules such that the constraint is valid.

However in this case there are still some issues and the SAT solver is not able to determine a solution, so we query the admin for the last remaining rule R4. In this query we have even some insights, so that when you ask for: “is R4 correct?” is translated into constraints as:

- knowing that (these conditions are in \wedge):
 - R2 is correct, so that: $\pi(R2) > \pi(R1) \wedge \pi(R2) > \pi(R3) \wedge \pi(R2) > \pi(R4)$ (= R2 is the first in the list)
 - R3 is correct, so that: $\pi(R3) > \pi(R1) \wedge \pi(R3) > \pi(R4)$ (= R3 is the second in the list)
- is it true that (these conditions are in \wedge):
 - R4’s priority is higher than the priority of all the rules that you have not asked about yet: NONE, we have queried about all rules.
 - R4’s priority is higher than the priority of all the rules that you have asked already but for which you are not sure the placement: $\pi(R4) > \pi(R1)$

Since we are only verifying one constraint and therefore solving the corresponding conflict (R1, R4), you are able to reach a solution and the corresponding final order.

In case this last query returns YES, then the final priority order is: $\pi(R2) > \pi(R3) > \pi(R4) > \pi(R1)$.

Otherwise, the final priority order is: $\pi(R2) > \pi(R3) > \pi(R1) > \pi(R4)$.

Instead, if the administrator returns $R3 = \text{NO}$, it means that one of those statement is false but you do not know which one, so you write them as: $\pi(R1) > \pi(R3) \vee \pi(R4) > \pi(R3)$. An additional consideration can be done: since R3 only had one remaining conflict and we received NO, it means that R3 lost that conflict and so the correct order is $\pi(R1) > \pi(R3)$.

However, since $(R1, R3)$ is a shadow conflict in which R1 wins, R3 is removed, therefore you remove also all the relations that contain R3. Now, if you put all the remaining constraints into the SAT solver, it is able to produce a solution, given there is only one possible result. The final order in this case is: $\pi(R2) > \pi(R4) > \pi(R1)$.

In case the second query's ("is R2 correct?") response was NO, it means that one of those statement is false but you do not know which one, so you write them as: $\pi(R1) > \pi(R2) \vee \pi(R3) > \pi(R2) \vee \pi(R4) > \pi(R2)$. You need to further query the admin in order to find the correct priority order. We call again the previous function and this time, it tells us that the next best rule to query is R2.

Then, the third query is: "is R3 correct?", which can be translated considering the priorities as: $\pi(R3) > \pi(R1) \wedge \pi(R3) > \pi(R2) \wedge \pi(R3) > \pi(R4)$. And also, you are asking "is R3 the first rule in the priority list?", since we know that R1 and R2 are not the first on the list, we check if R3 is the one.

If the answer is YES, it means that all those constraints are valid, so: $\pi(R3) > \pi(R1) \wedge \pi(R3) > \pi(R2) \wedge \pi(R3) > \pi(R4)$. R3 wins all its conflicts:

- $(R1, R3) = R3$ wins and you reorder the rules such that the constraint is valid.
- $(R2, R3) = R3$ wins and you reorder the rules such that the constraint is valid.

When you insert the constraints and the results of the conflicts in the SAT solver, it is not able to produce a final solution. We are still missing some information. Therefore we need to query again the administrator, but now we have an additional knowledge that R3 is the first in the list of priorities.

This time the function suggests us to interrogate the admin about R4. So, now the query is slightly different since we have some insights: "is R4 correct?" is translated into constraints as:

- knowing that R3 is correct, so that: $\pi(R3) > \pi(R1) \wedge \pi(R3) > \pi(R2) \wedge \pi(R3) > \pi(R4)$ (= R3 is the first in the list)
- is it true that (these conditions are in \wedge):
 - R4's priority is higher than the priority of all the rules that you have not asked about yet: NONE, we have queried all the previous rules.

- R4's priority is higher than the priority of all the rules that you have asked already but for which you are not sure the placement: $\pi(R4) > \pi(R1) \wedge \pi(R4) > \pi(R2)$

If the answer is YES, it means that all those constraints are valid, so: $\pi(R4) > \pi(R1) \wedge \pi(R4) > \pi(R2)$. R4 wins all its conflicts:

- (R1, R4) = R4 wins and you reorder the rules such that the constraint is valid.
- (R2, R4) = R4 wins and you reorder the rules such that the constraint is valid.

In this case, there are no more conflicts between these rules and the final order is: $\pi(R3) > \pi(R4) > \pi(R1) > \pi(R2)$.

When instead the answer is NO, it means that one of those statement is false but you do not know which one, so you write them as: $\pi(R1) > \pi(R4) \vee \pi(R2) > \pi(R4) \vee \pi(R3) > \pi(R4)$.

Technically, you have queried all the rules, but with respect to the beginning of the querying you now have an additional information: that R3 is correct, so it makes sense to restart from the beginning with querying the administrator and check if with this new information the answer changes (if it is able to solve the conflicts and if the SAT solver is able to produce a final solution). Thus, the next query is: "is R1 correct?", which is translated into constraints as:

- knowing that (these conditions are in \wedge):
 - R3 is correct, so that: $\pi(R3) > \pi(R1) \wedge \pi(R3) > \pi(R2) \wedge \pi(R3) > \pi(R4)$ (= R3 is the first in the list).
- is it true that (these conditions are in \wedge):
 - R1's priority is higher than the priority of all the rules that you have not asked about yet: NONE, we have queried about all rules.
 - R1's priority is higher than the priority of all the rules that you have asked already but for which you are not sure the placement: $\pi(R1) > \pi(R2) \wedge \pi(R1) > \pi(R4)$.

If the answer is YES, it means that all those constraints are valid, so: $\pi(R1) > \pi(R2) \wedge \pi(R1) > \pi(R4)$. R1 wins all its conflicts (better to say, R4 loses all its conflicts):

- (R1, R4) = R1 wins and nothing changes since R4 was at a lower priority level than R1.

- (R2, R4) = R2 wins and nothing changes since R4 was at a lower priority level than R2.

In this case, there are no more conflicts between these rules and the final order is: $\pi(R3) > \pi(R1) > \pi(R2) > \pi(R4)$.

Otherwise, the final priority order is: $\pi(R3) > \pi(R2) > \pi(R4) > \pi(R1)$.

After receiving three (N-1, with N being the total number of rules) negative answers, then you can assume that the fourth and last rule is correct, since there must always be at least one correct rule in the set. This implies that R4 wins its conflicts and that it is the first rule in the priority list: therefore it can be translated into constraints as: $\pi(R4) > \pi(R1) \wedge \pi(R4) > \pi(R2) \wedge \pi(R4) > \pi(R3)$. R4 wins all its conflicts:

- (R1, R4) = R4 wins and you reorder the rules such that the constraint is valid.
- (R2, R4) = R4 wins and you reorder the rules such that the constraint is valid.

Since you have finished all the rules, you restart querying from the beginning providing new information that R4 is correct, hence the next query is: "is R1 correct?", which is translated into constraints as:

- knowing that (these conditions are in \wedge):
 - R4 is correct, so that: $\pi(R4) > \pi(R1) \wedge \pi(R4) > \pi(R2) \wedge \pi(R4) > \pi(R3)$ (= R4 is the first in the list).
- is it true that (these conditions are in \wedge):
 - R1's priority is higher than the priority of all the rules that you have not asked about yet: NONE, we have queried about all rules.
 - R1's priority is higher than the priority of all the rules that you have asked already but for which you are not sure the placement: $\pi(R1) > \pi(R2) \wedge \pi(R1) > \pi(R3)$.

If the answer is YES, it means that all those constraints are valid, so: $\pi(R1) > \pi(R2) \wedge \pi(R1) > \pi(R3)$. R1 wins all its conflicts:

- (R1, R3) = R1 wins and R3 is removed.

In this case, there are no more conflicts between these rules and the final order is: $\pi(R4) > \pi(R1) > \pi(R2)$.

Otherwise, if you give as input the results and the constraints to the SAT solver, it is able to produce the correct final priority order: $\pi(R4) > \pi(R3) > \pi(R2) > \pi(R1)$.

7.3 Algorithm

Given the list of all the rules, you need to generate first the cluster between the rules, this means knowing which of the rules interact with each other. This means that you consider rules till the packet area of each rule create an intersection ($Rx \cap Ry \neq \{\}$).

So, for example: let's imagine that you have already read R1 and R2, and those two area packet intersect with each other, therefore you add them to the same cluster. At this point, you have to read R3, in order to check whether or not R3 is part of the same cluster you verify if $R3 \cap (R1 \cup R2) \neq \{\}$. If so, it means that R3 interacts with the area packet of the cluster so now you add R3 to the same cluster as R1 and R2. Otherwise if the check returns an empty set, R3 is completely disjoint from $Area_{R1} \cup Area_{R2}$, so you 'stop' this cluster before R3.

Then within a cluster, you generate all the possible rule couples and you add them to the the cluster. This is done with a generation function that uses the combinatorial analysis, in particular the combinations of all the rules in pairs.

Once you have generated all the rule couples, each one of them is passed to a function that returns the relation value between the two rules. This function tells you if the two rules inside the couple are related by a relation of equivalence, dominance (eventually inverts the two rules), correlation or disjointness.

The function is also able to alert if inside the rule couple there is a suboptimization anomalies, so that is possible to easily intervene by removing the rule with lower priority and also all the rule couples in which that rule is present.

At this point you need to solve for each cluster all of its conflicts. Till there are no more conflicts or the SAT solver has found a solution perform a loop that consists in:

- find the best query so that you can query the administrator about
- query the administrator about this rule, considering also that you may have some previous knowledge about rules that you have already queried and that are correct
 - if answer is YES: then you have to solve all the conflicts that contain that rule, making that rule the 'winning' rule in the conflict; add the rule to the list of the correct rules; mark done that you have queried this rule and that the answer is yes
 - if answer is NO: then you have to mark done that you have queried this rule and that the answer is no; then check that if you queried about all the rules, then you have to restart from the beginning, so you have to reset some flags; in case you have received N-1 (with N being the total number of rules) no, then it means that the last rule is necessarily

correct, therefore solve all the conflicts of that rule, add it to the list of the correct ones and set the flags as if you queried the admin about this rule, then reset all the necessary flag useful to restart querying the admin with the new information

- check if there are still present some conflicts inside the cluster, if not you can move one to the next cluster
- if instead there are still some conflicts, try using a SAT solver in which you pass the currently known constraints and check whether or not you could still obtain a solution. If, in fact you are able to obtain a correct solution, solve the conflicts thanks to the final order of the priority rules that the SAT solver is able to provide. Otherwise, keep looping till you finish the conflicts or you find a solution.

