ANTWERP UNIVERSITY

DOCTORAL THESIS

On the Evolvability of the TCP-IP Based Network Firewall Rule Base

Author:
ir Geert Haerens

Supervisor:
Prof. Dr. ir Herwig Mannaert

A thesis submitted in fulfillment of the requirements for the degree of Doctor in Applied Economics

in the

Management Information Systems

November 14, 2021

Declaration of Authorship

I, ir Geert Haerens, declare that this thesis titled, "On the Evolvability of the TCP-IP Based Network Firewall Rule Base" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:10/11/2021

[illegible]

"Things are not as simple as they look in reality."

Prof. Ann Haegemans, somewhere in time.

"It's been a long road.
Getting from there to here.
It's been a long time.
But my time is finally near.
I will see my dream come alive at last.
I will touch the sky.
And they're not gonna hold me down no more,
No they're not gonna change my mind.

'Cause I've got faith of the heart.
Going where my heart will take me.
I've got faith to believe.
I can do anything.
I've got strength of the soul.
No one's gonna bend or break me.
I can reach any star.
I've got faith.
Into the heart.
"

From "Star Trek Enterprise" - lyrics by Diane Warren - music by Russel Watson

ANTWERP UNIVERSITY

Abstract

Faculty of Business and Economics
Management Information Systems

Doctor in Applied Economics

On the Evolvability of the TCP-IP Based Network Firewall Rule Base

by ir Geert Haerens

A firewall is an essential network security component. The firewall rule base, the list of filters to be applied on network traffic, can have significant evolvability issues in a context where companies consider their firewall as complex. While sufficient literature exists on how to analyze a rule base, little research is available on how to properly construct a rule base upfront which prevents the occurrence of evolvability issues. According to Normalized Systems theory, a system is unstable under change if changes require an effort that is proportional to the type of change and the size of the system. A system that is unstable under change is considered non-evolvable. The issue with firewall changes relates to this instability under change. By analyzing the root cause of the evolvability issues and proposing design criteria making use of Normalized Systems theory, we attempt to solve the evolvability issues of TCP/IP-based firewalls. This work presents a set of design criteria to create an ex-ante proven evolvable rule base, as well as an algorithm which performs an essential step in converting an existing non-evolvable rule base into an evolvable rule base.

UNIVERSITEIT ANTWERP

Abstract

Faculteit Bedrijfswetenschappen en Economie
Beleidsinformatica

Doctor in de Toegepaste Economische Wetenschappen

Over de Evolueerbaarheid van de TCP-IP Gebaseerde Netwerk Firewall Configuratie

door ir Geert Haerens

De firewall is een essentiële beveiligingscomponent van bedrijfsnetwerken. De firewall rule base, zijn configuratie, bevat een lijst met regels volgens dewelke netwerkverkeer moet worden gefilterd. Deze configuratie kan ernstige evolueerbaarheidsproblemen hebben die ertoe leiden dat bedrijven hun firewall als een complex systeem gaan zien. Alhoewel er voldoende literatuur bestaat over hoe een configuratie/rule base kan geanalyseerd worden op problemen en conflicten, bestaat er weining effectief toepasbaar onderzoek dat aangeeft hoe een firewall moet geconfigureerd worden om de evolueerbaarheidsproblemen en voortvloeiende complexiteit te vermijden vanaf het initieel ontwerp. Volgens de Normalized Systems theory, is een systeem onstabiel onder verandering, indien de impact van een verandering niet alleen proportioneel is met het type van de verandering, maar ook met de grootte van het systeem. Een systeem dat onstabiel is onder verandering wordt als niet-evolueerbaar beschouwd. De problemen met de firewall configuratie zijn gerelateerd aan onstabiliteit t.g.v. verandering. Door de grondoorzaken van de evolueerbaarheidsproblemen te onderzoeken en een voorstel tot configuratieontwerp naar voren te schuiven, gebruik makend van de Normalized Systems theory, trachten we de evolueerbaarheidsproblemen van de TCP/IP gebaseerde firewall op te lossen. Dit werk stelt een set van ontwerpcriteria voor die leiden tot een evolueerbare rule base. Tevens wordt een algoritme voorgesteld dat een essentiële stap bevat om een bestaande niet evolueerbare rule base, om te zetten in een evolueerbare rule base.

Acknowledgements

"Thank you very much! Thank you very much! That's the nicest thing that anyone's ever done for me". What better way to start this acknowledgement than with the words of this Academy Award-winning song. While the performer was thanking Ebenezer for his timeful death, I would rather dedicate those words to all those who had a hand in this dissertation.

Thank you Herwig Mannaert for being my promoter and mentor over the past five years. Thank you Jan Verelst for being the instigator of my five-year mission, to seek out NS theory and new applications, to boldly write about firewalls as no one has done before. Thank you Robert Pergel for being part of my PhD commission.

This work would not have been possible without energy, and what better provider than Engie. Thank you Paul Buyle for inadvertently putting me on this track. Thank you Manuel Hervieu for letting me finish it.

There have been many sets of eyes and brains looking over my shoulder and pointing out numerous errors and shortcomings. Thank you Stefan Thys for every paper you reviewed over the past five years. Your support and friendship during this endeavor has been one of the more precious gifts I have ever received. Thank you Christophe De Clercq for the many debates we have had and will continue to have about NS and more earthly matters. Thank you Joeri Pavlovic, Philippe Le Cerf, and Frans Versteken, for reading my work and sharing your thoughts. Thank you Koen Van Damme for some extra mathematical insights. And a special thanks to Colleen O'Neill and her husband André Muise, for the overseas proofing work.

To all those people who asked me "How is your PhD going?", I want to say thank you. Having shown interest in my work has meant a lot to me, as it has become an essential part of who I am.

You never walk alone. To my girlfriend Ilse, and my two wonderful daughters, Annelies and Karolien, I want to say thank you for it all. Finally, I would have been nowhere without the support and opportunities my parents have given me. Father, I think you would have been proud. Mother, I know you are proud, as proud as I am of you.

Geert Haerens

...

Contents

List of Figures

List of Tables

List of Abbreviations

AVT	**A**ction **V**ersion **T**ransparency
API	**A**pplication **P**rogramming **I**nterface
AWS	**A**mazone **W**eb **S**ervices
BIBO	**B**ounded **I**nput **B**ounded **O**utput
CE	**C**ombinatorial **E**ffect
CSV	**C**omma **S**eparated **V**alues
DEMO	**Design** and **E**ngineering **M**ethodology for **O**rganizations
DI	**D**isjointness **I**ndex
DNS	**D**omain **N**ame **S**ervice
DSL	**D**igital **S**ubscriber **L**ine
DSF	**D**esign **S**cience **F**ramework
DSM	**D**esign **S**cience **M**ethodology
DSRM	**D**esign **S**cience **R**esearch **M**ethodology
DSMP	**D**esign **S**cience **M**ethodology **P**rocess
DVT	**D**ata **V**ersion **T**ransparency
EE	**E**nterprise **E**ngineering
FRANS	**F**irewall **R**ule Base **A**nalyser and **N**ormalizer **S**ystem
FQDN	**F**ully **Q**ualified **D**omain **N**ame
HR	**H**uman **R**esources
HTTP	**H**yper**t**ext **T**ransfer **P**rotocol
HTTPS	**H**yper**t**ext **T**ransfer **P**rotocol **S**ecured
IAF	**I**dentity **A**ware **F**irewall
IAM	**I**dentity and **A**ccess **M**anagement
ICF	**I**nter**C**onnect **F**iltering
ICMP	**I**nternet **C**ontrol **M**essage **P**rotocol
ILS	**I**terated **L**ocal **S**earch
IP	**I**nternet **P**rotocol
IS	**I**nformation **S**ystems
L1I	**L**evel **1** **I**terations
L2I	**L**evel **2** **I**terations
LS	**L**ocal **S**earch
NoR	**N**umber **o**f **R**ules
NoS	**N**umber **o**f **S**ervices
NoSG	**N**umber **o**f **S**ervice **G**roups
NoUG	**N**umber **o**f **U**nique **S**ervices
NS	**N**ormalized **S**ystems
OF	**O**bjective **F**unction
OS	**O**perating **S**ystem
OSI	**O**pen **S**ystems **I**nterconnection
PF	**P**ort **F**requencies
RINR	**R**elative **I**ncrease in **N**umber of **R**ules
RM	**R**esearch **M**ethodology

RO	**R**esearch **O**bjective
RQ	**R**esearch **Q**uestion
RTP	**R**eal-time **T**ransport **P**rotocol
SDI	**S**ervice **D**isjointess **I**ndex
SoC	**S**eparation **o**f **C**oncern
SoS	**S**epration **o**f **S**tate
SNMP	**S**imple **N**etwork **M**anagement **P**rotocol
SDN	**S**oftware **D**efined **N**etwork
SDF	**S**oftware **D**efined **F**irewall
SONET	**S**ynchronous **O**ptical **NET**working
TCP	**T**ransport **C**onnection **P**rotocoll
UDP	**U**ser **D**atagram **P**rotocol
VLAN	**V**irtual **L**ocal **A**rea **N**etwork
WiFi	**W**ireless **F**idelity
WoM	**W**ay of **M**odeling
WoO	**W**ay of **O**rganizing
WoS	**W**ay of **S**upporting
WoT	**W**ay of **T**hinking
WoW	**W**ay of **W**orking
ZT	**Z**ero **T**rust

This thesis is dedicated to Frederik Leemans — a great engineer, fantastic enterprise architect and homo universalis, but most importantly, a dear, loved and missed friend. He was my personal promoter and was looking forward to this work as eagerly as I was. This one goes out to you.

Chapter 1

Introduction

The Normalized Systems (NS) theory studies the evolvability of modular systems. Although the theory originates in software development, the insights of NS theory into modular structures and evolvability can be used in other domains. There has been previous PhD research on the topic of applying NS theory to enterprise engineering [1] [2], business processes [3], accounting systems [4] and document management [5]. This dissertation seeks to add the domain of IT infrastructure to this list. IT infrastructure is a constantly-evolving sub-domain within the field of Information Systems. Over a period of fifty years, this sub-domain has gone from the management of isolated mainframes to the world of the Internet of Things (in the broadest sense), wherein all possible digital devices are interconnected. In [6] we have shown that IT Infrastructure suffers from strong coupling between components and associated evolvability issues. For example: the introduction of a new anti-virus tool on all PCs, the replacement of a storage system, or the upgrade of an operating system. Each of these changes has potential to introduce a significant ripple effect that is proportional to the size of the environment. However, the problem is not limited to interaction between infrastructure components. The same issues exist inside an infrastructure component and even in the parts that can be configured. The TCP/IP-based firewall is an infrastructure component with serious evolvability issues. Some of these are inherent to the component, but the most serious evolvability issues are introduced during firewall configuration. We believe this may be avoided by capitalizing on the insights on modularity and evolvability that NS theory provides.

This dissertation concerns a complex technical component: the TCP/IP-based firewall. Before going into the details of the research, we begin in Section 1.1 wherein we explain the basics of the TCP/IP-based firewall. As this work relates to Normalized Systems theory, we provide a short introduction of NS theory in Section 1.2. Having set the scene, we continue in Section 1.3 with the introduction of the research problem. In Section 1.4, we outline the research methodology, and in Section 1.5 we describe our research objectives and our academic contribution. The final section, Section 1.6, outlines the structure of this dissertation.

1.1 The TCP/IP-Based Firewall

This section begins with an explanation of network basics, followed by the definition of a firewall rule, closing with how a firewall works with rules. We continue by explaining firewall group objects, and conclude by arguing for the relevance of further study of the evolvability of a firewall rule base.

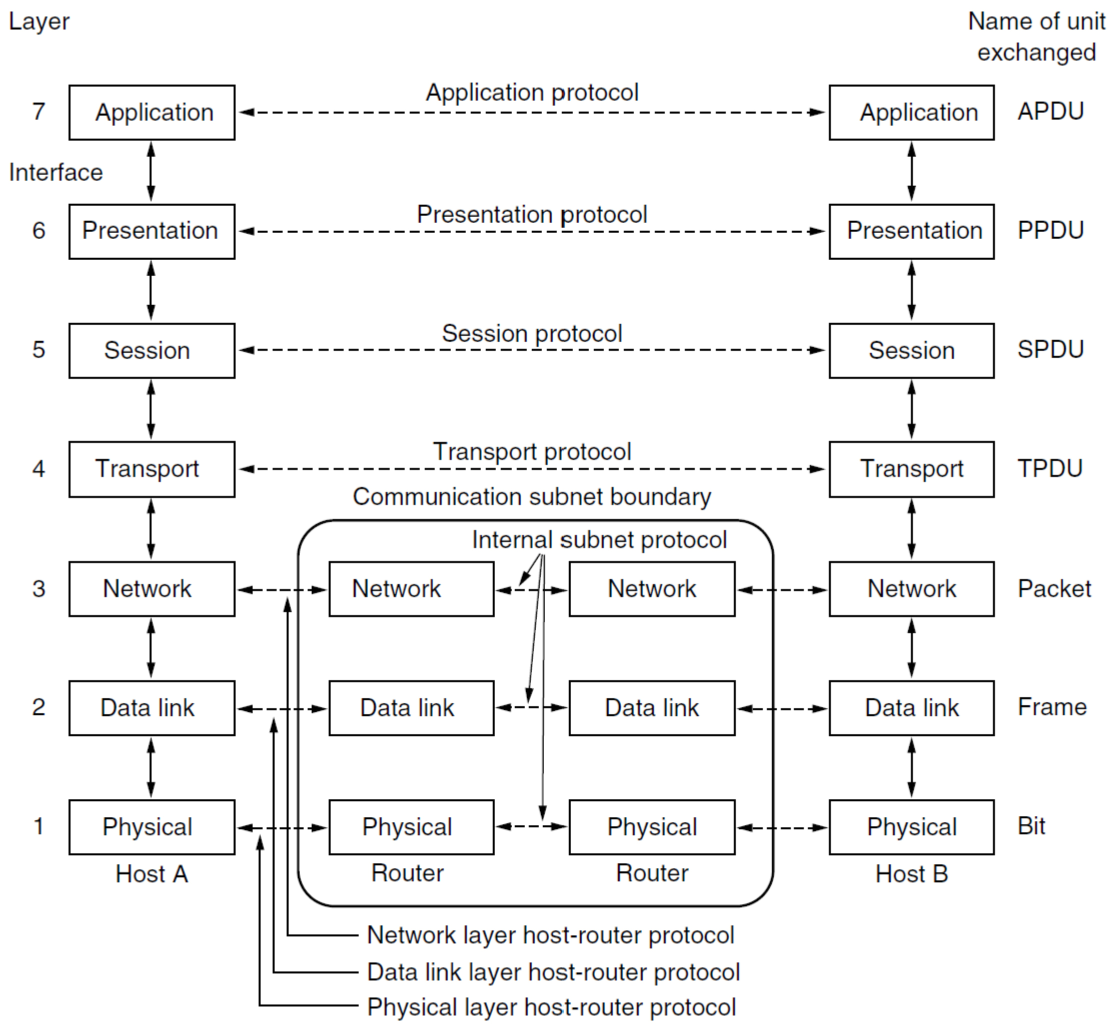


FIGURE 1.1: The OSI Reference Model (from [8])

1.1.1 Network Basics

The OSI model [7] [8] is a reference model that explains how two applications can communicate with each other over a network. It contains seven layers, each addressing one particular aspect related to communication over a network (see Figure 1.1). The model prescribes the interfaces that need to exist between the layers. As long as the implementation follows the direction of the interfaces, different implementation methods can be used and interchanged without affecting the other layers. The OSI model prescribes an evolvable modular structure.