As we said before, in the implementation we do not use a SAT solver, but in the theoretical algorithm it must be used to check the correctness of the solution before moving into the next cycle.

In our Java code, since we do not use this tool, the algorithm is similar but we can exit the cycle only when we have ended all the conflicts inside the cluster or we are sure that the conflicts inside the cluster are not dangerous (the couples are disjoint or they have the same action).

Therefore it may happen that sometimes you query the administrator more than the minimum since it is not able to understand earlier that all the conflicts are solved, but the algorithm needs more queries to the admin to do that.

Chapter 8

Implementation and Validation

In this chapter we discuss the code implementation, starting from the environment and the IDE that we chose. Then we describe briefly the main classes and the most meaningful functions that we employed.

8.1 Setup and Environment

Eclipse

Eclipse is an Integrated Development Environment (IDE) used for programming. It contains a base workspace and an extensible plug-in system for customizing the environment [5].

Eclipse is mostly used for writing and developing Java applications, but it can also be used with other programming languages through plug-ins such as C, C++, Fortran, JavaScript, Python and many others. The plug-in framework allows the Eclipse Platform to work with typesetting languages like LaTeX and networking applications such as telnet and database management systems [5].



Figure 8.1: Eclipse Logo

The Eclipse Software Development Kit (SDK) is free and open-source, it includes the Java development tools and it is meant for Java developers.

Eclipse uses plug-ins to offer all the functionality within and on top of the run-time system. Its run-time system is based on Equinox.

This architecture based on plug-ins supports writing any desired extension to the environment, such as for configuration management. Java and CVS support is provided in the Eclipse SDK, with support for other version control systems provided by third-party plug-ins. Users can extend its abilities by installing plug-ins

written for the Eclipse Platform, like the development toolkits for other programming languages, and can write and contribute with their own plug-in modules [5].

With the exception of a small run-time kernel, everything in Eclipse is a plug-in. Hence, every plug-in developed integrates with Eclipse in the same way as other plug-ins. This is done in order to follow the principle that: all features are "created equal".

The Eclipse SDK includes the Eclipse Java development tools (JDT), offering an IDE with a built-in Java incremental compiler and a full model of the Java source files. This allows for advanced refactoring techniques and code analysis.

Eclipse implements the graphical control elements of the Java toolkit called Standard Widget Toolkit (SWT). Eclipse's user interface also uses an intermediate graphical user interface layer called JFace, which simplifies the construction of applications based on SWT [5].

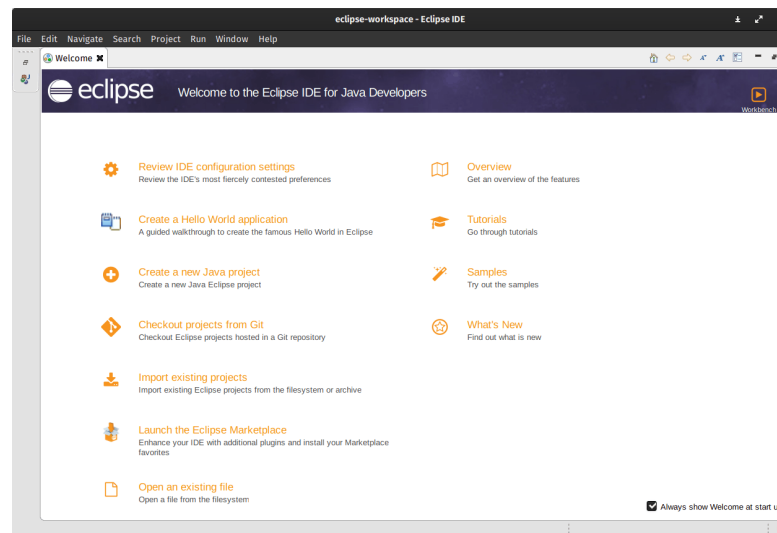


Figure 8.2: Welcome Screen of Eclipse 4.12

Our application was developed in Eclipse version 4.23.0 and in the Java programming language.

XML Schema

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable [12]. XML is designed to have few purposes: simplicity, generality, and usability over the Internet. In order to actually use XML, users need some software to send, receive, store, etc., since XML is only information wrapped in tags [12].

We started our application design from the **XML Schema** that contained all the most important objects that we were going to use in our program. Through this we define the building blocks of an XML document.

The XML Schema has a lot of advantages for example it is able to easily validate the correctness of data and to convert data between different data types. It uses the XML syntax, so you do not have to learn a new language and it is extensible. In order to use an XML document it must be well-formed, thus it has to follow some XML syntax rules, but it may still contain some errors, therefore it has to be validated [11].

```
<xsd:complexType name="RuleType">
  <xsd:sequence>
    <xsd:element name="AreaPacket" type="tns:PacketAreaType"/>
    <xsd:element name="ClusterID" type="xsd:positiveInteger"/>
  </xsd:sequence>
  <xsd:attribute name="rulePriority" type="xsd:positiveInteger"/>
  <xsd:attribute name="action" type="xsd:positiveInteger"/>
  <xsd:attribute name="response" type="xsd:boolean"/>
  <xsd:attribute name="asked" type="xsd:boolean"/>
  <xsd:attribute name="ruleID" type="xsd:positiveInteger" use="required"/>
</xsd:complexType>

<xsd:complexType name="RuleCouple">
  <xsd:sequence>
    <xsd:element name="firstRule" type="tns:RuleType"/>
    <xsd:element name="secondRule" type="tns:RuleType"/>
  </xsd:sequence>
  <xsd:attribute name="coupleRelation" type="xsd:integer"/>
  <xsd:attribute name="sameAction" type="xsd:boolean"/>
  <xsd:attribute name="RuleCoupleID" type="xsd:positiveInteger" use="required"/>
</xsd:complexType>
```

Figure 8.3: XML Schema

JAXB Classes

From the XML Schema we generated the XML document and the corresponding **JAXB classes** (Java Architecture for XML Binding). There are two basic functions or methods when you are dealing with XML and Java: *marshalling* and *unmarshalling*.

The *marshalling* operation is converting JAXB-derived Java objects into an XML data; by default the Marshaller uses UTF-8 encoding when generating XML data. Instead the *unmarshalling* is the reverse operation and it is able to convert XML data into JAXB-derived Java objects.

Once the basic classes are generated, we added to each one the setters, getters and toString() methods, and finally developed inside the main class our application.

SAT Solver

A **SAT solver** is a computer program that has the purpose to solve the Boolean satisfiability problem [13].

As we have noted before, in the algorithm after when queried the administrator and before moving on to the next cycle, we should use a SAT solver to check if we reached to a solution, if the constraints are valid and if there are no more conflicts in the cluster.

In our implementation of the program this is not done, we do not leverage the SAT solver. However, we suppose that when we query the administrator, they are smart enough to not give us a response that will make us incur into an ambiguous situation (since they are knowledgeable and they can see the whole picture). Also, we accept that in some cases there might be more queries than the minimum, since the algorithm has no intelligence to “see the constraints” and check them.

8.2 Main Classes and Data Objects

The most important classes are: *RuleType* that represents a rule, *RuleCouple* which represents a couple of rule with a certain relationship value, and the *Cluster* that can be seen as the single configuration, so the set of the rules that are interacting with each other plus additional information and data. Let’s see them a little more in depth.

RuleType

A *RuleType* is a class (complex type for XML schema) that contains:

- *ruleID* is an identifier which is a positive integer that increases at every new rule.
- *AreaPacket* is the packet area that this rule intercepts (it is the actual object, not a reference).
- *ClusterID* is the cluster identifier that represents the cluster to which the rule is part of; at the beginning it is equal to -1.
- *rule priority* is an integer, starting from 1 as the highest priority, used to understand, given two rules inside the same rule, which is the one that has to be removed/reordered.
- *action* is an integer that represents the action that has to be performed when following this rule.
- *response* it is a flag (boolean) that stores the response from the admin when you query them about this rule; at the beginning it is equal to false.
- *asked* is a flag (boolean) that allows you to understand if the rule has been already queried.

In this work, we consider:

- for the action: 0 = DENY, 1 = ALLOW.
- for the response: false = NO, true = YES.
- for the asked: false = not yet asked, true = already asked/just asked.

In general a PacketArea is the quintuplet the contains IP addresses for source and destination, port addresses and protocol for each rule (there is also an identifier that identifies the object).

This class implements the Comparable interface: in order to sort the elements by the rule priority, given two rules the highest priority as the lowest integer number.

RuleCouple

A *RuleCouple* is a class (complex type for XML schema) that contains:

- *ruleCoupleID* is an identifier which is a positive integer that increases at every new couple.
- *firstRule* is the first rule in the couple (it is the actual object, not a reference).
- *secondRule* is the second rule in the couple (it is the actual object, not a reference).
- *coupleRelation* represents the relation value of the couple with an integer.
- *sameAction* is a flag (boolean) that tells you whether or not the two rules have the same action value.

In this work, we consider:

- for the sameAction: false = the two rules have different actions, true = the two rules have the same action.

The coupleRelation is an integer that is assigned by the function *checkValueRelationFunction*. It can assume 4 different values:

- 0 = equivalence, the two rules inside the couple are equivalent (be careful, they might have different actions).
- 1 = dominance, the first rule includes the second rule.
- 2 = correlation, the two rules are correlated.
- 3 = disjointness, the two rules are disjoint.

This class also includes two extra functions that are able to tell if a RuleCouple contains a specific rule that is passed as a parameter (*containsRule*) and to give us the one rule inside the RuleCouple passing the other one as a parameter (*giveOtherRuleInCouple*).

Cluster

A *Cluster* is a class (complex type for XML schema) that contains:

- *ClusterID* is an identifier which is a positive integer that increases at every new cluster.
- *clusterRules* is the list of all the rules that are part of this cluster.
- *packetArea* is the list of all the packetAreas that are intercepted by the rules inside the cluster.
- *clusterRuleCouples* is the list of all the rule couples that can be generated inside this cluster.