The TCP/IP V4 protocol stack [9] is a concrete implementation of the OSI model and is the dominant method used to allow communication between IT infrastructure resources via a physical network. It does not comprehensively follow the OSI mode since it only implements four of the seven layers (see Figure 1.2).

- The Link Layer can be Ethernet, WiFi or DSL - these are the media through which the bits and bytes, transformed into some form of a electromagnetic signal, will travel.
- In the Internet Layer we see Internet Protocol (IP). Each network-connected resource must have a unique address. This is the IP address, which contains

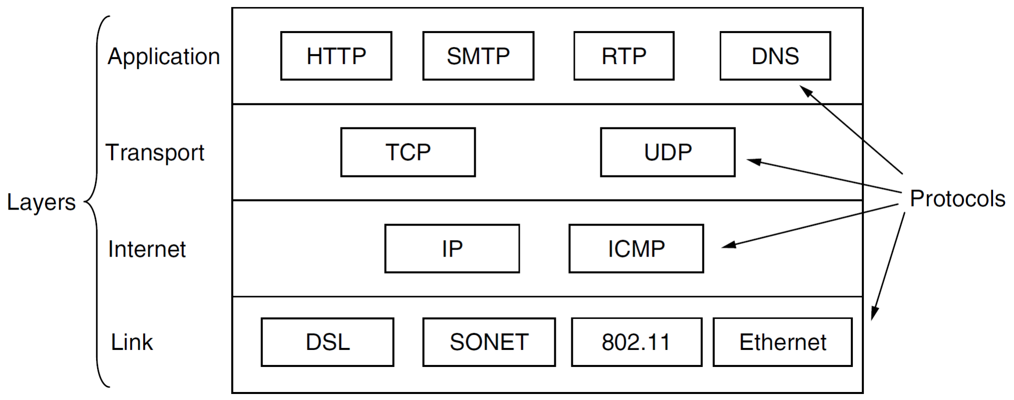


FIGURE 1.2: The TCP/IP Stack (from [8])

32 bits. This 32-bit address is represented as four numbers (each ranging from 0 to 255), separated by a dot (.), and split into four groups of eight bits (i.e., a byte).

Example: address 11111111.00000001.00000010.00000001, is represented as 255.1.2.1.

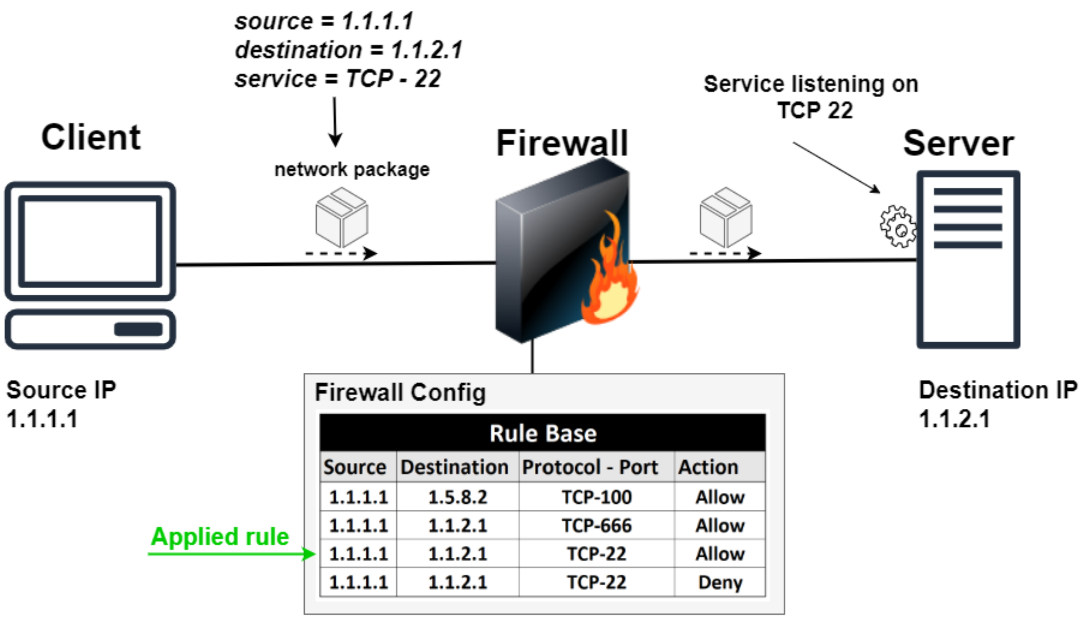
- The next layer up is the Transport Layer. This layer will send and receive packages and places them in the correct order before making them available to the Application Layer above. Each application in the Application Layer is assigned a TCP or UDP port number. It is this number that provides the necessary information to the Transport layer so that it can identify which application receives information. TCP and UDP are the two possible protocols in use at the Transport Layer. TCP and UDP ports are comprised of a 16-bit number, represented by its decimal value.
  Example: port 0000000000000001 is represented as port 1.

- The top layer is the Application Layer — the network package's final destination.

1.1.2 Firewall Functioning

The TCP/IP-based firewall located in the network path between resources can filter traffic between the resources based on the Internet Layer (IP address) and Transport Layer (TCP/UDP ports) properties of those resources. Filtering occurs through the application of rules. A rule is a tuple containing the following elements: <Source IP, Destination IP, Destination Port, Protocol, Action>. The rule is evaluated by the firewall, meaning that when the firewall receives traffic information originating from a resource with IP address =<Source IP>, going to resource =<Destination IP>, addressing a service listening on Port = <Destination port>, using Protocol = <Protocol>, the firewall will perform an action = <Action>. The action can be "Allow" or "Deny".

A firewall rule base is a collection of order-sensitive rules. The firewall will evaluate all inbound traffic against the ordered rule base. The firewall starts at the top of the rule base until it encounters the first rule that matches the criteria (Source, Destination, Destination Port, Protocol) of the traffic. The firewall then performs



Rule Base			
Source	Destination	Protocol - Port	Action
1.1.1.1	1.5.8.2	TCP-100	Allow
1.1.1.1	1.1.2.1	TCP-666	Allow
1.1.1.1	1.1.2.1	TCP-22	Allow
1.1.1.1	1.1.2.1	TCP-22	Deny

FIGURE 1.3: Firewall functioning

the action as specified in the rule. In a firewall rule, <Source IP>, <Destination IP>, <Destination Port> and <Protocol> can be one value or a range of values.

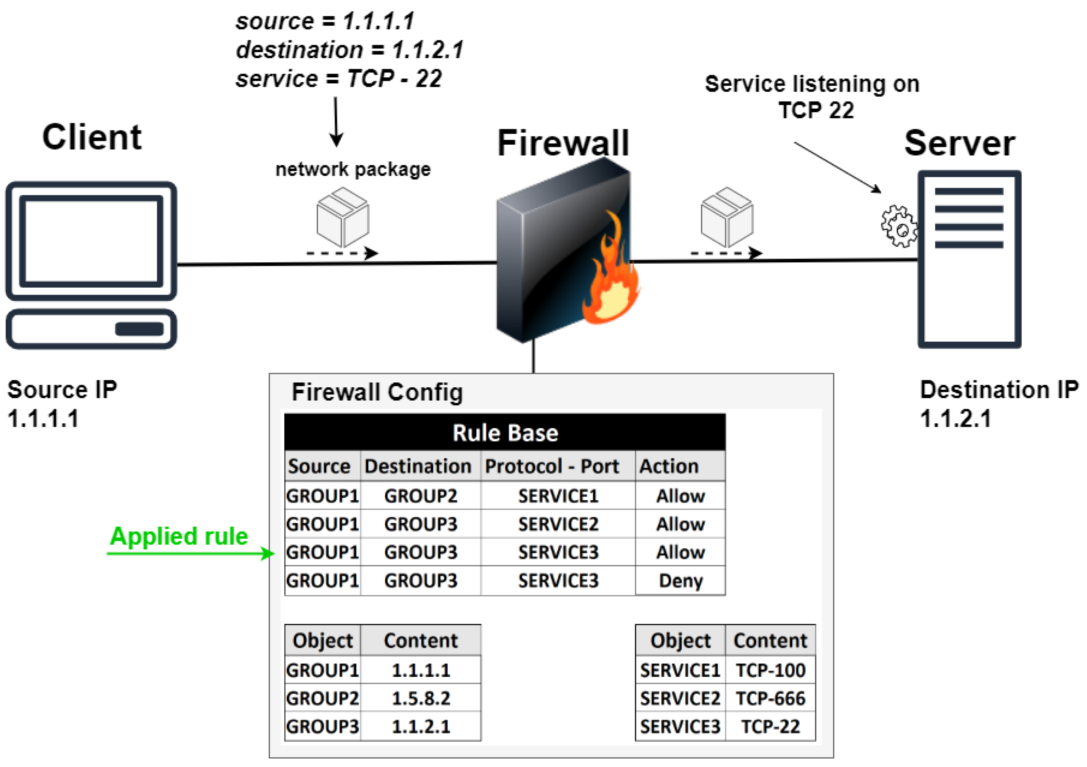
The protocol can be TCP or UDP. In the remainder of this document, the notion of protocol is omitted as it can be included in the Port variable (for example, TCP port 58 or UDP port 58).

Example: Figure 1.3 demonstrates the functioning of the firewall. A client with IP address 1.1.1.1 attempts to connect to a service listening on TCP port 22 on the server with IP 1.1.2.1. The firewall scans the rule base, starting at the top, until it finds a rule that is a match for source (1.1.1.1), destination (1.1.2.1), and service (TCP 22). The firewall performs the action specified in this matching rule. Any rules that also match the network package but that are located beyond the first matching rules are ignored.

1.1.3 Firewall Group Objects

It would be difficult for humans to interpret a rule base comprised of IP addresses as source/destination and port numbers. Users would be confronted by a series of apparently meaningless numbers. Firewalls allow the usage of firewall objects (called groups) to assign a logical name to a source, a destination, or a port, which is more human-friendly. Groups are populated with IP addresses or ports. Groups can be nested. Figure 1.4 demonstrates the usage of groups in the previous example of the firewall functioning.

Using groups should improve the manageability of the firewall. However, using groups may also result in degraded manageability.
For instance, “Group_Windows_APP” and “Group_Windows_APPS” could be two groups each containing the IP addresses of all Windows Application Servers. The latter may have been created without the knowledge of the former [10]. Although both groups represent the same concept, group memberships may start to deviate



Rule Base

Source	Destination	Protocol - Port	Action
GROUP1	GROUP2	SERVICE1	Allow
GROUP1	GROUP3	SERVICE2	Allow
GROUP1	GROUP3	SERVICE3	Allow
GROUP1	GROUP3	SERVICE3	Deny

Object	Content
GROUP1	1.1.1.1
GROUP2	1.5.8.2
GROUP3	1.1.2.1

Object	Content
SERVICE1	TCP-100
SERVICE2	TCP-666
SERVICE3	TCP-22

FIGURE 1.4: Firewall objects

from each other. We will see in Chapter 2 that this may lead to anomalies in the rule base. The group structure must be properly designed to avoid this.

1.1.4 Relevance of the Firewall in Today's Networks

Firewalls are an essential component of network security. They have been protecting network-connected resources for more than thirty years, and are expected to continue to do so for decades [11][12][13]. Initially, firewalls were used to protect business-computing operations against outside threats (i.e., the "evil Internet"). This type of filtering is called North-South traffic filtering [14]. However, security breaches are not only caused by access via the Internet. Indeed, a significant proportion of security breaches are caused from within company networks [15] where hacks have become more sophisticated. Getting a foothold on one resource on the internal network and from there on hopping between resources, is a known hacking strategy against which filtering North-South traffic offers no protection. For this reason, protecting the network-connected resources from internal traffic, referred to as East-West traffic [14], is gaining ground.

Networks are becoming increasingly complex: they often contain multiple firewalls protecting numerous network segments. The rule base of these firewalls (i.e., the definitions determining which traffic is and is not allowed) is becoming equally complex, approaching the point of unmanageability. In a survey organized by Firemon [10], 73 % of survey participants stated that their firewall ranges from "somewhat complex" to "out of control". Further, complexity is the highest-ranked challenge for firewall management [11][12] [13].