For this class there is no additional methods that we implemented other than setters and getters.

8.3 Important Functions

Let's focus on some of the most meaningful functions: *generate* is function that generates the RuleCouples, *checkValueRelationFunction* assigns the coupleRelation to the RuleCouples, *findBestRuleToQuery* allows us to find the next rule to query the administrator about, *query* is used to query the administrator, *resolveConflicts* is used to solve the conflicts given a winning rule.

Method generation

In order to generate all the possible RuleCouples inside a Cluster given a set of Rules, we used a combinatorial analysis method applied in Java, modifying it with our classes and types.

The *generate* method prepares the ArrayList of RuleCouples and then calls the recursive method *helper* that does actually the job of creating all the RuleCouples.

At the moment of the generation, a RuleCouple contains the two rules, the coupleRelation is set to -1 and depending on the rule's action the sameAction flag is set.

The generation and the helper method are therefore not shown since are not that relevant from the theoretical's point of view, you can look at the code in [Appendix b](#).

Method checkValueRelationFunction

This method tests the relationship of a single RuleCouple and assigns to each one an integer that represents the relation's value. There are 4 possible values that can be assigned that correspond to the four possible relations: equivalence, dominance, correlation and disjoint.

In case of the dominance relation we need to consider this two cases: on one hand the firstRule includes the second one and on the other hand the firstRule is included in the second one. In order to maintain the same syntax, in the latter case we need to swap the two rules inside the RuleCouple.

The check is done by looking at the PacketArea of each rule and verifying if the two have an intersection: more specifically, we only consider the IP addresses of the PacketArea object.

Listing 8.1: checkValueRelationFunction method

```
public static int checkValueRelationFunction(RuleCouple rc)
{
    int res=0;
    RuleType first = rc.getFirstRule();
    IPAddressType sourceIP1 =
        first.getAreaPacket().getSourceIP();
    IPAddressType destIP1 = first.getAreaPacket().getDestIP();
    String proto1 = first.getAreaPacket().getProtocol();

    RuleType second = rc.getSecondRule();
    IPAddressType sourceIP2 =
        second.getAreaPacket().getSourceIP();
    IPAddressType destIP2 = second.getAreaPacket().getDestIP();
    String proto2 = second.getAreaPacket().getProtocol();

    if(sourceIP1.equalFields(sourceIP2) &&
        destIP1.equalFields(destIP2) && proto1.equals(proto2))
        rc.setCoupleRelation(BigInteger.valueOf(0));
    else if(sourceIP1.isIncludedIn(sourceIP2) &&
        destIP1.isIncludedIn(destIP2)) //R1 is included into R2
    {
        rc.setCoupleRelation(BigInteger.valueOf(1));
        RuleType tmp = rc.getFirstRule();
        rc.setFirstRule(rc.getSecondRule());
        rc.setSecondRule(tmp);
    }
    else if(sourceIP2.isIncludedIn(sourceIP1) &&
        destIP2.isIncludedIn(destIP1)) //R2 is included into R1
        rc.setCoupleRelation(BigInteger.valueOf(1));
}
```

```

else if(sourceIP1.isIncludedIn(sourceIP2) ||
        sourceIP2.isIncludedIn(sourceIP1) ||
        destIP1.isIncludedIn(destIP2)
        || destIP2.isIncludedIn(destIP1))
    rc.setCoupleRelation(BigInteger.valueOf(2));
else rc.setCoupleRelation(BigInteger.valueOf(3));

if(rc.getCoupleRelation()==BigInteger.valueOf(0) &&
    rc.isSameAction()==true)
    res=-1;

return res;
}

```

Method findBestRuleToQuery

This method is based on the considerations and suggestions that we extracted from the study of all possible cases (see Chapter 6).

The basic idea is to assign to each rule inside the current cluster points depending on the type of relationships that this rule has with the other rules inside the cluster. This is because, as we have seen previously, some relationships have ‘more value’ than others, so they might generate more conflicts. At the end the rule that has the highest score is selected to be queried by the administrator.

The single scores have been decided based on “trial and error” method: we have tried some values with different configurations until we found the proper ones to make the program work correctly.

Listing 8.2: findBestRuleToQuery method

```

private static RuleType findBestRuleToQuery(List<RuleType>
    clusterRules, List<RuleCouple> clusterRuleCouples)
{
    int i,index=0,max=-10;
    int [] counter = new int[clusterRules.size()];
    Arrays.fill(counter, 0);
    for(i=0;i<clusterRules.size();i++)
    {
        RuleType rule=clusterRules.get(i);
        if(rule.isAsked()==false)
        {
            for(RuleCouple rc: clusterRuleCouples)
            {
                if(rc.containsRule(rule))
                {

```

```

        if(rc.getCoupleRelation()==BigInteger.valueOf(0))
            //contradiction
            counter[i]+=4;
        if(rc.getCoupleRelation()==BigInteger.valueOf(3))
            //disjoint
            counter[i]++;
        if(rc.getCoupleRelation()==BigInteger.valueOf(2))
            //correlation
        {
            if(rc.isSameAction()) //not conflict
                counter[i]+=3;
            else //conflict
                counter[i]+=6;
        }
        if(rc.getCoupleRelation()==BigInteger.valueOf(1))
            //dominance
        {
            if(rc.isSameAction()) //not conflict
            {
                if(rc.getFirstRule()==rule)
                    counter[i]+=4;
                else
                    counter[i]+=2;
            }
            else //conflict
            {
                if(rc.getFirstRule()==rule)
                    counter[i]+=8;
                else
                    counter[i]+=4;
            }
        }
    }
}

for(i=0;i<counter.length;i++)
{
    if(counter[i]>max)
    {
        index=i;
        max=counter[i];
    }
}
return clusterRules.get(index);

```


}

Method query

This method interacts with the administrator in order to receive the response about a rule. In theory after this method the algorithm should invoke a SAT solver to prove that the solution is correct or that we reached a solution, but in our implementation we do not use it.

So, we decided to by-pass this issue by simply printing on the standard output all the current knowledge (all the correct rules that we are aware in the moment of the call to this method) and then query the admin about the rule that we passed as a parameter. Afterwards we take their input as the response assuming 0 means NO and 1 means YES.

Since this method contains only some prints on the standard output, it is not shown here but you can look at the code in [Appendix e](#).

Method resolveConflicts

This method is the one that is called to solve the conflicts inside a cluster. It takes into considerations:

- the rule that you have just queried (*ruleToQuery*), that is the “winner” rule in the conflicts
- the whole list of conflicts that you are considering at the moment (*clusterRuleCouples*)
- the list of all the rules inside the cluster, since you may have to re-order or remove some of them

The biggest issue in this function is that while you are reading the list and solving the conflicts, you are also modifying the list, by removing some of the RuleCouples. This is why you need the *do... while* structure so that you are able to stop and restart after you have deleted all the RuleCouples that you need.

Inside this method there are some points where you call the function *removeRuleCouple*: this is an auxiliary method that has the purpose to remove the couples that contain the rule that is passed as a parameter from the list of all RuleCouples.

Throughout the function there are some prints on the standard output that aims to show the user what is happening and which conflicts are being solved and which ones are still inside the list.

Listing 8.3: resolveConflicts method

```

private static void resolveConflicts(RuleType ruleToQuery,
    List<RuleCouple> clusterRuleCouples, List<RuleType>
    clusterRules)
{
    int i,j, invert=0;
    for(i=0;i<clusterRuleCouples.size();i++)
    {
        RuleCouple couple = clusterRuleCouples.get(i);
        if(couple.containsRule(ruleToQuery)) //only for couple
            that contain ruleToQuery
        {
            RuleType otherRule;
            invert=0;
            System.out.println("SOLVING: (" +
                couple.getFirstRule().getRuleID()+ "," +
                couple.getSecondRule().getRuleID() + ") = " +
                couple.getCoupleRelation());
            do
            {
                otherRule = null;
                otherRule = couple.giveOtherRuleInCouple(ruleToQuery);

                if(couple.getCoupleRelation()==BigInteger.valueOf(0))
                {
                    clusterRules.remove(otherRule);
                    i--;
                    break;
                }
            }
            else
                if(couple.getCoupleRelation()==BigInteger.valueOf(1))
                    //dominance
                {
                    if(ruleToQuery.getRulePriority().intValue()<
                        otherRule.getRulePriority().intValue())
                    {
                        clusterRules.remove(otherRule);
                        i--;
                        break;
                    }
                }
            else //if the winning rule has lower priority you
                need to invert the rules
            {
                invertPriority(ruleToQuery, otherRule,
                    clusterRules);
                clusterRuleCouples.remove(couple);
            }
        }
    }
}

```

```

        invert=1;
        i--;
        break;
    }
}

else
    if(couple.getCoupleRelation()==BigInteger.valueOf(2))
        //correlation
    {
        if(ruleToQuery.getRulePriority().intValue() >
            otherRule.getRulePriority().intValue())
        {
            invertPriority(ruleToQuery, otherRule,
                           clusterRules);
            invert=1;
        }

        clusterRuleCouples.remove(couple);
        i--;
        break;
    }
else
    if(couple.getCoupleRelation()==BigInteger.valueOf(3))
        //disjoint
    {
        clusterRuleCouples.remove(couple);
        i--;
        break;
    }
}while(otherRule!=null);

if(otherRule!=null && couple!=null)
{
    if(couple.getCoupleRelation()==BigInteger.valueOf(0))
        //contradiction
        removeRuleCouple(clusterRuleCouples, otherRule);
    else
        if(couple.getCoupleRelation()==BigInteger.valueOf(1)
            && ruleToQuery.getRulePriority().intValue() <
            otherRule.getRulePriority().intValue() && invert==0)
        {
            removeRuleCouple(clusterRuleCouples, otherRule);
            //this is the case of dominance and the winning
            rule has higher priority
        }
    }
}

```

```
        }
    }

    }

    else
    {
        System.out.println("DO NOT SOLVE: (" +
            couple.getFirstRule().getRuleID() + "," +
            couple.getSecondRule().getRuleID() + ") = " +
            couple.getCoupleRelation());
    }
}

System.out.println("CONFLICTs remaining: ");
if(!clusterRuleCouples.isEmpty())
{ for(RuleCouple couple : clusterRuleCouples)
    System.out.println("(" +
        couple.getFirstRule().getRuleID() + "," +
        couple.getSecondRule().getRuleID() + ") = " +
        couple.getCoupleRelation());
}
else
    System.out.println("NONE");
}
```

8.4 Example with Test Case and Validation

In this part of the chapter, we will see a brief and simple example of how the program works in order to understand it better. Let's first consider a few rules that belong to our firewall that are in conflicts with each other. With this set of rules let's run the application.