The firewall rule base is a classic example of a system that needs to e volve. What

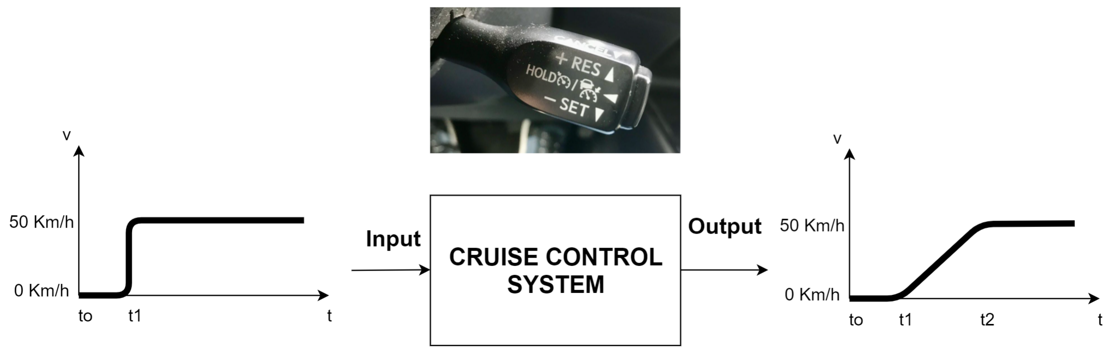


FIGURE 1.5: A stable system

begins with a single firewall that separates two network segments and filters rules between them rapidly increases in complexity. As the network grows, the number of resources connected to the network grows, as does the number of services offered on the network, and thus the potential number of security threats. The resulting firewall rule base will enlarge dramatically. This evolution will, at some point, result in a rule base where regular changes (i.e., the addition or removal of a rule) result in unforeseen side effects. Those effects are proportional to the size of the rule base: the bigger the system (rule base), the more pronounced the effects [11].

A network rarely contains only one firewall. Large companies have networks containing many firewalls. Valuable IT assets located in data centers are protected by multiple firewall layers. A single firewall can quickly become a non-evolvable system. Multiple firewalls only exacerbate the problem. Besides the question of how to create the correct rule and implement it on the rule base, one also must determine to which firewall(s) this rule should be applied.

Multiple vendors sell tools that analyze a firewall rule base and may even be used to simplify it (e.g., Firemon, Tufin, Algosec). Some academic research on such analyses is available. Both industry and academics seem to focus on improving existing rule bases. However, a more ambitious objective would be to avoid this type of problem at the outset of the firewall rule base design through the conscious restriction of design-space to incorporate evolvability.

1.2 Normalized Systems Theory

The Normalized Systems (NS) theory [16, 17, 18, 19] originates from the field of software development. There is widespread agreement in the software engineering community that the use of software modules decreases complexity and increases evolvability. It is also well understood that one should strive toward "low coupling and high cohesion". The problem is that the community has not achieved consensus as to how exactly "low coupling and high cohesion" are best achieved, nor what the ideal module size should be in order to achieve optimal low complexity and high evolvability.

NS theory takes the concept of system theoretical stability from the domain of classic

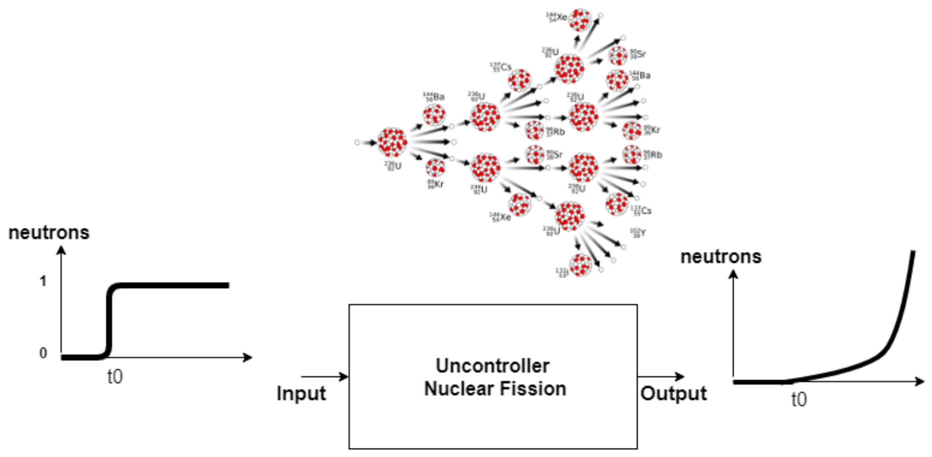


FIGURE 1.6: An unstable system

engineering, and applies it to the design cycle of systems in order to determine the necessary conditions which a modular structure of a system must meet in order for the system to exhibit stability when design changes are made. Stability is defined as Bounded Input results in Bounded Output (BIBO). A system is considered stable when a bounded input to the system leads to a bounded output. For example, the cruise control system of a car is a stable system, as a bounded input - the setting of a target speed for the car - leads to a bounded output - the car accelerating until it has reached the desired speed. A system is considered unstable if a bounded input leads to an unbounded output. An example of an unstable system is an uncontrolled nuclear reaction. A bounded input - firing a neutron into a mass of Uranium 235 - leads to an unbounded output - a chain reaction of neutrons being released due to fission, resulting in an explosion. See Figure 1.5 and Figure 1.6 for a graphical representation.

When translating this concept to software design, one can consider bounded input as a certain number of functional changes to the software and the bounded output as the number of effective software changes. If the amount of effective software changes is proportional to the amount of functional changes as well as to the size of the existing software system, then NS theory states that the system exhibits a Combinatorial Effect (CE) and is considered unstable under change. NS theory proves that four design rules for the modular software structure of the system must be respected. These design rules are a necessary condition to eliminate CEs. These rules are:

- Separation of Concern (SoC): a module should only address one concern or change driver.
- Separation of State (SoS): a state should separate the use of a module from another module during its operation.
- Action Version Transparency (AVT): a module that performs an action should be changeable without impacting the modules that call this action.
- Data Version Transparency (DVT): a module performing a certain action on a data structure should be able to continue doing this action, even if the data structures has undergone change (add/remove attributes).

Only by respecting these rules can a system undergo infinite growth while maintaining the capacity to incorporate new requirements.

Although NS theory originates in software design, the applicability of NS principles exists in other disciplines such as process design [3], organizational design [2], accounting [4], document management [5], and physical artifacts [20]. The theory may be extended to study evolvability in any system that can be seen as a modular system and derive design criteria for the evolvability of such a system. In this work, NS theory is used to underpin the study the evolvability of the TCP/IP firewall rule base.

NS theory studies combinatorics in modular systems and provides a set of theorems to design modular systems exhibiting *ex-ante* proven evolvability. The goal is to avoid so-called Combinatorial Effects (CEs). CEs are impacts that are proportional to the type of change as well as to the size of the system to which the change is applied. When all modules of a system respect NS theorems, the system will be free of such CEs. At that point, the system can be considered stable under change for a set of anticipated changes (such as adding and removing system components).

1.3 Research Problem: the Firewall as Non-Evolvable System

The TCP/IP-based firewall has been and will continue to be an essential network security component in protecting network-connected resources from unwanted traffic. The increasing size of corporate networks and connectivity needs has resulted in the considerable increase of firewall rule bases. Large rule bases have a nasty side effect: it becomes increasingly difficult to add the correct rule at the correct location in the firewall.

In larger rule bases, the probability of anomalies being present increases, potentially resulting in the erosion of the firewall's security policy or incorrect functioning. Changing the firewall rule base becomes increasingly complex as the size of the system increases. This observation is shared by [15] Forrester and the firewall security industry [12] [21].

NS theory defines a CE as the effect that occurs when the impact of a change is proportional to the nature of the change and the system's size. According to NS theory, a system that suffers from CEs is considered unstable under change. A firewall's vulnerability to CEs increases as the rule base size increases. Firewalls thus suffer from evolvability issues. Order sensitivity plays a vital role in the rule base's evolvability issues. The necessary condition to remove order sensitivity is known, i.e. non-overlapping or disjoint rules. However, firewalls don't enforce that condition, leaving open the potential for misconfiguration.

Issues with evolvability of the firewall rule base induce business risks. The first is the risk of technical communication paths being unavailable to properly execute business activities. The second is that flaws in the rule base may result in security risks, leading to the business's vulnerability to malicious hacks and resulting in damage to business activities. The third is the increased cost and delay of firewall changes.

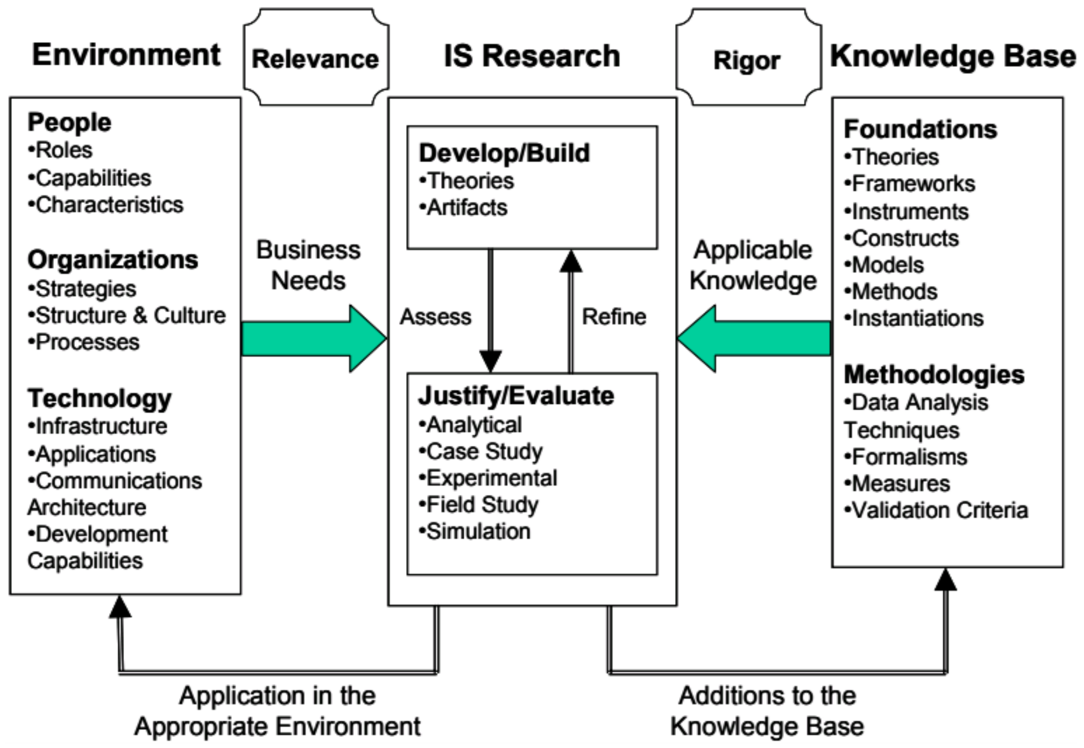


FIGURE 1.7: The Design Science Framework (from [23])

.

1.4 Research Methodology

The TCP/IP firewall is one component of an Information System (IS) landscape. Research in this landscape is referred to as IS research. The IS research within this dissertation relies heavily on Design Science Methodology. Researchers attempt to identify a solution for an existing problem on a technical system. The term "technical system" refers to a man-made technical system with no human agency and within which cause-and-effect relations are deterministic rather than correlated. Design Science, the science of the artificial [22], is a relatively new discipline that aims at increasing the scientific rigor in the creation of artifacts. Because Design Science does not follow the same steps that the classical scientific method requires, it has received push-back from adherents of the latter community.

According to classical Research Methodology (RM), and more precisely, research that aims to observe the effect of applying a treatment aimed at having an effect on a population, the treatment is taken as a given. However, treatments are consciously created artifacts aimed at inducing an effect. They required conscious design, to ensure that they address the problem at hand, can be administered, and produce an intended effect. Without conscious design of the treatment, the resulting experiment and associated contribution to science is of little value.

Gregor and Hevner [23] propose a Design Science Framework (DSF) for IS research, comprised of two interacting cycles (Figure 1.7): 1) the Relevance Cycle, i.e. the interaction of what is being designed and the environmental context to which it will be applied, and 2) the Rigor Cycle, which links what is being designed to the existing knowledge base and thereby ensuring that it adds new knowledge.

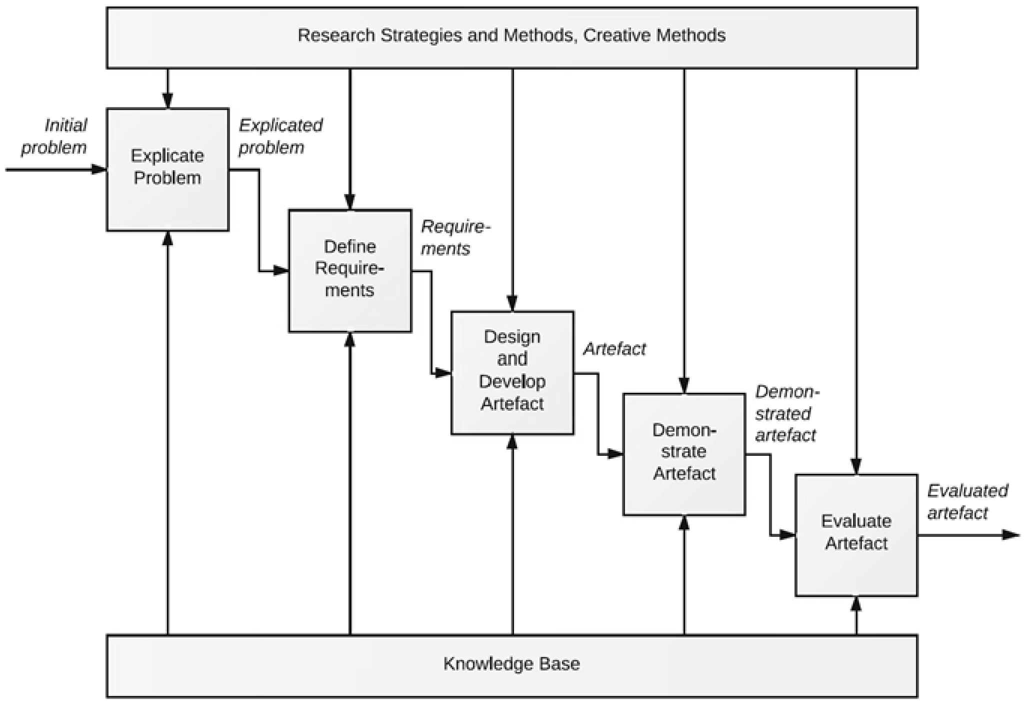


FIGURE 1.8: Design Science Research Methodology Process (from [26]).

Work undertaken according to Design Science Research Methodology (DSRM) should include both cycles. The Relevance Cycle concerns identification of a real problem that is applicable to either persons, organizations or technologies and whose significance warrants IS Research. During IS Research, the artifact is created, tested, and improved across multiple access/refine cycles. The end result of IS Research must be an artifact that is actually applicable to the environment wherein the problem resides. The Rigor Cycles must feed the IS Research with foundational and domain knowledge related to the problem, together with relevant methodologies for studying the domain, and ensure the required rigor during artifact creation, assessment and refinement. The end result — i.e., the artifact — should not only solve the problem but also add new knowledge to the existing Knowledge Base.

Peffer [24] proposes a Design Science Methodology Process (DSMP) (Figure 1.8) consisting of various steps, ranging from problem description to the collection of requirements and artifact design, with the final steps of demonstration and evaluation. These process steps are well suited as a reporting canvas for Design Science-related research [25] [26].

1.5 Research Objective and Questions

In Design Science, the focus shifts from Research Question (RQ) to Research Objective (RO). Our objective is to address the evolvability issues of the TCP/IP firewall, by means of artifacts. Our purpose is to create two artifacts:

- RO1: A *green-field artifact*, as a method to create an evolvable firewall rule base that is *ex ante* proven evolvable, assuming one has the luxury of starting from

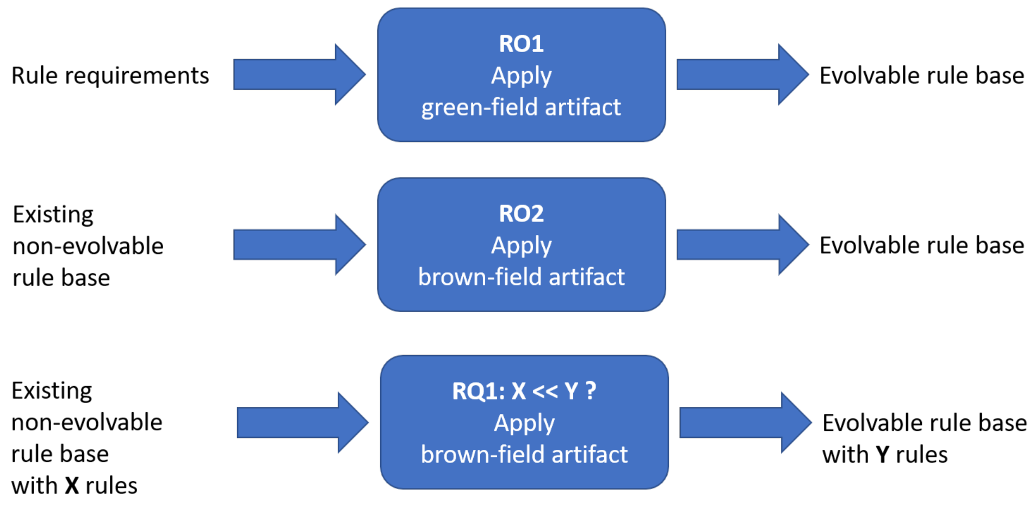


FIGURE 1.9: RO1, RO2 and RQ1 conceptual model

an empty rule base that will grow.

- RO2: A *brown-field artifact*, as an algorithm that transforms an existing firewall rule base from a non-evolvable state to an evolvable state while keeping intact the existing firewall logic.

The application of NS theory results in a fine-grained modular structure. We expect this to also be the case when applying NS theory to firewalls, meaning that in doing so we would expect to obtain more fine-grained rule base. When our brown-field artifact is applied to an existing rule base, we expect the number of rules to increase. We do, however, not know the order of magnitude of this increase. The relationship between the application of the result of RO2 (brown-field artifact) and the increase in rule base size will be studied in RQ1.

Associated with RO2, comes RQ1:

- RQ1: How many additional rules will the application of the brown-field artifact introduce?

The conceptual models of the ROs and RQ may be found in Figure 1.9.

Research conducted on such artifacts contributes to the DSF Relevance Cycle. This is in part because evolvability issues impact organizations (the need), and their application can be expected to improve evolvability and thus lower the negative impact (on organizations) of firewalls (technology). We also contribute to the DSF Rigor Cycle by grounding the artifact in an existing body of knowledge related to firewall evolvability issues and addressing root causes. NS theory aids in understanding the evolvability issue and structuring the artifacts. The classical Research Methodology will help in investigating the validity of the experimental paradigm to answer RQ1. The existing Knowledge Base is enriched by the formal expression of the size of the problem, a method of measuring the degree of evolvability, the algorithm that converts a non-evolvable rule base into an evolvable rule base, as well as insights on the impact of the artifact on rule base size. By making artifacts that are grounded in Normalized Systems theory, we extend the applicability of NS theory to a new domain and thus strengthen NS theory in terms of Design Theory [19]].

1.6 Structure of the Dissertation

As proposed by Gregor and Hevner [25], this work is structured according to the DSMP. Chapter 2 - Problem Description - wherein we review the existing literature on firewall issues and expand on the problem's various dimensions. Chapter 3 – Artifact Requirements – wherein we discuss the expectation for an artifact that will either build a new rule base with evolvability as a design criterion — referred to as a green-field artifact — and the expectations for a brown-field artifact that will convert an existing non-evolvable rule base into an evolvable rule base. Chapter 4 – Green-Field Artifact Creation and Demonstration – wherein we address the creation, by means of NS theory applied as a Design Theory, and demonstrate a design method for rules that will lead to an evolvable rule base. Chapter 5 - Brown-Field Artifact Creation and Demonstration - wherein we propose and demonstrate an algorithm based on meta-heuristics that will deliver an essential building block for converting a non-evolvable rule base into a rule base that adheres to the design criteria of the green-field artifact. Demonstration of the artifact will occur by means of an experiment which proves that the brown-field algorithm was responsible for the observed effect. Further, we will demonstrate the impact of the artifact on the size of the rule base. Chapter 6 – Artifact Implications – wherein we focus on the implication of various filtering strategies on the artifacts, on extending the evolvability of a single firewall to multiple firewalls, and on firewall scaling. Chapter 7 – Evaluation and Discussion – wherein we critically examine adherence to the DSF and use the classical Research Methodology to ground the various validity claims involved in artifact creation and performance. Chapter 8 - Conclusion and Future Work - wherein we summarize and conclude the dissertation.

Chapter 2

Problem Description

This chapter discusses the problems related to the firewall rule base. Section 2.1 begins with a literature review related to firewall rule base issues. In Section 2.2, we continue by zooming in on the most important cause underlying the issues — the relationships between rules. In Section 2.3, we formalize the components comprising a rule base, and examine the combinatorics of the components in order to understand the size of the problem space. In Section 2.4, we present an ontological model of a firewall rule base and its components. This ontological model serves as a reference to study the implemented data model used by the three major firewall producers (Fortinet, Palo Alto, Check Point), which will be reverse-engineered, based on rule base exports, in Section 2.5. In Section 2.6, we discuss the issues related to both the ontological model and the implementation models. The chapter ends with a conclusion, which addresses all identified problem dimensions of firewall rule bases, in Section 2.7. The findings of this chapter have been published in [27], [28] and [29].

2.1 Literature Review and Related Work

The academic literature concerning firewalls can be divided into three essential groups. Published roughly between 1990 and 2000, the first of these focuses on the performance of the firewall and the hardware used to perform the actual package filtering. The second group (published roughly between 2000 and 2006) focuses on the complexity and issues with the rule base of the firewall. The final group (published roughly after 2006) focuses on the firewall within the context of a Software Defined Network (SDN), where distributed firewalls and software-defined firewalls are used.

As the present thesis focuses on the complexity and issues related to the firewall rule base, the following literature review will focus exclusively on the second group of papers [30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41]. Our research did not iden-tify existing academic literature that specifically addresses and attempts to solve the evolvability issues of the firewall rule base. In the absence of such published studies, we conclude that academic literature on this specific question does not exist.

In addition to academic papers, we studied reports from Forrester, as well as various white papers published by industry leaders [10, 11, 12, 21, 42] . Those include surveys, which provide information on the current state of affairs. One might be forgiven for concluding, based on observing a post-2006 decline in the number of academic publications concerning rule base issues, that the problem is solved. However, the surveys support an alternate conclusion. Companies continue to struggle with their firewalls [10, 11, 12, 14, 15, 21, 42, 43]. This can be due to the "knowing-doing" gap or because the issue is not fully resolved.

Most papers that discuss firewall anomalies and anomaly resolution algorithms start by stating that there is a problem with the firewall rule base due to:

- ***Translation issues***: how to convert a high-level security policy into the low-level language of firewall rules [30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42].
- ***Issues related to the size of the rule base***: a large rule base is considered complex [10, 33, 37, 38, 39].
- ***Error and anomalies issues***: A rule base is error-prone due to complexity and manual interventions [10, 11, 12, 14, 15, 21, 32, 33, 39, 42, 43] and may contain firewall rule conflicts or anomalies [10, 21, 30, 31, 32, 33, 36, 38, 39, 41].

The ***"Translation-issue"*** is tackled by proposing tools which can translate high-level security concepts into low-level firewall rules. FANG [36], FIRMATO [33], LUMETA [35] are artifacts that assist this translation. There is, however, no guarantee that these tools will deliver a small and simple firewall rule base free of anomalies [33]. Companies such as TUFIN, ALGOSEC, FIREMON, and VMWare also offer commercial tools which claim to help manage network security complexity. The tools, however, neither prescribe nor enforce how a rule base should be created in order to be anomaly free and exhibit evolvability.

The ***"Size of the rule base issue"*** is the subject of much attention. Effort is put into reducing the rule base to a minimum list of rules while still meeting the filtering requirements. The motivation for this "reduction of the rule base" is performance, although in [33] it was suggested that the actual size of the rule base is unrelated to the manner by which the hardware actually applies the rules. This suggests that size of the rule base is uncorrelated to firewall performance.

The ***"Error issue"***, arising from complexity and manual intervention, is recognized and confirmed in recent surveys [10, 11, 12, 14, 15, 21, 42, 43]. Available academic papers focus primarily on the anomalies in the rule base as the technical root cause of error. Over time, the identification of the anomaly types, as well as their formal definition and proof, became more precise and resulted in formalizing how a firewall rule base should be formulated in order to remain stable under change: it should only include disjoint rules [30, 32, 38, 39, 40, 41]. Artifacts have been proposed [30, 32, 33, 37, 38, 41], which would allow for scanning the rule base for non-disjoint rules and, if required, making them disjoint. The same artifacts allow one to assess the impact of adding a new rule and adjusting the rules whereby the rule base exclusively contains disjoint rules. However, each time a rule is entered, the entire rule base must be scanned in order to detect potential anomalies between the existing rule base and the new rule. Thus, the effort required to make a change to the system is proportional to system size.

The literature review shows that the problems related to the firewall rule base are well understood, as is the necessary condition to keep the rule base under control (i.e., having disjoint rules). However, clear architectural guidance on how to create a disjoint rule base from the very moment of conception is lacking. It is precisely this architectural guidance utilizing NS theory that is the main contribution of this doctoral thesis.

By structuring the rules such that that they are always disjoint, one may add and remove rules without having to analyze the rule base or be concerned about unforeseen side effects of the change.

2.2 Rule Relationships and Order Sensitivity

As a rule base changes over time, different rules start interfering with each other, resulting in complexity.
Based on [30], the relationships between rules and rule components are defined as follows:

- **Field**: A field in a rule is defined as a source, destination or service. A field is a set of values, with a minimum of size one.
  ***Example**: The source field of a rule contains 3 IP addresses/values - (10.10.10.1, 10.10.10.2, 10.10.10.3)*

- **Equal Fields**: Two corresponding fields of two rules are equal if the set of values of the fields are the same.
  ***Example**: The source field of a rule **R1** and source field in rule **R2** contain the same 3 IP addresses - (10.10.10.1, 10.10.10.2, 10.10.10.3)*

- **Inclusive Fields:** Two corresponding fields of two rules are inclusive if the set of values of the field of the first rule are a subset of, but not equal to, the second rule field's set of values.
  ***Example**: The source field of **R1** contains (10.10.10.1, 10.10.10.2) and the source field of **R2** contains (10.10.10.1, 10.10.10.2, 10.10.10.3). The IPs (10.10.10.1, 10.10.10.2) are a subset of (10.10.10.1, 10.10.10.2, 10.10.10.3). The source field of **R1** is inclusive with regards to the source field of **R2**.*

- **Correlated Fields**: Two corresponding fields of two rules are correlated if there are some values, but not all, of the field of the first rules that are equal to some values, but not all, of the field of the second rule. The intersection between the sets of values of the fields is not empty, but the fields are not equal or inclusive either.
  ***Example**: The source field of **R1** contains (10.10.10.1, 10.10.10.2, 10.10.10.3) and the source field of **R2** contains (10.10.10.2, 20.20.20.20, 30.30.30.30). The two source fields are correlated as they intersect with the IP 10.10.10.2.*

- **Distinct Fields:** Two corresponding fields in two rule are distinct if they are not equal, not inclusive or not correlated. The intersection between the sets of values of the fields is empty.
  ***Example**: Source field (10.10.10.10) of rule **R1** and source field (10.10.10.100) of rule **R2** are distinct.*

- **Matching Fields:** Two corresponding fields in two rules match if they are equal or inclusive.
  ***Example**: Source field of **R1** = (10.10.10.1, 10.10.10.10) and the source field of **R2** = (10.10.10.1, 10.10.10.10, 10.10.10.30), are matching.*

- **Exactly Matching Rules:** Rules **R1** and **R2** are exactly matched if every field in **R1** is equal to the corresponding field in **R2**.
  ***Example**: Rule **R1**: (source = (10.10.10.10); destination = (20.20.20.20); service =*