Table 8.1: Example of rules inside a Firewall

Rule	Source IP Address	Destination IP Address	Action	Priority
R1	1.1.*.*	1.2.3.*	ALLOW	1
R2	1.1.*.*	1.2.3.*	DENY	2
R3	1.1.2.1	1.2.3.2	ALLOW	3
R4	1.*.*.*	2.5.4.*	DENY	4
R5	3.1.1.*	1.2.3.4	DENY	5
R6	1.5.4.3	2.5.4.1	ALLOW	6
R7	3.1.1.3	3.1.1.1	ALLOW	7
R8	2.1.1.*	2.1.1.1	ALLOW	8
R9	2.1.1.*	2.1.1.2	ALLOW	9
R10	2.1.1.*	2.1.1.3	DENY	10
R11	4.*.*.*	5.5.*.*	ALLOW	11
R12	4.5.*.*	5.*.*.*	DENY	12
R13	4.5.1.1	5.5.1.1	ALLOW	13
R14	4.2.3.1	5.5.3.7	DENY	14

The program will first divide the rules into clusters looking at the intersection of the rule's *PacketAreas*. Based on the IP addresses we will get three clusters, in order to have a better look at them let's use a graphical representation of each of them.

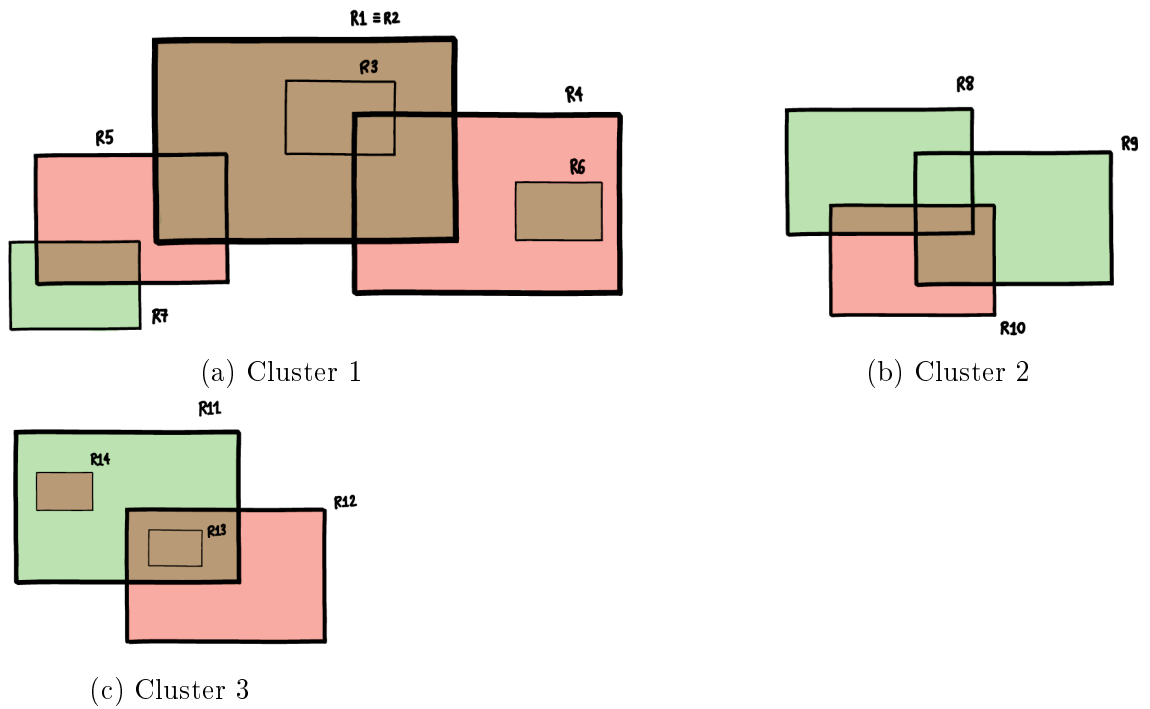
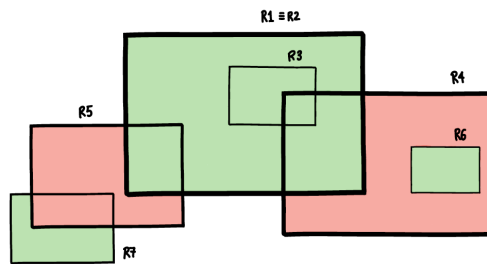


Figure 8.4: Graphical Representation of the Clusters

The program will cycle the cluster list, considering one cluster of rules at the time since different clusters do not affect each others in terms of conflicts. In this example we will only focus on Cluster 1, but the same principle can be applied to the other two.

Now, let's imagine that when the administrator responds to our queries, they have in mind the following picture as the solution of the cluster's conflicts:

Figure 8.5: Solution of Cluster 1



After having generated the clusters, the program creates for each cluster all the possible *RuleCouples* that can generated given the set of rules that belong to the cluster. In our case of Cluster 1, since we have 7 rules we have a total of 21 (following the combination without repetition formula).

Table 8.2: RuleCouples generated in Cluster1

(R1,R2)	(R1,R3)	(R1,R4)	(R1,R5)	(R1,R6)	(R1,R7)	(R2,R3)
(R2,R4)	(R2,R5)	(R2,R6)	(R2,R7)	(R3,R4)	(R3,R5)	(R3,R6)
(R3,R7)	(R4,R5)	(R4,R6)	(R4,R7)	(R5,R6)	(R5,R7)	(R6,R7)

Since none of them generate a sub-optimization anomaly of type duplication, we have to consider all of them. In the next phase of the application, we have to assign to each of the *RuleCouples* the relation value through the function *check-ValueRelationFunction*. So now, the RuleCouples look like this:

Table 8.3: RuleCouples with Relation Values in Cluster1

(R1,R2) = 0	(R1,R3) = 1	(R1,R4) = 2	(R1,R5) = 2	(R1,R6) = 3
(R1,R7) = 3	(R2,R3) = 1	(R2,R4) = 2	(R2,R5) = 2	(R2,R6) = 3
(R2,R7) = 3	(R3,R4) = 2	(R3,R5) = 3	(R3,R6) = 3	(R3,R7) = 3
(R4,R5) = 3	(R4,R6) = 1	(R4,R7) = 3	(R5,R6) = 3	(R5,R7) = 2
(R6,R7) = 3				

In Table 8.3 we have also highlighted in red, those couple that are actually in conflict. These are the conflict that we have to solve before moving on to the next cluster.

In the next part of the application, we will have to solve the conflicts by querying the administrator about the rules that are part of this cluster. Thanks to the method *findBestRuleToQuery*, we are able to find the rule that is the most advantageous to query. In our case the scores that are assigned by the method to each rule are:

- $R1 = 4 + 4 + 6 + 6 + 1 + 1 = 22$
- $R2 = 4 + 8 + 3 + 3 + 1 + 1 = 20$
- $R3 = 2 + 4 + 6 + 1 + 1 + 1 = 15$
- $R4 = 6 + 3 + 6 + 1 + 8 + 1 = 25$
- $R5 = 6 + 3 + 1 + 1 + 1 + 6 = 18$
- $R6 = 1 + 1 + 1 + 4 + 1 + 1 = 9$

- $R7 = 1 + 1 + 1 + 1 + 6 + 1 = 11$

In the first step, the winner rule is R4, therefore we query the administrator about it.

Knowing that the solution for this cluster is Figure 8.5, the administrator tells us that R4 is wrong: this means that there is something inside the packet area defined by R4 that does not follow R4's action, which is DENY. Since we are not able to draw any conclusions about any conflicts, the list of the remaining conflicts is the same, we just set for R4 the flags *asked* and *response*, so that we will skip and not query again this rule in the next cycle.

Since there are still conflicts remaining, we run again the method *findBestRuleToQuery* to find another rule: this time the function returns R1. When queried, the administrator responds that R1 is indeed correct (see Figure 8.5), so now the algorithm will solve the conflicts.

First thing to do is to set R1's the flags *asked* and *response*, so that we will skip and not query again this rule in the next cycle. Then the program calls the method *resolveConflicts*: now all the conflicts that contain R1 will be solved by making R1 the winner rule.

In our case, we have to solve:

- $(R1, R2) = 0 \rightarrow$ this is a contradiction conflict: R2 is removed. You also remove all the RuleCouples that contain R2 (this is done by *removeRuleCouple*).
- $(R1, R3) = 1 \rightarrow$ this is a sub-optimization anomalies of type shadow redundancy: R3 is removed. You also remove all the RuleCouples that contain R3 (this is done by *removeRuleCouple*).
- $(R1, R4) = 2 \rightarrow$ this is a correlation conflict: nothing changes, since R1 has already higher priority than R4.
- $(R1, R5) = 2 \rightarrow$ this is a correlation conflict: nothing changes, since R1 has already higher priority than R5.
- $(R1, R6) = 3 \rightarrow$ this is removed since they do not interact.
- $(R1, R7) = 3 \rightarrow$ this is removed since they do not interact.

After solving all those conflicts, the current situation in the Cluster 1 is the following:

Table 8.4: Rules in the Cluster1 After Querying R1

Rule	Source IP Address	Destination IP Address	Action	Priority
R1	1.1.*.*	1.2.3.*	ALLOW	1
R4	1.*.*.*	2.5.4.*	DENY	4
R5	3.1.1.*	1.2.3.4	DENY	5
R6	1.5.4.3	2.5.4.1	ALLOW	6
R7	3.1.1.3	3.1.1.1	ALLOW	7

Table 8.5: RuleCouples in the Cluster1 After Querying R1

(R4,R5) = 3	(R4,R6) = 1	(R4,R7) = 3
(R5,R6) = 3	(R5,R7) = 2	(R6,R7) = 3

We still have some conflicts inside the cluster, therefore we need to keep query the administrator once again. The *findBestRuleToQuery* method suggests to interrogate about R5. So now, knowing that R1 is correct, hence it should not be “considered” anymore, R5 is stated to be correct. Hence, after setting R5’s *asked* and *response*, we solve its conflicts:

- (R4, R5) = 3 \rightarrow this is removed since they do not interact.
- (R5, R6) = 3 \rightarrow this is removed since they do not interact.
- (R5, R7) = 2 \rightarrow this is a correlation conflict: nothing changes, since R5 has already higher priority than R7.