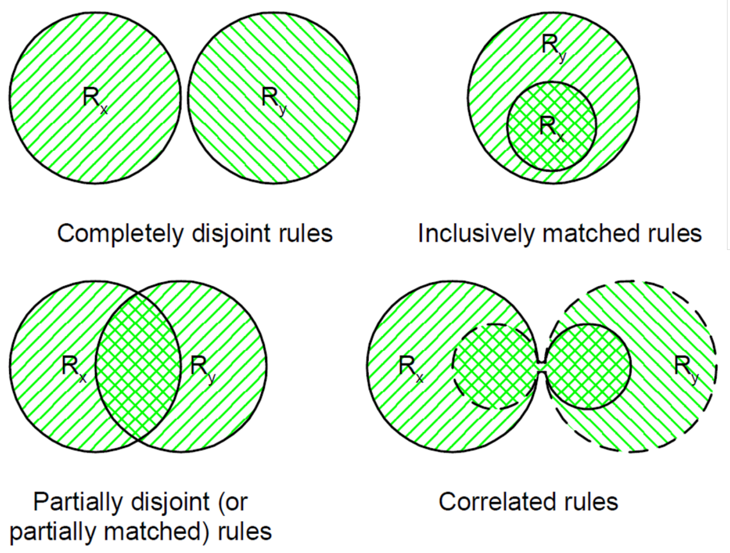

R x
R y
Completely disjoint rules
R y
R x
Inclusively matched rules
R x
R y
Partially disjoint (or partially matched) rules
R x
R y
Correlated rules

FIGURE 2.1: Possible relationships between rules (from [30])

(TPC 100); action = allow) and ***R2****: (source = (10.10.10.10); destination = (20.20.20.20); service = (TPC 100); action = deny), are exactly matching rules.*

- **Completely Disjoint Rules:** Rules **R1** and **R2** are completely disjoint if every field in **R1** and **R2** is distinct.
  Example*: Consider rule* ***R1****: (source = (10.10.10.10); destination = (20.20.20.20); service = (TPC 100); action = allow) and rule* ***R2****: (source = (30.30.30.30, 30.30.30.21); destination = (40.40.40.40, 40.40.40.41); service = (TPC 200,201); action = deny). Both rules are completely disjoint.*

- **Partially Disjoint Rules or Partially Matching Rules:** Rules **R1** and **R2** are partially disjoint (or partially matched) if there is at least one field in **R1** and **R2** that is distinct. The other fields can be equal, inclusive or correlated.
  Example*: Consider rule* ***R1****: (source = (10.10.10.10); destination = (20.20.20.20); service = (TPC 100); action = allow) and rule* ***R2****: (source = (10.10.10.10); destination = (40.40.40.40, 40.40.40.41); service = (TPC 100,201); action = deny).* ***R1*** *and* ***R2*** *are partially disjoint, as destination is a distinct field.*

- **Inclusively Matching Rules:** Rules **R1** and **R2** are inclusively matched if there is at least one field that is inclusive, and the remaining fields are either inclusive or equal.
  Example*: Consider Rule* ***R1****: (source = (10.10.10.10); destination = (20.20.20.20); service = (TPC 100); action = allow) and* ***R2****: (source = (10.10.10.10, 10.10.10.11); destination = (20.20.20.20, 20.20.20.21); service = (TPC 100); action = deny). Then rule* ***R1*** *inclusively matches rule* ***R2****.*

- **Correlated Rules**: Rules **R1** and **R2** are correlated there is at least one field that is correlated, while the remaining fields are either equal or inclusive.
  Example*: Consider rule* ***R1****: source = (10.10.10.10, 10.10.10.11); destination = (20.20.20.20, 40.40.40.41); service = (TPC 100); action = allow) and rule* ***R2****: (source = (10.10.10.10); destination = (40.40.40.40, 40.40.40.41); service = (TPC 100,201); action = deny). Rules* ***R1*** *and* ***R2*** *are correlated.*

Figure 2.1 graphically represents these various relations. Exactly matching, inclusively matching, and correlated rules can result in the following firewall anomalies [32]:

- *Shadowing Anomaly*: A rule **R1** is shadowed by another rule **R2** if **R2** precedes **R1** in the policy, and **R2** can match all the packets matched by **R1**. The result is that **R1** is never activated.
- *Correlation Anomaly*: Two rules **R1** and **R2** are correlated if they have different filtering actions and **R1** matches some packets that **R2** matches and **R2** matches some packets that **R1** matches.
- *Redundancy Anomaly*: A redundant rule **R1** performs the same action on the same packets as another rule **R2** so that if **R1** is removed the security policy will be unaffected. Redundancy anomalies are a subset of the shadowing anomalies.

A fully consistent rule base should only contain disjoint rules. Disjoint rules are either completely disjoint or partially disjoint. If the rule base only contains disjoint rules, the order of the rules in the rule base is immaterial and the anomalies described above will not occur [30, 32, 38, 39, 40, 41]). However, due to several confounds (e.g., unclear requirements, faulty change management processes, lack of organization, manual interventions, and system complexity) [10], the rule base will include correlated, exactly matching, and inclusively matching rules. Combined with the order-sensitivity of the rule base, changes to the rule base (e.g., a rule's addition or removal) can result in unforeseen side effects. To be confident that a change will not introduce unintended side effects, the entire rule base needs to be analyzed. Therefore, the impact of the change is proportional to the change and the size of the system, which contains the complete rule base. According to NS theory, this is a CE. As a result, a firewall rule base containing rules other than disjoint rules is unstable under change.

2.3 Size of the Problem Space

For any given network comprised of clients and hosts, a firewall can contain many rules. In this section we will operationalize the size of the design space as the number of rules that could be defined on a firewall of a given network. The size of the design space thus logically provides insight into the problem space.

2.3.1 Formal Definitions of Rule Base Components

Let **N** represent a Layer 4 TCP/IP-based network in which two groups of network-connected resources can be defined:

- The hosts, providing network services via TCP/IP ports.
- The clients, requiring access to the services offered by the host.

Let **N** contain a firewall with rule base **F**, that is configured in such a way that only certain clients have access to certain services on certain hosts.

Let **Port** represent a Layer 4 TCP/IP defined port with the following attributes:

- Port.name = the name of the port.

- Port.protocol = the layer 4 TCP/IP protocol, being one of the following two values: TCP or UDP.
- Port.number = the number of the port, represented as an integer ranging from 1 to 2^{16}.

Let **P** represent the list of **Ports**, of length = pj .

$$\begin{cases} \mathbf{P}[1] \ldots \mathbf{P}[pj]. \\ \mathbf{P}[j] \text{ contains a } \textbf{Port}. \\ 1 \leq j \leq pj. \end{cases}$$

Let **Service** represent a network service accessible via a list of layer 4 TCP/IP ports, with the following attributes:

- **Service**.name = the name of the service.
- **Service**.ports = the list of ports = **P**.

Let **S** represent a list of **Services**, of length = sj.

$$\begin{cases} \mathbf{S}[1] \ldots \mathbf{S}[sj]. \\ \mathbf{S}[j] \text{ contains a } \textbf{Service}. \\ 1 \leq j \leq sj. \end{cases}$$

Let **Host** represent a network host that provides services, with the following attributes:

- **Host**.name = the Fully Qualified Domain Name (FQDN) of the network host.
- **Host**.IP = the IP address of the network host.

Let **H** represent a list of **Hosts**, of length = hj. The length of **H** is a function of the network **N**.

$$\begin{cases} \mathbf{H}[1] \ldots \mathbf{H}[hj]. \\ \mathbf{H}[j] \text{ contains a } \textbf{Host}. \\ 1 \leq j \leq hj. \\ hj = f_h(\mathbf{N}) \end{cases}$$

Let **Client** represent a network client that requires access to hosted services, with the following attributes:

- **Client**.name = the FQDN of the network client.
- **Client**.IP = the IP address of the network client.

Let **C** represent a list of **Clients**, of length = cj. The length of **C** is a function of the network **N**.

$$\begin{cases} \mathbf{C}[1] \ldots \mathbf{C}[cj]. \\ \mathbf{C}[j] \text{ contains a } \textbf{Client}. \\ 1 \leq j \leq cj. \\ cj = f_c(\mathbf{N}) \end{cases}$$

Let **R** represent a firewall rule, with the following attributes:

- **R**.Source = a list of Clients **Cs** of length = csj, where

  - $1 \leq$ csj$\leq$ cj
  - **Cs** $\subset$ **C**

- **R**.Destination = a list of Hosts **Hd** of length = hdj, where
  - $1 \leq$ hdj$\leq$ hj.
  - **Hd** $\subset$ **H**.
- **R**.Ports = a list of Ports = a Service **Sp**
  - where **Sp** $\in$ **S**$[sj]$.
- **R**.Action = either "Allow" or "Deny".

Let **F**, representing a list of rules **R** of length = fj, be the ordered firewall rule base **F**

- **F**[1] ... **F**[fj]
- **F**[m] contains a firewall rule **R**
- $1 \leq$ m $\leq$ fj
- **F** is order-sensitive. If **R**x is a firewall rule at location y in **F**, then the behavior of the firewall can be different if **R**x is located at position z instead of y, where z:1$\rightarrow$ fj and z $\neq$ y. Whether or not the behavior is different depends on the relation **R**x has with the other rules of **F**.

Example:
Suppose we have a rule base containing one rule **R**1:

- Source: 1.1.1.1
- Destination: 1.1.2.1
- Protocol/port: TCP/100-200
- Action: Allow

Suppose we are adding a rule **R**2:

- Source: 1.1.1.1
- Destination: 1.1.2.1
- Protocol/port: TCP/150
- Action: Deny

R2 is correlated to **R**1 as there is a full match in source and destination, an overlap in protocol/port, and difference in action.
If **R**2 is added before **R**1, then the rule **R**2 will execute the desired action (i.e., Deny). If **R**2 is added after **R**1, then the rule **R**2 will not have the desired effect, as **R**2 does not get activated.

2.3.2 Combinatorics

Ports

Port numbers are represented by 16-bit binary numbers and thus range from 1 to 2^{16}. Assuming that only TCP and UDP protocols are considered for OSI Layer 4 filtering, the possible number of values for Ports is equal to $2.2^{16} = 2^{17}$.

Services

S is the list of all possible services delivered via all ports exposed on the network **N**. $\mathbf{S_{max}}$ is the largest possible list of services, with length = $\mathrm{sj_{max}}$, in which all possible combinations of possible **Ports** are being used, where

$$sj_{\max} = \sum_{k=1}^{2^{17}} \binom{2^{17}}{k} \tag{2.1}$$

Hosts

The size of the list **H**, hj, is a function of the network **N** and is expressed as hj $= f_h(\mathbf{N})$. $\mathbf{H_{max}}$ is the list of all possible lists of hosts that are part of **H**. The length of this list is $\mathrm{hj_{max}}$, where

$$hj_{\max} = \sum_{a=1}^{hj} \binom{hj}{a} \tag{2.2}$$

and where hj $= f_h(\mathbf{N})$.

Services on Host

The maximum number of Hosts/Services combinations = $\mathrm{hj_{max}.sj_{max}}$ =

$$hj_{\max}.sj_{\max} = \left(\sum_{a=1}^{hj} \binom{hj}{a}\right) \cdot \left(\sum_{k=1}^{2^{17}} \binom{2^{17}}{k}\right) \tag{2.3}$$

where hj $= f_h(\mathbf{N})$.

Clients

The size of the list **C**, cj, is a function of the network **N**, and is expressed as cj $= f_c(\mathbf{N})$. $\mathbf{C_{max}}$ is the list of all possible lists of clients that are part of **C**. The length of this list is $\mathrm{cj_{max}}$ where

$$cj_{\max} = \sum_{a=1}^{cj} \binom{cj}{a} \tag{2.4}$$

where cj $= f_c(\mathbf{N})$.

Rules and Rule Base

In a rule **R**,

- **R**.Source can contain any element of $\mathbf{C_{max}}$.

- **R**.Destination can contain any element of $\mathbf{H_{max}}$.
- **R**.Ports can contain any element of $\mathbf{S_{max}}$.
- **R**.Action is the maximum number of action combinations, being 2 (“Allow” or “Deny”)

The firewall rule base $\mathbf{F_{max}}$ contains all possible rules that can be made with $\mathbf{C_{max}}$, $\mathbf{H_{max}}$ and $\mathbf{S_{max}}$

$$fj_{\text{max}} = 2.cj_{\text{max}}.hj_{\text{max}}.sj_{\text{max}} \tag{2.5}$$

$$fj_{\text{max}} = 2.\left(\sum_{a=1}^{cj}\binom{cj}{a}\right)\cdot\left(\sum_{a=1}^{hj}\binom{hj}{a}\right)\cdot\left(\sum_{k=1}^{2^{17}}\binom{2^{17}}{k}\right) \tag{2.6}$$

where cj = $f_c(\mathbf{N})$ and hj = $f_h(\mathbf{N})$

Example: Suppose we have a network containing ten clients (cj = 10), five servers (hj = 5), and on each server, five services are defined (sj=5). According to the above combinatorics, the solution space has a size of 1.196.206 rules.

$\mathbf{F_{max}}$ is the possible design space for a rule base, and has a phenomenal size. It represents the number of rules you can choose from when designing a filtering action on a firewall. Multiple rules can deliver one filtering requirement. Choosing the optimally evolvable rule is a considerable challenge. As the network grows and $f_c(\mathbf{N})$ and $f_h(\mathbf{N})$ grow, choosing the optimally evolvable firewall rule from the design space becomes even more difficult. The design space must be consciously reduced by means of design restrictions.

2.4 Ontological Model of a Firewall Rule Base

In this section we propose an ontological model for a firewall rule base. The model will contain the essential information required to define a filtering rule. Such a model will help us in subsequent sections to compare the ontological model (by default independent of implementation) with the implementation of such model in firewalls. Analysing the ontological model and implementation model will help us in defining the evolvability issues of the firewall rule base.

2.4.1 The Ontological Model

The analysis of a firewall rule base’s ontology is undertaken employing the DEMO FACT model [44]. This simple, straightforward ontological modeling method sufficiently reveals the issues we intend to illuminate. In terms of the scope of a security department managing the firewalls, the “create firewall rule” transaction is considered an ontological transaction. During the request phase, information must be provided to the executor, corresponding with the filtering objective: source, destination, service, and action.