Subsequently to this step, the number of rules inside the Cluster1 is the same as before and the remaining RuleCouples are:

We still have one more conflict: (R4,R6), the other two *RuleCouples* do not actually count, since they have *coupleRelation* equal to 3 which means they are disjoint. The program also consider the case in which the only remaining RuleCouples are disjoint, in that case a flag is set and we can exit from the cycle and switch to the next cluster.

Finally the administrator is asked about R6, following the solution Figure 8.5, then they answer that R6 is correct. You first set the usual R6’s flags and then you call the *resolveConflicts* method.

- (R4, R6) = 1 \rightarrow this is a shadow conflict: R6 has a lower priority than R4, so we invert the two rules.
- (R6, R7) = 3 \rightarrow this is removed since they do not interact.

Table 8.6: RuleCouples in the Cluster1 After Querying R5

$(R4, R6) = 1$	$(R4, R7) = 3$	$(R6, R7) = 3$
----------------	----------------	----------------

Table 8.7: RuleCouples in the Cluster1 After Querying R6

$(R4, R7) = 3$

As said before we check that even if the list of RuleCouples of Cluster1 is not empty, the RuleCouples that are present do not create any conflict. In this case the only remaining couple (R4, R7) has a relationship value equal to 3, they are disjoint, therefore it is safe to exit from the cycle and to proclaim that this cluster is free from conflicts.

The same argument can be done for the other two clusters and also for other cases of firewalls in different configurations of rules.

Just pay attention that this implementation has no “intelligence” and cannot see nor verify the constraints between priorities. In order to do that we would need a SAT solver.

Chapter 9

Conclusion

As we have stated before, in literature there were some attempts to solve the issue of conflicts inside a firewall but, one group of solutions were only detecting the conflicts [2], the other one tried to solve the anomalies inside the firewall, but in a completely automatic way[3], which it may not be the best thing to do, since you may not follow the administrator's design.

Throughout this study, we understood why it is not possible to automatize the process of solving conflicts and why we need the help of the administrator. Even though we may choose some best practise the last word has to be the administrator's one.

For these reasons, this project aimed to look for an optimized algorithm to query the administrator when there are conflicts inside a firewall. Such that, the number of queries that are going to be made to the administrator are the minimum and the simplest way possible.

In order to do that, we first started to study the general theoretical problem on how different rules interact with each other in a firewall and which are all the possible conflicts that may arise depending on the rule's actions. From this we extracted some possible approaches that were helpful to determine the algorithm.

Afterwards, we had to face the issue on how to interact with the administrator in order to query them in the "simplest" way possible. Once it was done, we developed an algorithm that satisfies these conditions.

In the theoretical algorithm we used a SAT solver to bypass the issue of some conflicts that may not be solved directly but thanks to the constraints applied we know that they are already solved or correct. This means, that we interrogate the administrator with the minimum number of queries. The constraints are forced inside the SAT solver and it assures us the correctness of the solution, therefore we can exit from the cycle prematurely.

However in the implementation we avoided to use a SAT solver, instead we developed only in Java language. This is a limitation but we accept that in some cases we might not reach to the solution in the minimum number of queries and we may need one more query to reach the final solution.

Another limitation is that we did not use efficient methods for the generation of the *RuleCouples*, rather we could have adopted some library functions that are specifically made for combinatorial analysis.

As for future development related in this field, the obvious direction is to implement the algorithm with a SAT solver so that we can actually achieve the minimum number of queries all the time and not in just specific cases.

Also, we defined the rules in a static way, not using a database: so, another further application would be to add databases. A further development would be to implement REST APIs in order to interact with administrator, since in our project we just use some prints on the standard output.

Bibliography

- [1] F. Valenza and M. Cheminod, “An optimized firewall anomaly resolution,” *J. Internet Serv. Inf. Secur.*, vol. 10, no. 1, pp. 22–37, 2020. [Online]. Available: <https://doi.org/10.22667/JISIS.2020.02.29.022>
- [2] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan, “Conflict classification and analysis of distributed firewall policies,” *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 10, pp. 2069–2084, 2005.
- [3] H. Hu, G.-J. Ahn, and K. Kulkarni, “Detecting and resolving firewall policy anomalies,” *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 3, pp. 318–331, 2012.
- [4] M. Cheminod, L. Durante, L. Seno, and A. Valenzano, “An algorithm for security policy migration in multiple firewall networks,” in *Proceedings of the Italian Conference on Cybersecurity, ITASEC 2021, All Digital Event, April 7-9, 2021*, ser. CEUR Workshop Proceedings, A. Armando and M. Colajanni, Eds., vol. 2940. CEUR-WS.org, 2021, pp. 344–359. [Online]. Available: <http://ceur-ws.org/Vol-2940/paper29.pdf>
- [5] Wikipedia, “Eclipse (software).” [Online]. Available: [https://en.wikipedia.org/wiki/Eclipse_\(software\)](https://en.wikipedia.org/wiki/Eclipse_(software))
- [6] S. Security, “The 5 different types of firewalls explained.” [Online]. Available: <https://www.techtarget.com/searchsecurity/feature/The-five-different-types-of-firewalls>
- [7] Compuquip, “What is a firewall? the different firewall types & architectures.” [Online]. Available: <https://www.compuquip.com/blog/types-firewall-architectures>
- [8] JavaTPoint, “Types of firewall.” [Online]. Available: <https://www.javatpoint.com/types-of-firewall>
- [9] GeeksforGeeks, “Types of network firewall.” [Online]. Available: <https://www.geeksforgeeks.org/types-of-network-firewall>
- [10] spiceworks, “What is a firewall? definition, key components, and best practices.” [Online]. Available: <https://www.spiceworks.com/it-security/web-security/articles/what-is-firewall-definition-key-components-best-practices/>

- [11] W3Schools, “Xml schema - intro.” [Online]. Available: https://www.w3schools.com/xml/schema_intro.asp
- [12] GeeksforGeeks, “Xml basic.” [Online]. Available: [https://www.geeksforgeeks.org/xml-basics/#:~:text=Extensible%20Markup%20Language%20\(XML\)%20is,and%20usability%20across%20the%20Internet.](https://www.geeksforgeeks.org/xml-basics/#:~:text=Extensible%20Markup%20Language%20(XML)%20is,and%20usability%20across%20the%20Internet.)
- [13] Wikipedia, “Sat solver.” [Online]. Available: https://en.wikipedia.org/wiki/SAT_solver
- [14] F. Valenza, S. Spinoso, C. Basile, R. Sisto, and A. Liroy, “A formal model of network policy analysis,” in *2015 IEEE 1st International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*, 2015, pp. 516–522.
- [15] D. Bringhenti, G. Marchetto, R. Sisto, and F. Valenza, “Short paper: Automatic configuration for an optimal channel protection in virtualized networks,” in *Proceedings of the 2nd Workshop on Cyber-Security Arms Race*, ser. CYSARM’20. New York, NY, USA: Association for Computing Machinery, 2020, p. 25–30. [Online]. Available: <https://doi.org/10.1145/3411505.3418439>
- [16] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, “Automated firewall configuration in virtual networks,” *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2022.
- [17] F. Valenza, S. Spinoso, and R. Sisto, “Formally specifying and checking policies and anomalies in service function chaining,” *Journal of Network and Computer Applications*, vol. 146, p. 102419, 2019. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S108480451930253X>
- [18] D. Bringhenti and F. Valenza, “Optimizing distributed firewall reconfiguration transients,” *Computer Networks*, vol. 215, p. 109183, 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S138912862200281X>
- [19] D. Bringhenti, G. Marchetto, R. Sisto, S. Spinoso, F. Valenza, and J. Yusupov, “Improving the formal verification of reachability policies in virtualized networks,” *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 713–728, 2021.
- [20] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, “Towards a fully automated and optimized network security functions orchestration,” in *2019 4th International Conference on Computing, Communications and Security (ICCCS)*, 2019, pp. 1–7.
- [21] F. Valenza, C. Basile, D. Canavese, and A. Liroy, “Classification and analysis of communication protection policy anomalies,” *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2601–2614, 2017.

- [22] C. Basile, D. Canavese, A. Lioy, and F. Valenza, “Inter-technology conflict analysis for communication protection policies,” in *Risks and Security of Internet and Systems*, J. Lopez, I. Ray, and B. Crispo, Eds. Cham: Springer International Publishing, 2015, pp. 148–163.
- [23] E. Karafili, F. Valenza, Y. Chen, and E. C. Lupu, “Towards a framework for automatic firewalls configuration via argumentation reasoning,” in *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, 2020, pp. 1–4.