The ontology of a rule base is expressed in a DEMO FACT model, as shown in Figure 2.2. For simplicity of explanation, details regarding value types and the transactions related to the coming about of facts and products are excluded. There are multiple ways to identify a source or a destination depending on whether the network resource(s) are identified by their IP address(es) (IPRESOURCE, IPRESOURCEGROUP)

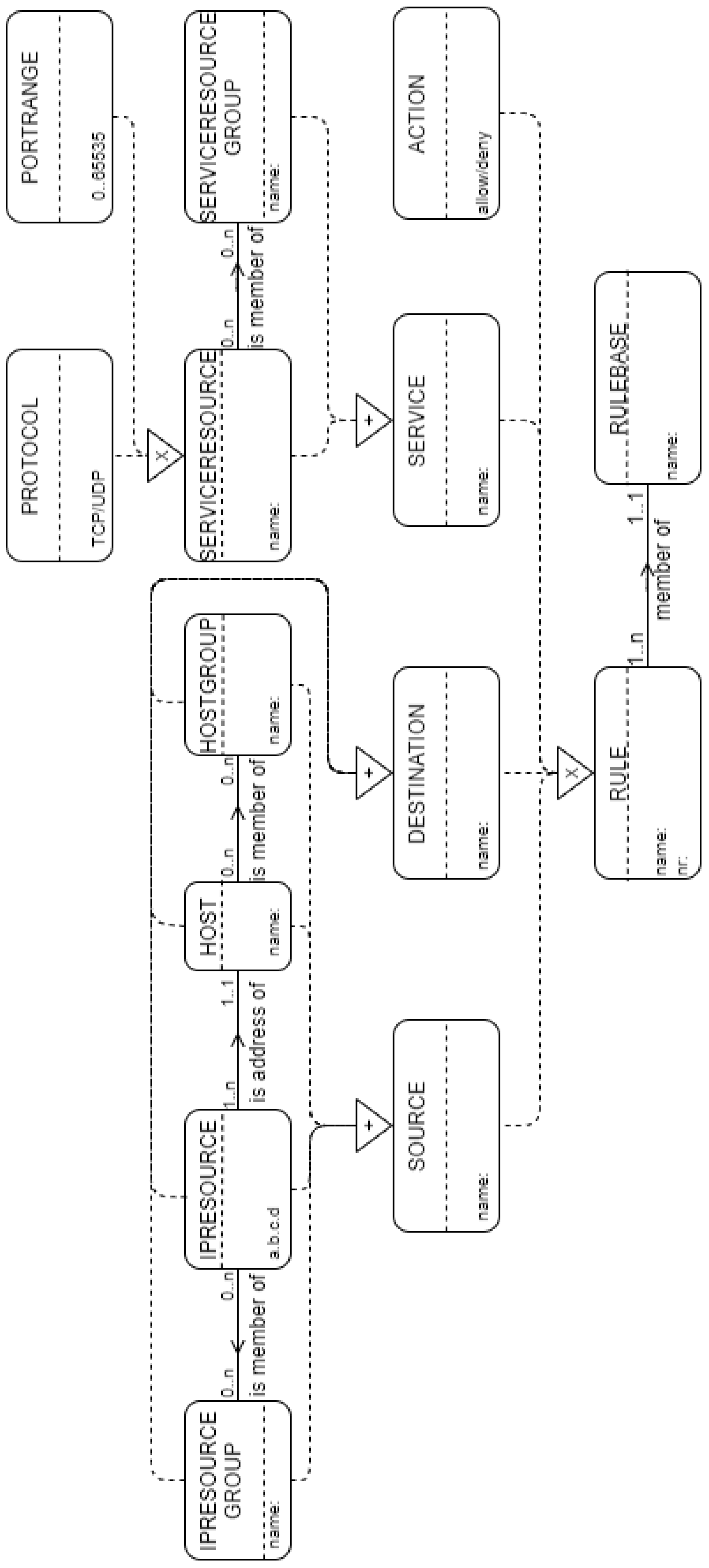


FIGURE 2.2: Ontological FACT model of a firewall rule base

or by their logical name(s) (HOST, HOSTGROUP). SOURCE and DESTINATION are generalizations of those different resource identifications. Note that the same network resource can be both a source and a destination.
A similar reasoning holds for SERVICERESOURCE and SERVICERESOURCEGROUP. A SERVICERESOURCE is the aggregation/Cartesian product of PROTOCOL and PORTRANGE. A RULE is the aggregation/Cartesian product of the different entity types comprising a rule.

2.5 Reverse-Engineering the Implementation Model

An ontology is an implementation independent model and needs to be translated towards a technology-implementable solution. Certain design decisions must be made during this conversion. The FACT model shown in Figure 2.2 contains generalization relations or (inverse) inheritance relations. The manner by which inheritance relations are translated towards an implementation can profoundly impact the evolvability of the implementation model [45].

Firewall vendors neither share nor publish the firewall's internal data model explicitly. We will reverse-engineer the implemented data model based on firewall configuration exports. Exports were made from three different firewalls types used inside the Engie network.

Three additional firewalls, built by industry leaders, shall be discussed:

- Fortinet
- Palo Alto
- Check Point

Based on these exports, we attempt to reverse-engineer the firewall data model. For simplicity of illustration, we will focus exclusively on the TCP/UDP protocols and exclude export information of other protocols of the OSI Layer 4 (e.g., ICMP).

2.5.1 The Fortinet Firewall

Service Objects

The Service objects export contains a data definition header. The relevant fields are: Name (reference name), Protocol (TCP or UDP), and Destination Port (one or more ports or port ranges). Our analysis of the various objects yields the following observations:

- No formal naming convention for Service objects.
- No unique Service definitions.
- No disjoint Service definitions.

ServiceGroup Objects

The ServiceGroup objects export contains a data definition header. The relevant fields are: Name (reference name) and Services (one or more Service objects). Our analysis of the various objects yields the following observations:

- No formal naming convention for ServiceGroup objects.
- No unique ServiceGroup definitions.
- No disjoint ServiceGroup definitions.
- As ServiceGroups are an aggregation of services, ServiceGroup and Service objects are by design not disjoint.

Address Objects

The Address objects export contains a data definition header. The relevant fields are: Name (reference name), Type (a Fully Qualified Domain Name (= DNS-resolvable name) or IP Netmask (/32 for a single IP or a /x for a range of IP addresses) or an IP range (from a.b.c.d to a.b.c.f) and Address (the address according to one of the three types). Our analysis of the various objects yields the following observations:

- No formal naming convention for Address objects.
- No unique Address definitions.
- No disjoint Address definitions.

AddressGroup Objects

The AddressGroup objects export contains a data definition header. The relevant fields of this header are: Name (reference name), Address (an Address Object). Our analysis of the various objects yields the following observations:

- No formal naming convention for AddressGroup objects.
- No unique AddressGroup definitions.
- No disjoint AddressGroup definitions.
- As AddressGroups are an aggregation of addresses, AddressGroup and Address objects are, by definition, not disjoint.

Rule Objects

The Rule objects export contains a data definition header. The relevant fields are: Nr (location in the rule base (1 = top)), Name (reference name), SourceAddress (an Address object and/or an AddressGroup object referenced via its Name), DestinationAddress (an Address object and/or an AddressGroup object referenced via its Name), Services (a Services object and/or ServiceGroup object referenced via its Name), Action (Allow or Deny). Our analysis of the various objects yields the following observations:

- Source- and DestinationAddress: Source- and DestinationAddress objects contain AddressGroup objects and/or Address objects.
- Source- and DestinationAddress: AddressGroup objects or Address objects can be used for both SourceAddress and DestinationAddress.
- The rule base allows both Allow and Deny rules at any location in the rule base. The firewall approach enforces neither the usage of a white-list (rule base only contains "Allow" rules and a default "Deny" rule at the end) nor a black-list (rule base only contains "Deny" rules and a default "Allow" rule at the end).

Implementation Model of a Fortinet Firewall

Based on the firewall export, we can reverse-engineer the implementation model of the Fortinet firewall. We find that:

- The ontological concepts PROTOCOL and PORTRANGE have no equivalent in the Fortinet implementation model. They are directly aggregated in the Fortinet Service object.
- The ontological concept of SERVICERESOURCE does not exist. It is absorbed in the Fortinet Service object.
- The ontological concept of SERVICERESOURCEGROUP does not exist given that the ontological concept of SERVICERESOURCE does not exist.
- The Fortinet object ServiceGroup aggregates Service objects. This is not the same as the relation between the ontological concepts SERVICERESOURCE and SERVICERESOURCEGROUP. SERVICERESOURCE is a combination of a PORTRANGE and a PROTOCOL instance and a SERVICERESOURCEGROUP instance is a collection of SERVICERESOURCEs. Conversely, the Service object aggregates PORTRANGE, PROTOCOL, SERVICERESOURCE and SERVICERESOURCEGROUP. The ServiceGroup object is a collection of Service objects.
- The ontological concepts SOURCE and DESTINATION do not exist. They are directly aggregated in the Rule object.
- The ontological concepts IPRESOURCE, IPRESOURCEGROUP, HOST and HOSTGROUP do not exist. The implementation object Address aggregates IPRESOURCE and HOST, while the implementation object AddressGroup is a collection of Address objects and thus an aggregation of both HOSTGROUP and IPRESOURCEGROUP.
- An instance of the ontological concept of RULE is made up of an instance of a SOURCE, a DESTINATION, and a SERVICE. In the Implementation Rule object, the source is defined in the rule, and not first outside the rule via the ontological SOURCE concept.
- Looking at the implementation objects Address and AddressGroup, one cannot determine whether or not those play the role of sources or destinations in rules.

Figure 2.3 represents the derived implementation model as a set of tables and relationships between the tables. The various objects found in the firewall export correspond to rows within these tables.

2.5.2 The Palo Alto Firewall

Service Objects

The Service objects export contains a data definition header. The relevant fields are: Name (reference name), Protocol (TCP or UDP), and Destination Port (one or more ports or port ranges). Our analysis of the various objects yields the following observations:

- No formal naming convention for Service objects.

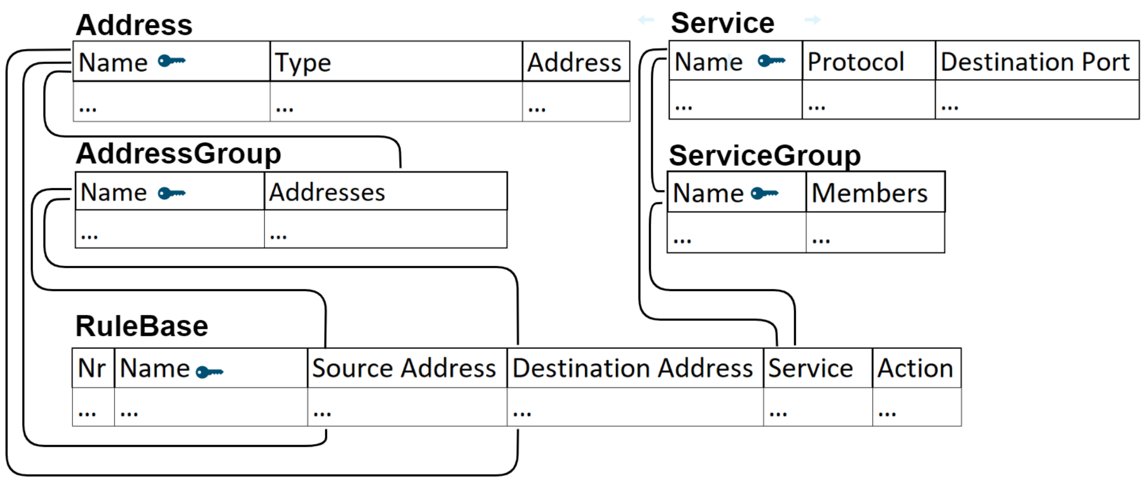


FIGURE 2.3: Implementation model of a Fortinet firewall

- No unique Service definitions.
- No disjoint Service definitions.

ServiceGroup Objects

The ServiceGroup objects export contains a data definition header. The relevant fields of this header are: Name (reference name) and Services (one or more Service objects). Our analysis of the various objects yields the following observations:

- No formal naming convention for ServiceGroup objects.
- No unique ServiceGroup definitions.
- No disjoint ServiceGroup definitions.
- As ServiceGroups are an aggregation of services, ServiceGroup and Service objects are, by definition, non disjoint.

Application Objects

The Application objects export contains a data definition header. The relevant fields are: Name (reference name), Protocol (TCP or UDP), and Destination Port (one or more ports or port ranges). Our analysis of the various objects yields the following observations:

- The list of Application objects is a standard industry list that is a component of the firewall default configuration. It contains the names of the known applications and the standard assigned port to the applications. Example: Application = Minecraft, Protocol = TCP 80, 443, 25565.
- In addition to Name and Protocol, each application is assigned to a category, subcategory, technology, and risk level. Example: Application = Minecraft, Category = media, Subcategory = game, Technology = browser-based, risk level = 2.
- The list is editable, enabling the addition of custom-developed applications or non-standard usage of ports.

- No formal naming convention for Application objects.
- Different application can use the same ports.
- Application objects are not disjoint with respect to Service definitions.

ApplicationGroup Objects

The ApplicationGroup objects export contains a data definition header. The relevant fields are: Name (reference name) and Applications (one or more Application Object names). Our analysis of the various objects yields the following observations:

- This list is customer created and aggregates applications into a meaningful package. Example: a group is created that contains all individual applications required for proper active directory function (dsn, kerberos, ntp, etc.).
- The members of this group are applications from the Application objects list.
- No formal naming convention for ApplicationGroup objects.
- No unique ApplicationGroup definitions.
- No disjoint ApplicationGroup definitions.
- No disjoint ApplicationGroup and ServiceGroup definitions.
- As ApplicationGroups are an aggregation of Applications, ApplicationGroup and Application objects are by design overlapping.