Appendices

Appendix A

In this appendix we will show the list of all possible cases that was used in Chapter 6, both at the beginning of the study and also at the end of the considerations.

a Initial List

1. $(R1,R2) = E \text{ --- } (R2,R3) = E \text{ --- } (R1,R3) = E$
2. $(R1,R2) = E \text{ --- } (R2,R3) = E \text{ --- } (R1,R3) = DM$
3. $(R1,R2) = E \text{ --- } (R2,R3) = E \text{ --- } (R3,R1) = DM$
4. $(R1,R2) = E \text{ --- } (R2,R3) = E \text{ --- } (R1,R3) = C$
5. $(R1,R2) = E \text{ --- } (R2,R3) = E \text{ --- } (R1,R3) = DJ$
6. $(R1,R2) = E \text{ --- } (R2,R3) = DM \text{ --- } (R1,R3) = E$
7. $(R1,R2) = E \text{ --- } (R2,R3) = DM \text{ --- } (R1,R3) = DM$
8. $(R1,R2) = E \text{ --- } (R2,R3) = DM \text{ --- } (R3,R1) = DM$
9. $(R1,R2) = E \text{ --- } (R2,R3) = DM \text{ --- } (R1,R3) = C$
10. $(R1,R2) = E \text{ --- } (R2,R3) = DM \text{ --- } (R1,R3) = DJ$
11. $(R1,R2) = E \text{ --- } (R3,R2) = DM \text{ --- } (R1,R3) = E$
12. $(R1,R2) = E \text{ --- } (R3,R2) = DM \text{ --- } (R1,R3) = DM$
13. $(R1,R2) = E \text{ --- } (R3,R2) = DM \text{ --- } (R3,R1) = DM$
14. $(R1,R2) = E \text{ --- } (R3,R2) = DM \text{ --- } (R1,R3) = C$
15. $(R1,R2) = E \text{ --- } (R3,R2) = DM \text{ --- } (R1,R3) = DJ$
16. $(R1,R2) = E \text{ --- } (R2,R3) = C \text{ --- } (R1,R3) = E$
17. $(R1,R2) = E \text{ --- } (R2,R3) = C \text{ --- } (R1,R3) = DM$
18. $(R1,R2) = E \text{ --- } (R2,R3) = C \text{ --- } (R3,R1) = DM$

19. $(R1,R2) = E \text{ --- } (R2,R3) = C \text{ --- } (R1,R3) = C$
20. $(R1,R2) = E \text{ --- } (R2,R3) = C \text{ --- } (R1,R3) = DJ$
21. $(R1,R2) = E \text{ --- } (R2,R3) = DJ \text{ --- } (R1,R3) = E$
22. $(R1,R2) = E \text{ --- } (R2,R3) = DJ \text{ --- } (R1,R3) = DM$
23. $(R1,R2) = E \text{ --- } (R2,R3) = DJ \text{ --- } (R3,R1) = DM$
24. $(R1,R2) = E \text{ --- } (R2,R3) = DJ \text{ --- } (R1,R3) = C$
25. $(R1,R2) = E \text{ --- } (R2,R3) = DJ \text{ --- } (R1,R3) = DJ$
26. $(R1,R2) = DM \text{ --- } (R2,R3) = E \text{ --- } (R1,R3) = E$
27. $(R1,R2) = DM \text{ --- } (R2,R3) = E \text{ --- } (R1,R3) = DM$
28. $(R1,R2) = DM \text{ --- } (R2,R3) = E \text{ --- } (R3,R1) = DM$
29. $(R1,R2) = DM \text{ --- } (R2,R3) = E \text{ --- } (R1,R3) = C$
30. $(R1,R2) = DM \text{ --- } (R2,R3) = E \text{ --- } (R1,R3) = DJ$
31. $(R1,R2) = DM \text{ --- } (R2,R3) = DM \text{ --- } (R1,R3) = E$
32. $(R1,R2) = DM \text{ --- } (R2,R3) = DM \text{ --- } (R1,R3) = DM$
33. $(R1,R2) = DM \text{ --- } (R2,R3) = DM \text{ --- } (R3,R1) = DM$
34. $(R1,R2) = DM \text{ --- } (R2,R3) = DM \text{ --- } (R1,R3) = C$
35. $(R1,R2) = DM \text{ --- } (R2,R3) = DM \text{ --- } (R1,R3) = DJ$
36. $(R1,R2) = DM \text{ --- } (R3,R2) = DM \text{ --- } (R1,R3) = E$
37. $(R1,R2) = DM \text{ --- } (R3,R2) = DM \text{ --- } (R1,R3) = DM$
38. $(R1,R2) = DM \text{ --- } (R3,R2) = DM \text{ --- } (R3,R1) = DM$
39. $(R1,R2) = DM \text{ --- } (R3,R2) = DM \text{ --- } (R1,R3) = C$
40. $(R1,R2) = DM \text{ --- } (R3,R2) = DM \text{ --- } (R1,R3) = DJ$
41. $(R1,R2) = DM \text{ --- } (R2,R3) = C \text{ --- } (R1,R3) = E$
42. $(R1,R2) = DM \text{ --- } (R2,R3) = C \text{ --- } (R1,R3) = DM$
43. $(R1,R2) = DM \text{ --- } (R2,R3) = C \text{ --- } (R3,R1) = DM$
44. $(R1,R2) = DM \text{ --- } (R2,R3) = C \text{ --- } (R1,R3) = C$
45. $(R1,R2) = DM \text{ --- } (R2,R3) = C \text{ --- } (R1,R3) = DJ$
46. $(R1,R2) = DM \text{ --- } (R2,R3) = DJ \text{ --- } (R1,R3) = E$

47. $(R1,R2) = DM \text{ --- } (R2,R3) = DJ \text{ --- } (R1,R3) = DM$
48. $(R1,R2) = DM \text{ --- } (R2,R3) = DJ \text{ --- } (R3,R1) = DM$
49. $(R1,R2) = DM \text{ --- } (R2,R3) = DJ \text{ --- } (R1,R3) = C$
50. $(R1,R2) = DM \text{ --- } (R2,R3) = DJ \text{ --- } (R1,R3) = DJ$
51. $(R2,R1) = DM \text{ --- } (R2,R3) = E \text{ --- } (R1,R3) = E$
52. $(R2,R1) = DM \text{ --- } (R2,R3) = E \text{ --- } (R1,R3) = DM$
53. $(R2,R1) = DM \text{ --- } (R2,R3) = E \text{ --- } (R3,R1) = DM$
54. $(R2,R1) = DM \text{ --- } (R2,R3) = E \text{ --- } (R1,R3) = C$
55. $(R2,R1) = DM \text{ --- } (R2,R3) = E \text{ --- } (R1,R3) = DJ$
56. $(R2,R1) = DM \text{ --- } (R2,R3) = DM \text{ --- } (R1,R3) = E$
57. $(R2,R1) = DM \text{ --- } (R2,R3) = DM \text{ --- } (R1,R3) = DM$
58. $(R2,R1) = DM \text{ --- } (R2,R3) = DM \text{ --- } (R3,R1) = DM$
59. $(R2,R1) = DM \text{ --- } (R2,R3) = DM \text{ --- } (R1,R3) = C$
60. $(R2,R1) = DM \text{ --- } (R2,R3) = DM \text{ --- } (R1,R3) = DJ$
61. $(R2,R1) = DM \text{ --- } (R3,R2) = DM \text{ --- } (R1,R3) = E$
62. $(R2,R1) = DM \text{ --- } (R3,R2) = DM \text{ --- } (R1,R3) = DM$
63. $(R2,R1) = DM \text{ --- } (R3,R2) = DM \text{ --- } (R3,R1) = DM$
64. $(R2,R1) = DM \text{ --- } (R3,R2) = DM \text{ --- } (R1,R3) = C$
65. $(R2,R1) = DM \text{ --- } (R3,R2) = DM \text{ --- } (R1,R3) = DJ$
66. $(R2,R1) = DM \text{ --- } (R2,R3) = C \text{ --- } (R1,R3) = E$
67. $(R2,R1) = DM \text{ --- } (R2,R3) = C \text{ --- } (R1,R3) = DM$
68. $(R2,R1) = DM \text{ --- } (R2,R3) = C \text{ --- } (R3,R1) = DM$
69. $(R2,R1) = DM \text{ --- } (R2,R3) = C \text{ --- } (R1,R3) = C$
70. $(R2,R1) = DM \text{ --- } (R2,R3) = C \text{ --- } (R1,R3) = DJ$
71. $(R2,R1) = DM \text{ --- } (R2,R3) = DJ \text{ --- } (R1,R3) = E$
72. $(R2,R1) = DM \text{ --- } (R2,R3) = DJ \text{ --- } (R1,R3) = DM$
73. $(R2,R1) = DM \text{ --- } (R2,R3) = DJ \text{ --- } (R3,R1) = DM$
74. $(R2,R1) = DM \text{ --- } (R2,R3) = DJ \text{ --- } (R1,R3) = C$

75. $(R2,R1) = DM \text{ --- } (R2,R3) = DJ \text{ --- } (R1,R3) = DJ$
76. $(R1,R2) = C \text{ --- } (R2,R3) = E \text{ --- } (R1,R3) = E$
77. $(R1,R2) = C \text{ --- } (R2,R3) = E \text{ --- } (R1,R3) = DM$
78. $(R1,R2) = C \text{ --- } (R2,R3) = E \text{ --- } (R3,R1) = DM$
79. $(R1,R2) = C \text{ --- } (R2,R3) = E \text{ --- } (R1,R3) = C$
80. $(R1,R2) = C \text{ --- } (R2,R3) = E \text{ --- } (R1,R3) = DJ$
81. $(R1,R2) = C \text{ --- } (R2,R3) = DM \text{ --- } (R1,R3) = E$
82. $(R1,R2) = C \text{ --- } (R2,R3) = DM \text{ --- } (R1,R3) = DM$
83. $(R1,R2) = C \text{ --- } (R2,R3) = DM \text{ --- } (R3,R1) = DM$
84. $(R1,R2) = C \text{ --- } (R2,R3) = DM \text{ --- } (R1,R3) = C$
85. $(R1,R2) = C \text{ --- } (R2,R3) = DM \text{ --- } (R1,R3) = DJ$
86. $(R1,R2) = C \text{ --- } (R3,R2) = DM \text{ --- } (R1,R3) = E$
87. $(R1,R2) = C \text{ --- } (R3,R2) = DM \text{ --- } (R1,R3) = DM$
88. $(R1,R2) = C \text{ --- } (R3,R2) = DM \text{ --- } (R3,R1) = DM$
89. $(R1,R2) = C \text{ --- } (R3,R2) = DM \text{ --- } (R1,R3) = C$
90. $(R1,R2) = C \text{ --- } (R3,R2) = DM \text{ --- } (R1,R3) = DJ$
91. $(R1,R2) = C \text{ --- } (R2,R3) = C \text{ --- } (R1,R3) = E$
92. $(R1,R2) = C \text{ --- } (R2,R3) = C \text{ --- } (R1,R3) = DM$
93. $(R1,R2) = C \text{ --- } (R2,R3) = C \text{ --- } (R3,R1) = DM$
94. $(R1,R2) = C \text{ --- } (R2,R3) = C \text{ --- } (R1,R3) = C$
95. $(R1,R2) = C \text{ --- } (R2,R3) = C \text{ --- } (R1,R3) = DJ$
96. $(R1,R2) = C \text{ --- } (R2,R3) = DJ \text{ --- } (R1,R3) = E$
97. $(R1,R2) = C \text{ --- } (R2,R3) = DJ \text{ --- } (R1,R3) = DM$
98. $(R1,R2) = C \text{ --- } (R2,R3) = DJ \text{ --- } (R3,R1) = DM$
99. $(R1,R2) = C \text{ --- } (R2,R3) = DJ \text{ --- } (R1,R3) = C$
100. $(R1,R2) = C \text{ --- } (R2,R3) = DJ \text{ --- } (R1,R3) = DJ$
101. $(R1,R2) = DJ \text{ --- } (R2,R3) = E \text{ --- } (R1,R3) = E$
102. $(R1,R2) = DJ \text{ --- } (R2,R3) = E \text{ --- } (R1,R3) = DM$