Address Objects

The Address objects export contains a data definition header. The relevant fields are: Name (reference name), Type (a Fully Qualified Domain Name (= DNS-resolvable name) or IP Netmask (/32 for a single IP or a /x for a range of IP addresses) or an IP range (from a.b.c.d to a.b.c.f) and Address (the address according to one of the three types). Our analysis of the various objects yields the following observations:

- No formal naming convention for Address objects.
- No unique Address definitions.
- No disjoint Address definitions.

AddressGroup Objects

The AddressGroup objects export contains a data definition header. The relevant fields are: Name (reference name), Address (an Address Object). Our analysis of the various objects yields the following observations:

- No formal naming convention for AddressGroup objects.
- No unique AddressGroup definitions.
- No disjoint AddressGroup definitions.
- As AddressGroups are an aggregation of addresses, AddressGroup and Address objects are, by definition, overlapping.

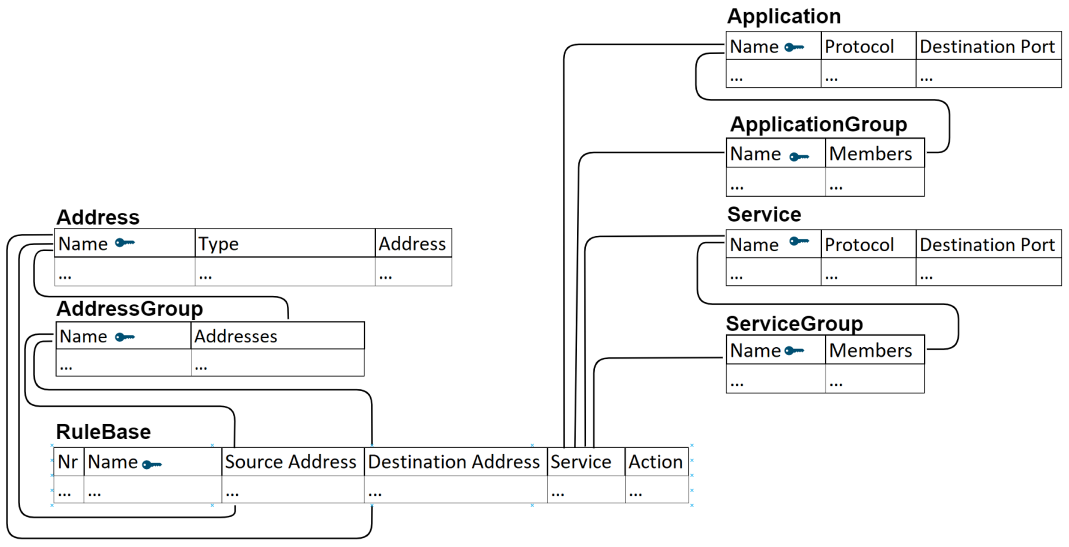


FIGURE 2.4: Implementation model of a Palo Alto firewall

Rule Objects

The Rule objects export contains a data definition header. The relevant fields are: Nr (location in the rule base (1 = top), Name (reference name), SourceAddress (an Address object and/or an AddressGroup object referenced via its Name), DestinationAddress (an Address object and/or a AddressGroup object referenced via its Name), Services (a Services object and/or ServiceGroup object referenced via its Name), Application (an Application object and/or ApplicationGroup object referenced via its name, Action (Allow or Deny). Our analysis of the various objects yields the following observations:

- Source- and DestinationAddress: Source- and DestinationAddress contain AddressGroup objects and/or Address objects.
- Source- and DestinationAddress: AddressGroup objects or Address objects can be used for both SourceAddress and DestinationAddress.
- The rule base allows both Allow and Deny rules at any location in the rule base. The firewall enforces neither the usage of a white-list nor a black-list approach.
- When the Service field contains the value "Application-Default", the Application field is used to determine the protocol and port.

Implementation Model of a Palo Alto Firewall

Based on the firewall export, we can reverse-engineer the implementation model of the Palo Alto firewall. We find that:

- The ontological concepts PROTOCOL and PORTRANGE have no equivalent in the Palo Alto implementation model. They are directly aggregated in the Palo Alto Service and Application objects.
- The ontological concept of SERVICERESOURCE does not exist. It is encompassed by Palo Alto Service and Application objects.

- The ontological concept of SERVICERESOURCEGROUP does not exist, given that the ontological concept of SERVICERESOURCE does not exist.
- The Palo Alto ServiceGroup object aggregates Service objects and ApplicationGroup Application objects.
- The Palo Alto Service and Application objects both represent the ontological concept of SERVICERESOURCE.
- The ontological concepts SOURCE and DESTINATION do not exist. They are directly aggregated in the Rule object.
- The ontological concepts IPRESOURCE, IPRESOURCEGROUP, HOST and HOSTGROUP do not exist. The implementation object Address aggregates IPRESOURCE and HOST, while the implementation object AddressGroup is a collection of Address objects and thus an aggregation of both HOSTGROUP and IPRESOURCEGROUP.
- An instance of the ontological concept of RULE is comprised of an instance of a SOURCE, a DESTINATION and a SERVICE. In the implementation Rule object, the source is defined in the rule, and not first outside the rule via the ontological SOURCE concept.
- Examining the implementation objects Address and AddressGroup, one cannot determine whether or not those play the role of sources or destinations in rules.

Figure 2.4 shows the derived implementation model as a set of tables and relationships between the tables. The various objects located in the firewall export map correspond to rows within these tables.

2.5.3 The Check Point Firewall

Service<Protocol> Objects

The Service objects export is comprised of two sub exports: ServiceTCP and ServiceUDP. The sub exports have a data definition header. The relevant fields of this header are: Name (reference name), Type (TCP/UDP), and Destination Port (one or more ports or port ranges). Our analysis of the various objects yields the following observations:

- The Protocol field is pre-filled with TCP or UDP, depending on membership of ServiceTCP or ServiceUDP.
- No formal naming convention for Service objects.
- No unique Service definitions.
- No disjoint Service definitions.

ServiceGroup Objects

The ServiceGroup objects export contains a data definition header. The relevant fields are: Name (reference name) and Services (one or more Service objects) Our analysis of the various objects yields the following observations:

- No formal naming convention for ServiceGroup objects.
- No unique ServiceGroup definitions.
- No disjoint ServiceGroup definitions
- As ServiceGroups are an aggregation of services, ServiceGroups and Service objects are, by definition, not disjoint.

Address Objects

The Address objects export contains three sub objects and subsequent exports: Host, Network and AddressRange. The sub-exports have Name (reference name) as a common field. Host is a sub-object with IP as an attribute. Network is a sub-object with network range and subnetmask as attributes. AddressRange is a sub-object with startIP and endIP as attributes. Our analysis of the various objects yields the following observations:

- No formal naming convention for Host, Network, or AddressRange objects.
- No unique Host, Network, or AddressRange definitions.
- No disjoint Address (Host, Network, AddressRange) definitions.

Group Objects

The Group objects are aggregations of the three different Address objects (Network, AddressRange, Host). The export contains a data definition header. The relevant fields are: Name (reference name), members (a Host, Network or Address Range object referenced by name). Our analysis of the various objects yields the following observations:

- No formal naming convention for Group objects.
- No unique Group definitions.
- No disjoint Group definitions.
- As Groups are an aggregation of hosts and networks, Group, Host, Network and AddressRange objects are, by definition, not disjoint.

Rule Objects

The Rule objects export contains a data definition header. The relevant fields of this header are: Nr (location in the rule base (1 = top), Name (reference name), SourceAddress (an Address object and/or a Group object referenced via its Name), DestinationAddress (an Address object and/or a Group object referenced via its Name), Services (a Services object and/or ServiceGroup object referenced via its Name), Action (Allow or Deny). Our analysis of the various objects yields the following observations:

- Source- and DestinationAddress: Source- and Destination Address contain Group objects and/or Address objects.
- Source- and DestinationAddress: Group objects and/or Address objects can be used for both SourceAddress and DestinationAddress.

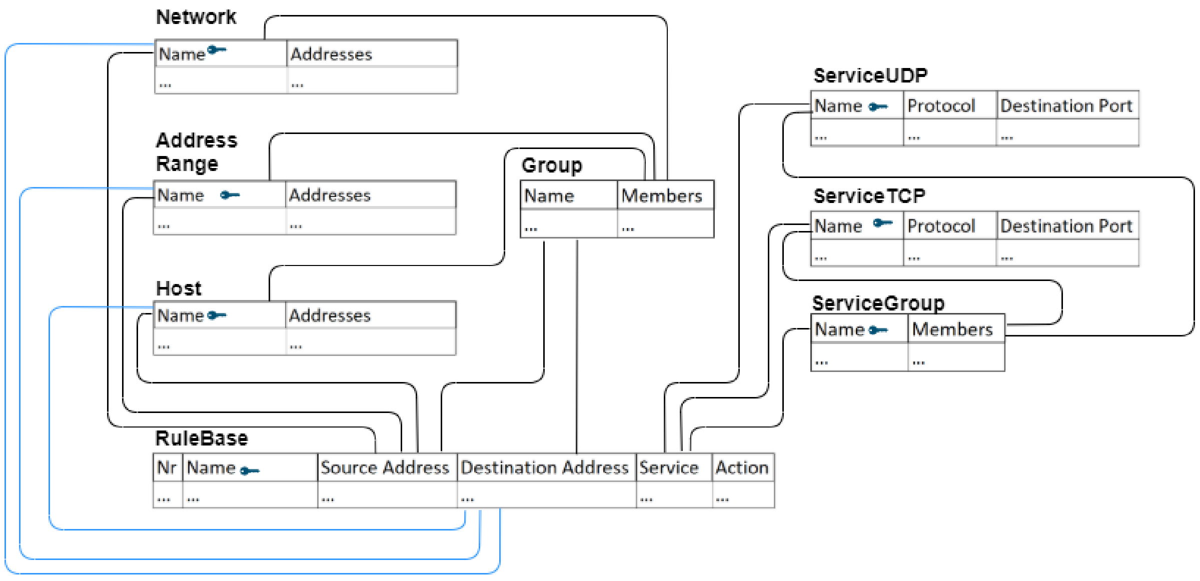


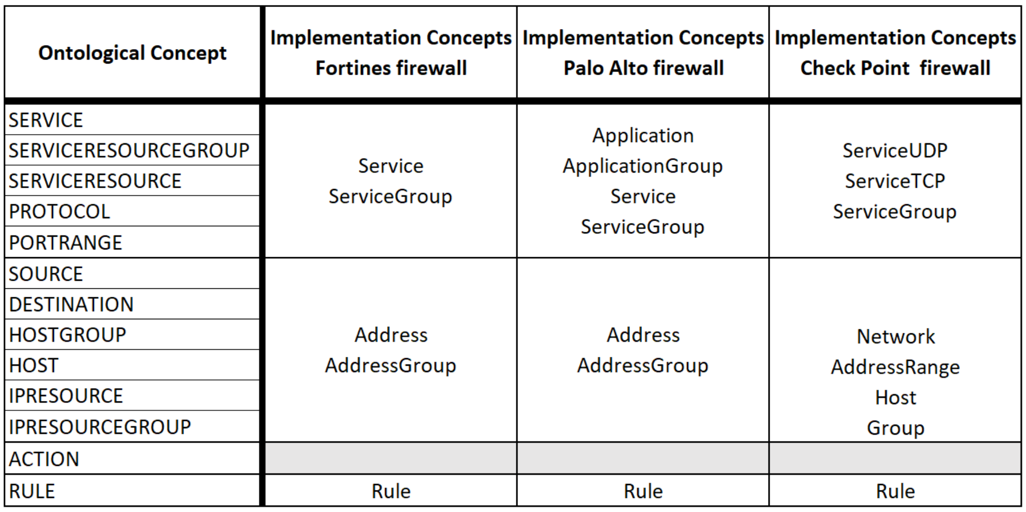
FIGURE 2.5: Implementation model of a Check Point firewall

- The rule base contains both Allow and Deny rules, mixing a white-list and black-list approach.

Implementation Model of a Check Point Firewall

Based on the firewall export, we can reverse-engineer the implementation model of the Check Point firewall. We find that:

- The ontological concepts PROTOCOL and PORTRANGE are implemented as ServiceUDP and ServiceTCP objects on the Check Point
- The ontological concept of SERVICERESOURCE does not exist. It is encompassed by the Check Point ServiceGroup object.
- The ontological concept of SERVICE is implemented as the ServiceGroup object.
- The ontological concept of SERVICERESOURCEGROUP is also implemented as the ServiceGroup object.
- The Check Point ServiceGroup object, combines both SERVICESOURCE and SERVICERESOURCEGROUP.
- The ontological concepts SOURCE and DESTINATION do not exist. They are directly aggregated in the Rule object.
- The ontological concepts IPRESOURCE, IPRESOURCEGROUP are implemented as the Network and AddressRange objects.
- The ontological concepts of HOST and HOSTGROUP are implemented as Host and Group objects.
- An instance of the ontological concept of RULE is comprised of an instance of a SOURCE, a DESTINATION, and a SERVICE. In the implementation Rule object, the source is defined in the rule, and not first outside the rule via the ontological SOURCE concept.



Ontological Concept	Implementation Concepts Fortines firewall	Implementation Concepts Palo Alto firewall	Implementation Concepts Check Point firewall
SERVICE	Service ServiceGroup	Application ApplicationGroup Service ServiceGroup	ServiceUDP ServiceTCP ServiceGroup
SERVICERESOURCEGROUP			
SERVICERESOURCE			
PROTOCOL			
PORTRANGE			
SOURCE	Address AddressGroup	Address AddressGroup	Network AddressRange Host Group
DESTINATION			
HOSTGROUP			
HOST			
IPRESOURCE			
IPRESOURCEGROUP			
ACTION			
RULE	Rule	Rule	Rule

FIGURE 2.6: Overview of ontological and implementation concepts

- When considering the implementation objects Network, AddressRange, Host and Group, one is unable to determine whether or not these play the role of sources or destinations in rules.

2.5.4 Overview Implementation Models

Figure 2.6 provides an overview of the ontological concepts and implementation concepts.

2.6 Evolvability Issues Due to the Data Model

According to NS theory, a modular structure will be free of CE with respect to a set of anticipated changes when each module respects the four NS theorems – SoC, SoS, AVT, and DVT. The SoC condition is the most relevant in our case.