- 103. $(R1,R2) = DJ \text{ --- } (R2,R3) = E \text{ --- } (R3,R1) = DM$
- 104. $(R1,R2) = DJ \text{ --- } (R2,R3) = E \text{ --- } (R1,R3) = C$
- 105. $(R1,R2) = DJ \text{ --- } (R2,R3) = E \text{ --- } (R1,R3) = DJ$
- 106. $(R1,R2) = DJ \text{ --- } (R2,R3) = DM \text{ --- } (R1,R3) = E$
- 107. $(R1,R2) = DJ \text{ --- } (R2,R3) = DM \text{ --- } (R1,R3) = DM$
- 108. $(R1,R2) = DJ \text{ --- } (R2,R3) = DM \text{ --- } (R3,R1) = DM$
- 109. $(R1,R2) = DJ \text{ --- } (R2,R3) = DM \text{ --- } (R1,R3) = C$
- 110. $(R1,R2) = DJ \text{ --- } (R2,R3) = DM \text{ --- } (R1,R3) = DJ$
- 111. $(R1,R2) = DJ \text{ --- } (R3,R2) = DM \text{ --- } (R1,R3) = E$
- 112. $(R1,R2) = DJ \text{ --- } (R3,R2) = DM \text{ --- } (R1,R3) = DM$
- 113. $(R1,R2) = DJ \text{ --- } (R3,R2) = DM \text{ --- } (R3,R1) = DM$
- 114. $(R1,R2) = DJ \text{ --- } (R3,R2) = DM \text{ --- } (R1,R3) = C$
- 115. $(R1,R2) = DJ \text{ --- } (R3,R2) = DM \text{ --- } (R1,R3) = DJ$
- 116. $(R1,R2) = DJ \text{ --- } (R2,R3) = C \text{ --- } (R1,R3) = E$
- 117. $(R1,R2) = DJ \text{ --- } (R2,R3) = C \text{ --- } (R1,R3) = DM$
- 118. $(R1,R2) = DJ \text{ --- } (R2,R3) = C \text{ --- } (R3,R1) = DM$
- 119. $(R1,R2) = DJ \text{ --- } (R2,R3) = C \text{ --- } (R1,R3) = C$
- 120. $(R1,R2) = DJ \text{ --- } (R2,R3) = C \text{ --- } (R1,R3) = DJ$
- 121. $(R1,R2) = DJ \text{ --- } (R2,R3) = DJ \text{ --- } (R1,R3) = E$
- 122. $(R1,R2) = DJ \text{ --- } (R2,R3) = DJ \text{ --- } (R1,R3) = DM$
- 123. $(R1,R2) = DJ \text{ --- } (R2,R3) = DJ \text{ --- } (R3,R1) = DM$
- 124. $(R1,R2) = DJ \text{ --- } (R2,R3) = DJ \text{ --- } (R1,R3) = C$
- 125. $(R1,R2) = DJ \text{ --- } (R2,R3) = DJ \text{ --- } (R1,R3) = DJ$

b Final List

- 7. $(R1,R2) = E \text{ --- } (R2,R3) = DM \text{ --- } (R1,R3) = DM$
- 19. $(R1,R2) = E \text{ --- } (R2,R3) = C \text{ --- } (R1,R3) = C$
- 27. $(R1,R2) = DM \text{ --- } (R2,R3) = E \text{ --- } (R1,R3) = DM$
- 32. $(R1,R2) = DM \text{ --- } (R2,R3) = DM \text{ --- } (R1,R3) = DM$
- 36. $(R1,R2) = DM \text{ --- } (R3,R2) = DM \text{ --- } (R1,R3) = E$
- 39. $(R1,R2) = DM \text{ --- } (R3,R2) = DM \text{ --- } (R1,R3) = C$
- 42. $(R1,R2) = DM \text{ --- } (R2,R3) = C \text{ --- } (R1,R3) = DM$
- 44. $(R1,R2) = DM \text{ --- } (R2,R3) = C \text{ --- } (R1,R3) = C$
- 45. $(R1,R2) = DM \text{ --- } (R2,R3) = C \text{ --- } (R1,R3) = DJ$
- 47. $(R1,R2) = DM \text{ --- } (R2,R3) = DJ \text{ --- } (R1,R3) = DM$
- 49. $(R1,R2) = DM \text{ --- } (R2,R3) = DJ \text{ --- } (R1,R3) = C$
- 57. $(R2,R1) = DM \text{ --- } (R2,R3) = DM \text{ --- } (R1,R3) = DM$
- 59. $(R2,R1) = DM \text{ --- } (R2,R3) = DM \text{ --- } (R1,R3) = C$
- 69. $(R2,R1) = DM \text{ --- } (R2,R3) = C \text{ --- } (R1,R3) = C$
- 82. $(R1,R2) = C \text{ --- } (R2,R3) = DM \text{ --- } (R1,R3) = DM$
- 84. $(R1,R2) = C \text{ --- } (R2,R3) = DM \text{ --- } (R1,R3) = C$
- 85. $(R1,R2) = C \text{ --- } (R2,R3) = DM \text{ --- } (R1,R3) = DJ$
- 89. $(R1,R2) = C \text{ --- } (R3,R2) = DM \text{ --- } (R1,R3) = C$
- 92. $(R1,R2) = C \text{ --- } (R2,R3) = C \text{ --- } (R1,R3) = DM$
- 93. $(R1,R2) = C \text{ --- } (R2,R3) = C \text{ --- } (R1,R3) = DM$
- 94. $(R1,R2) = C \text{ --- } (R2,R3) = C \text{ --- } (R1,R3) = C$
- 95. $(R1,R2) = C \text{ --- } (R2,R3) = C \text{ --- } (R1,R3) = DJ$
- 97. $(R1,R2) = C \text{ --- } (R2,R3) = DJ \text{ --- } (R1,R3) = DM$

Appendix B

In this appendix we will show and discuss all the methods and functions that are used in the implementation of our program in further details.

a Method intersect

This method is used in order to determine the clusters of rules: we keep reading and adding Rules to the same cluster if it does not belong to any other cluster (ClusterID = -1) and if the current Rule's PacketArea has an intersection with Cluster's PacketArea. So, you need to check it for all the Rules that belong to the cluster.

An intersection can occur if the two rules have an equivalent IP address interval, either just source or destination addresses or both, but also if one IP address interval includes the other one.

Listing 1: intersect method

```
public static boolean intersect (Cluster cluster, RuleType rule)
{
    boolean res=false;
    for(PacketAreaType CLareaPacket : cluster.getPacketArea())
    {
        IPAddressType clusterIPS = CLareaPacket.getSourceIP();
        IPAddressType clusterIPD = CLareaPacket.getDestIP();

        IPAddressType ruleIPS = rule.getAreaPacket().getSourceIP();
        IPAddressType ruleIPD = rule.getAreaPacket().getDestIP();

        if(ruleIPS.equalFields(clusterIPS) &&
           ruleIPD.equalFields(clusterIPD))
        {
            res=true;
            break;
        }
        else if(ruleIPS.equalFields(clusterIPS) ||
                ruleIPD.equalFields(clusterIPD))
```

```

    {
        res=true;
        break;
    }
    else if(ruleIPS.isIncludedIn(clusterIPS) ||
            ruleIPD.isIncludedIn(clusterIPD))
    {
        res=true;
        break;
    }
    else if(clusterIPS.isIncludedIn(ruleIPS) ||
            clusterIPD.isIncludedIn(ruleIPD))
    {
        res=true;
        break;
    }
}

return res;
}

```

b Methods generate and helper

These two methods are displayed together since they are tightly linked: *generate* is the wrapper function that calls the recursive method *helper* that generates all the possible RuleCouples following the combinatorial analysis formula of the combination without repetition.

These functions were taken from the internet and then modified for our purposes.

Considering that they are recursive, they might not be the most efficient solution, but in our case we accept their cost since they are easy to use. Also you might want to consider to use libraries such as CombinatoricsLib, Guava and ApacheCommons, that are specifically designed to perform combinatorial calculus and analysis, therefore more efficient and performant.

Listing 2: generation method

```

public static ArrayList<RuleCouple> generate(int n, int k, int
    cl)
{
    ArrayList<RuleCouple> combinations = new ArrayList<>();
    helper(combinations, new RuleType[k], 0, n - 1, 0, cl);
    return combinations;
}

```


Listing 3: helper method

```
private static void helper(ArrayList<RuleCouple> combinations,
    RuleType data[], int start, int end, int index, int clusterID)
{
    if (index == data.length)
    {
        RuleType[] combination = data.clone();
        RuleCouple rc = new
            RuleCouple(BigInteger.valueOf(combinations.size()+1));
        rc.setFirstRule(combination[0]);
        rc.setSecondRule(combination[1]);
        rc.setCoupleRelation(BigInteger.valueOf(-1));
        if(combination[0].getAction()==combination[1].getAction())
            rc.setSameAction(true);
        else
            rc.setSameAction(false);

        combinations.add(rc);
    } else {
        int max = Math.min(end, end + 1 - data.length + index);
        for (int i = start; i <= max; i++) {
            data[index]=
                allcluster.get(clusterID).getClusterRules().get(i);
            helper(combinations, data, i + 1, end, index + 1,
                clusterID);
        }
    }
}
```

c Method findMinPriorityRule

This method is an auxiliary function that is used to find the rule with a lower priority inside a RuleCouple (passed as a parameter). The method returns this rule.

It is used after generating assigning the relationship value to the RuleCouple, if the method *checkValueRelationFunction* returns 1, then it is the case of a sub-optimization anomaly of type duplication and we need to remove it. So, we eliminate the rule with a lower priority in the RuleCouple, in order to find this Rule we use this method.

Listing 4: findMinPriorityRule method

```
public static RuleType findMinPriorityRule(RuleCouple rc)
{
    RuleType rule1 = rc.getFirstRule();
    RuleType rule2 = rc.getSecondRule();
    if(rule1.getRulePriority().intValue()
        <rule2.getRulePriority().intValue())
        return rule2;
    else
        return rule1;
}
```

d Method removeRuleCouple

This method is called to remove all the RuleCouples that contain a certain Rule *otherRule*. Hence, it must be used when there is:

- duplication anomaly = you remove the lower priority Rule, then you must remove also all the RuleCouples that contain that Rule.
- contradiction = you remove the loser Rule, then you must remove also all the RuleCouples that contain that Rule.
- shadow conflict = the winning rule has a higher priority, therefore you must remove the lower priority Rule and also all the RuleCouple that contain that Rule.

Listing 5: removeRuleCouple method

```

private static void removeRuleCouple(List<RuleCouple>
    clusterRuleCouples, RuleType otherRule)
{
    int i;
    ArrayList<RuleCouple> listToRemove = new
        ArrayList<RuleCouple>();
    for(RuleCouple couple: clusterRuleCouples)
    {
        if(couple.containsRule(otherRule))
            listToRemove.add(couple);
    }
    clusterRuleCouples.removeAll(listToRemove);
}

```

e Method query

This method was mentioned in Chapter 8.3, now we can have a look at the code implementation.

Listing 6: query method

```

private static int query(Scanner input, RuleType ruleToQuery,
    List<RuleType> correctRules, List<RuleType> clusterRules)
{
    System.out.println("Given the fact that these rules are
        correct:");
    if(correctRules!=null)
    {
        for(RuleType rule: correctRules)
        {
            String action="";
            if(rule.getAction().intValue()==0)
                action="DENY";
            else
                action="ALLOW";
            System.out.println("RuleID: " + rule.getRuleID() + " -
                action: " + action);
        }
    }
    else
    {
        System.out.println("NONE");
    }
}

```

```

        System.out.println("Is R" + ruleToQuery.getRuleID()+"
            correct?");
        System.out.print("Admin, please answer (yes=1/no=0): ");
        int answer = input.nextInt();
        return answer;
    }

```

f Method getRuleByID

This method is used to retrieve the index inside the list of the Cluster's Rules of the Rule that you have to query. It is used right after the method *query*.

Listing 7: getRuleByID method

```

public static int getRuleByID (BigInteger ruleID, int indexCL)
{
    int i,index=0;
    RuleType find=null;
    for(RuleType rule: allrules)
    {
        if(rule.getRuleID()==ruleID)
        {
            find=rule;
            break;
        }
    }
    for(i=0;
        i<allcluster.get(indexCL).getClusterRules().size();i++)
    {
        if(find==allcluster.get(indexCL).getClusterRules().get(i))
            index=i;
    }
    return index;
}

```

g Method invertPriority

This method is used to swap the priorities of two rules when the winner rule in a conflict has a lower priority. We use the method sort to reorder the Rules inside the cluster.

Listing 8: invertPriority method

```
private static void invertPriority(RuleType winner, RuleType
    loser, List<RuleType> clusterRules)
{
    BigInteger tmp = loser.getRulePriority();
    for(RuleType rule: clusterRules)
    {
        if(rule==winner)
            rule.setRulePriority(tmp);
        else if(rule==loser)
            rule.setRulePriority(winner.getRulePriority());
    }
    Collections.sort(clusterRules);
}
```

h Method findLastRule

When the program has received from the administrator N-1 negative responses, then we can avoid asking the last remaining rule since we know for sure that this is going to be correct. However we still need to find the Rule that is still to not queried, in order to solve its conflicts as if we received a positive response from the administrator.

Listing 9: findLastRule method

```
private static RuleType findLastRule(List<RuleType> clusterRules)
{
    for(RuleType rule: clusterRules)
    {
        if(rule.isAsked()==false)
            return rule;
    }
    return null;
}
```

i Method algorithm

This is the main code of the application in which the methods are called.

Listing 10: algorithm method

```
private static void algorithm()
{
    int i,j, cluster_count=0, issafe=0, queriedAll=0,notconflict=0;
    Scanner input = new Scanner(System.in);

    //1. clusters' creation
    for(i=0;i<allrules.size();i++)
    {
        if(allrules.get(i).getClusterID() == BigInteger.valueOf(-1))
        {
            RuleType rule = allrules.get(i);
            cluster_count++;
            Cluster cl = new
                Cluster(BigInteger.valueOf(cluster_count));
            cl.getClusterRules().add(rule);
            rule.setClusterID(BigInteger.valueOf(cluster_count));
            cl.getPacketArea().add(rule.getAreaPacket());
            for(j=i+1;j<allrules.size();j++)
            {
                if(allrules.get(j).getClusterID()==
                    BigInteger.valueOf(-1) &&
                    intersect(cl,allrules.get(j)))
                {
                    cl.getClusterRules().add(allrules.get(j));
                    allrules.get(j).setClusterID(BigInteger.
                        valueOf(cluster_count));
                    cl.getPacketArea().add(allrules.get(j).getAreaPacket());
                }
            }
            allcluster.add(cl);
        }
    }

    //2. RuleCouple's generation inside each cluster
    for(i=0;i<allcluster.size();i++)
    {
        int cl = i;
        ArrayList<RuleCouple> rulecouples =
            generate(allcluster.get(i).getClusterRules().size(),2,cl);
        RuleType ruleToRemove;
        do
```

```

{
    ruleToRemove = null;
    for(RuleCouple rc: rulecouples)
    {
        if (rc.getCoupleRelation().signum() < 0)
        {
            int res= checkValueRelationFunction(rc);
            if(res == -1)
            {
                ruleToRemove = findMinPriorityRule(rc);
                break;
            }
        }
    }

    if (ruleToRemove != null)
    {
        allcluster.get(i).getClusterRules().remove(ruleToRemove);
        removeRuleCouple(rulecouples, ruleToRemove);
    }

    }while (ruleToRemove != null);

    allcluster.get(i).getClusterRuleCouples().addAll(rulecouples);
}

/*print to see the relationship values, used to check if
   everything is correct*/
for(Cluster cluster : allcluster)
{
    for(RuleCouple couple : cluster.getClusterRuleCouples())
        System.out.println("COUPLES: (" +
            couple.getFirstRule().getRuleID()+ "," +
            couple.getSecondRule().getRuleID() +") = " +
            couple.getCoupleRelation());
}

//3. solve the conflicts in each cluster
for(i=0;i<allcluster.size();i++)
{
    issafe=0; //reset flag
    ArrayList<RuleType> correctRules = new ArrayList<RuleType>();
    while((!allcluster.get(i).getClusterRuleCouples().isEmpty())
        && issafe==0)
    {

```

```

RuleType ruleToQuery =
    findBestRuleToQuery(allcluster.get(i).getClusterRules(),
        allcluster.get(i).getClusterRuleCouples());
int answer = query(input,ruleToQuery,correctRules,
    allcluster.get(i).getClusterRules());
int index= getRuleByID(ruleToQuery.getRuleID(),i);

if(answer==1) //the rule is correct
{
    allcluster.get(i).getClusterRules().get(index)
        .setAsked(true);
    allcluster.get(i).getClusterRules().get(index)
        .setResponse(true);
    resolveConflicts(ruleToQuery,
        allcluster.get(i).getClusterRuleCouples(),
        allcluster.get(i).getClusterRules());
    correctRules.add(ruleToQuery);
    //new order of rule priority:
    for(RuleType rule: allcluster.get(i).getClusterRules())
        System.out.println("rule: " + rule.getRuleID() + "
            priority: " + rule.getRulePriority());
}
else //the rule is wrong
{
    allcluster.get(i).getClusterRules().get(index)
        .setAsked(true);
    allcluster.get(i).getClusterRules().get(index)
        .setResponse(false);
    queriedAll=0;
    for(RuleType rule: allcluster.get(i).getClusterRules())
    {
        if(rule.isAsked()==false)
            queriedAll=1;
    }
    if(queriedAll==0) //if you queried all rules, then you
        have to restart
    {
        for(RuleType rule: allcluster.get(i).getClusterRules())
        {
            if(rule.isResponse()==false)
                rule.setAsked(false);
        }
    }
    int count=0;
    //check how many negative answers you received
    for(RuleType rule: allcluster.get(i).getClusterRules())

```



```

{
    if(rule.isAsked()==true && rule.isResponse()==false)
        count++;
}

//if count == N-1 then you may skip the last query
if(count==allcluster.get(i).getClusterRules().size()-1)
{
    RuleType winningRule =
        findLastRule(allcluster.get(i).getClusterRules());
    resolveConflicts(winningRule,
        allcluster.get(i).getClusterRuleCouples(),
        allcluster.get(i).getClusterRules());
    correctRules.add(winningRule);
    allcluster.get(i).getClusterRules().get(index)
        .setAsked(true);
    allcluster.get(i).getClusterRules().get(index)
        .setResponse(true);
    for(RuleType rule: allcluster.get(i).getClusterRules())
    {
        if(rule.isResponse()==false)
            rule.setAsked(false);
    }
}
}

//if there are no more conflict
if(allcluster.get(i).getClusterRuleCouples().isEmpty())
    issafe=1;
notconflict=1;

//print the remaining conflict in the cluster
for(RuleCouple couple :
    allcluster.get(i).getClusterRuleCouples())
{
    System.out.println("REMAINING CONFLICT: (" +
        couple.getFirstRule().getRuleID()+ "," +
        couple.getSecondRule().getRuleID()+ ") = " +
        couple.getCoupleRelation());
    if(couple.isSameAction()==false &&
        couple.getCoupleRelation()!= BigInteger.valueOf(3))
    {
        notconflict=0;
    }
}
}
if(notconflict==1)

```

```
        issafe=1;
    }
}
input.close();
System.out.println("No more conflicts in the firewall!");
}
```