From Section 2.4 we may deduce that a source, a destination, and a service can be specified in multiple ways as they are generalizations of other types. At the time of instantiation, the same logical concept of a network resource can be represented by different objects, thus resulting in overlap(s). This overlap could lead to non-disjoint rules. The ontology does not include restrictions that would favor non-disjoint rule creation. When implementation follows this ontology, it would be expected that no restrictions will be present to avoid the creation of non-disjoint rules, meaning that there already exists something fundamentally wrong with the ontological model.

From Section 2.5 we may deduce that SoC between the different rule base objects is not respected. In a Fortinet firewall, Service and ServiceGroup address the same concern, given that they form a protocol/port(s) pair. In a Palo Alto firewall, this concern is represented by Service, ServiceGroup, Application, and ApplicationGroup. The Check Point respects a separation between various protocols via the usage of ServiceTCP and ServiceUDP, but allows violation of SoC via ServiceGroup. In a Fortinet and Palo Alto firewall, Address and AddressGroup represent the same concern, given that they refer to network resources. The different types of network resources, i.e. a host, a network and an address range, are aggregated into Address and AddressGroup. The Check Point firewall separates the network resource types into Host, Network and AddressRange sub-objects, but aggregates them in Group. Within a rule, the concern of being a source or a destination, is combined into Address and AddressGroup for the Fortinet and Palo Alto firewalls, and in Host, Network, AddressRange, and Group in the Check Point firewall.

The disrespect of SoC represents a form of coupling inside the rule base that becomes visible when the model is being instantiated. Different objects can represent the same thing, requiring a detailed analysis of all rules in the rule base containing those objects. Only then can we understand the context and decide on how to reflect the desired change. All the firewall objects aggregate in the rule objects. Seemingly different rules could, in fact, address the same source, destination, and service, as these can be made with different objects that actually represent the same thing.

The naming of the Service, ServiceGroup, Address, AddressGroup, etc. objects is independent of the objects' content. They are considered as two concerns that can evolve independently from one other. Similar to the best practice within the field of programming to assign names to variables that are anthropomorphic to what they represent, so should the naming convention of rule-based objects be anthropomorphic to their content. If name and content deviate from one another, incorrect objects can be chosen to make rules, thus creating incorrect rules. The separation of naming and content results in objects with different names, yet both representing the same item. Changes to the resource, such as a new IP, need to be propagated to all objects representing that resource. As one cannot deduce this from the design of the objects, it means that at run-time, it will be necessary to search through the rule base objects in order to determine the impact. The change will ripple through the system and is a function of the size of the system; thus, it is a CE.

The implemented rule base's design violates SoC in two ways: 1) it combines different concerns, thus introducing additional coupling, and 2) it decomposes concerns, thus violating cohesion. As changes to names, IP masks, IP ranges, ports, and protocols are made over time, resulting violations of SoC will lead to ripple effects in the configuration of the firewall rule base. These can result in non-disjoint rules and become CEs with respect to the addition and deletion of rules in the rule base.

Given the above, at this point, we may conclude that neither the ontological model nor the implementation model of firewalls facilitates the creation of disjoint rules.

2.7 Problem Overview

From the previous sections, we may conclude that:

- Relationships between rules, in combination with the inherent design of the firewall whereby its rule base is order-sensitive, result in anomalies.
- The ontological model of a firewall rule base, based on actual information provided when a firewall rule is created, contains elements that do not favor the establishment of an evolvable rule base.
- The current implementation models of three major firewall vendors do not contain mechanisms that favor the evolvability of the rule base.

- Neither the ontological model nor the implementation model embeds explicit design criteria and conventions. As a result, any rule from the full design space may be used without favoring rules that lead to an evolvable rule base.

Chapter 3

Artifact Requirements

In this chapter, we explain the different requirements related to building an evolvable rule base.
In Section 3.1 we elaborate on the need to eliminate the impact of order sensitivity in a rule base, and in Section 3.2 we explain the need for strict naming conventions of the group objects. Although we anticipate a fine-grained rule base, we nonetheless need to limit the number of rules. We elaborate on minimizing the number of rules in Section 3.3. In Section 3.4 we provide a rationale for the need to generate a rule base that is compliant with the Zero Trust security policy. Section 3.5 provides a rationale for the need for a green-field artifact. The green-field artifact is a set of design criteria for the creation of an evolvable rule base in accordance with previous requirements, while in Section 3.6 we express the need for a brown-field artifact that will convert a non-evolvable rule base into an evolvable rule-base. Section 3.7 summarizes all the requirements.

3.1 Eliminating Order Sensitivity

In Section 2.2, we have shown that rules with relationships other then disjoint or partially disjoint can result in rule base anomalies. We can thus only allow disjoint and partial disjoint rules in our rule base (hereon referred to as disjoint rules). By exclusively allowing these kind of rules, we eliminate the order sensitivity of the rule base. Rules can be added and removed anywhere in the rule base, without concern for unintended negative consequences. This is the most important design requirement for the artifact.

3.2 Group Object Naming Conventions

In Section 2.5, we have shown that the inconsistent naming of group objects has a negative impact on evolvability. While the introduction of group objects is intended to improve manageability (names compared to numbers), without proper naming conventions, additional complexity is easily introduced. The same item (source, destination, service) being represented by different group objects, or different items (source, destination, service) being represented by nearly the same group objects (case sensitivity of names, spelling errors, different delimiters), can result in invoking the wrong objects when making a rule, or introduce configuration drift between rules, eventually resulting in rules that do not match their intent.

The artifact must impose naming conventions such that group objects can only address one concern and all ambiguity is removed as to when to use a group. This will eliminate possible configuration drift (different groups, same concern, different content).

3.3 Minimize the Number of Rules in the Rule Base

In Section 2.1, we have shown that, both in the literature and in industry reports, there has been a focus in minimizing the rule base. It makes perfect sense to minimize the number of rules, since fewer rules means more efficient filtering. However, compacting a rule base can result in a non-evolvable rule base, in which it becomes increasingly difficult to add/remove/change rules. There is also a risk that a rule no longer matches its original intent but rather becomes an aggregation and combination of multiple intents. The function-construction gap increases, resulting in rule base complexity, where it becomes difficult to interpret the system by looking at its configuration.

Removing anomalies and making disjoint rules will lead to more rules. As each concern is allocated its own discrete rule, a more fine-grained rule base will emerge — an outcome which NS theory predicts. While this is positive with respect to evolvability, doing so may also introduce negative impacts to firewall performance.

In Section 2.3.2, the design space of a firewall in a specific context was proposed. This design space includes both the rules that do and do not favor evolvability. Of all the rules that favor evolvability, there is a subset that will always work but which maximizes size: a rule for each client, each host, each service, and each action. It is also admissible to have a subset that aggregates some clients, some hosts, and some services, while still being disjoint from each other and respecting the same filtering logic. This subset will necessarily be smaller than the previous subset. As it is smaller, we consider it more desirable for performance reasons. Hence the need to have an evolvable rule base of a minimal size as a design criteria.

3.4 Zero Trust

A rule has three filtering parameters: source, destination, and service. The same filtering result may be accomplished by different configurations of those three parameters. In Section 2.3.2 we presented the combinatorics underlying this as well as the difficulty of selecting rules that favor evolvability in the absence of design criteria.

Forrester advocates the usage of a Zero Trust (ZT) model [14, 15] [43], since design criteria:

- Ensure all resources are accessed securely, regardless of location and hosting model,
- Adapt a "least privilege" strategy and strictly enforce access control,
- Inspect and log all traffic for suspicious activity.

The working assumption in the case of protecting network-connected resources is that all traffic towards those resources is considered a threat and must be inspected and secured. A network-connected resource should only expose those services via the network, which are minimally required. Additionally, each network-connected resource should be allowed access only on an as-needed basis.

A network-connected resource can take various forms. It can be defined as a single machine (host) with a unique IP address, a VLAN consisting of a range of IP addresses, or an IP subnet representing a whole address space. In the ontological model discussed in Section 2.4 we see this as well. The type of network-connected resource being protected determines the optimal type of rule. The definition of the destination as either a host, VLAN or subnet, in combination with the delivered services, will enable the expression of the allowable access.

The artifact must account for a "Zero Trust" strategy: rules in the rule base express the explicit granting of access to a resource. If there is no rule in the rule base that explicitly grants access, then access should be denied. This kind of rule base is also known as a white-list. Not on the list = Access Denied. The rule base will only contain rules that have "Allow" as a filtering action, with the exception of the final rule of the rule base. The final rule is the default "Deny" rule. If no matching rule is found, traffic is not allowed.

The opposite of a white-list is a black-list. In such a list, the end of the rule base contains the default "Allow" rule and all preceding rules specify all the Deny rules.

Network-connected resources normally offer far fewer services relative to the maximum amount of 2^{17} (131.072) possible services . If a ZT policy is applied, it makes more sense to expressly provide access via a white-list instead of a black-list, given that a white-list will contain fewer rules as compared to a black-list.

3.5 Requirements for a Green-Field Artifact

In the existing firewall literature, we found no set of design criteria beyond "minimize your rule base, minimize overlap". Our objective is to provide greater specificity, as well as to provide design criteria that may be systematically applied, independent of rule base size. If we were to build a firewall rule base from scratch from a green-field, we could follow these design criteria and be confident of having an evolvable rule base with respect to a set of anticipated changes, such as the addition, removal, and modification of rules. We call this the green-field artifact and it must adhere to all previous expressed requirements.

3.6 Requirements for a Brown-Field Artifact

A green-field artifact is a rare luxury. The "swamp of practice" consists of firewalls that present a variety of evolvability issues. In small rule bases, non-disjoint rules are not overly concerning. Where rule bases are small, it is possible for the firewall administrator to oversee the problem and reshuffle the rules as required. As rules are added, however, they become increasingly difficult to manage. As system size increases, the undesired positive feedback mechanism of non-disjoint rules is exacerbated, ultimately resulting in increasingly unstable systems with each newly-added rule.

Our objective is to rejuvenate the rule bases of those firewalls, converting them into a rule base that adheres to the green-field design criteria and that from that point on can be managed and will continue to evolve in line with the green-field artifact. We call this the brown-field algorithm and it must adhere to all previous expressed requirements.

The existing literature includes examples of algorithms that will look for and resolve anomalies. However, there are no examples of algorithms that also install the criteria to keep the rule base free of anomalies. Instead, every change to the rule base requires a full re-analysis of the rule base and a corresponding resolution of anomalies introduced by the new rule.

3.7 Requirements Overview

Based on all that has been discussed in previous sections, we may summarize the requirements for our artifacts as follows:

- R1: Only fully-disjoint and partial-disjoint rules.
- R2: A consistent naming convention for all firewall objects.
- R3: Minimization of the number of rules, while maximizing evolvability.
- R4: Application of a white-list approach.
- R5: Creation of a green-field artifact, consisting of design rules for an evolvable rule base, adhering to R1 through R4.
- R6: Creation of a brown-field artifact, converting an existing non-evolvable rule base into a rule base that responds to the green-field artifact specifications (R5).

Chapter 4

Green-Field Artifact Creation and Demonstration

Based on the artifact requirements outlined in the previous chapter, the present chapter discusses the design of the green-field artifact. In Section 4.1, we convert the requirements into design criteria, which are then applied to the green-field artifact - a method that is presented in Section 4.2. In Section 4.3, we demonstrate the artifact by applying a set of anticipated changes to a rule and rule base and confirm whether or not they result in CEs. The findings of this chapter have been published in [27] and [28].

4.1 Designing an Evolvable Rule Base

From Chapter 2 we know that: a firewall rule base is order-sensitive; relationships between rules are possible and represent coupling; disjoint rules are not order-sensitive with respect to each other; if all rules in the rule base are disjoint and a new disjoint rule is added, then this rule can be added at any location in the rule base.

If the entire firewall rule base needs to be investigated to determine whether or not a rule is disjoint to all existing rules, then a CE is being introduced at run-time. Introducing a new rule to, or removing a rule from the system should result in work that is proportional to the newly-required functionality and not into work that has no logical link to the required functionality and that requires searching throughout the entire rule base. According to NS theory, the impact of the change should be proportional to the nature of the change itself, and not proportional to the system to which the change is applied.

Disjoint rules have no overlap in source or destination or services (combination of ports). The following combinations are possible:

- No overlap in sources - overlaps of destinations and services are ignored.
- No overlap in destinations - overlaps of sources and services are ignored.
- No overlap in ports - overlaps of sources and destinations are ignored.
- No overlap in source-destination combination - overlaps of services are ignored.
- No overlap in source-ports combinations - overlaps of destinations are ignored.

- No overlap in destination-ports combinations - overlaps of sources are ignored.
- No overlap in source-destination-services combination.

Consider a rule **R** comprised of the following components:

- **Cs** representing the Source, where $\mathbf{Cs} \subset \mathbf{C_{max}}$.
- **Hd** representing the Destination, where $\mathbf{Hd} \subset \mathbf{H_{max}}$.
- **Sp** representing the Ports, where $\mathbf{Sp} \in \mathbf{S_{max}}$.
- Action is to be "Allow" as each rule in the rule base explicitly provides access to allowed services on allowed hosts.
- **R** = (**Cs**, **Hd**, **Sp**, "Allow')

Cs is $f_c(\mathbf{N})$ and **Hd** is $f_h(\mathbf{N})$. The network **N**, the hosts **Hd** and the clients **Cs**, are context-dependent. We are unable to generalize from those variables. Attempting to structure **Cs** and **Hd** in a manner that allows for disjoint rules starting from this variable will not yield anything useful. On the other hand, **Sp** represents the ports. The number of ports is bound and the values are generic over all possible rule bases. We can now calculate the combinatorics related to port. Out of all possible port combinations, we are only interested in those port/service groups that do not overlap (i.e., are disjoint).

Let us consciously restrict **Sp** to **Su**, such that **Su** exclusively contains disjoint services.

$$\begin{cases} \exists!\mathbf{Su}[m] \text{ in } \mathbf{Su} \text{ for m:1}\rightarrow\text{suj.} \\ \mathbf{Su}[u] \cap \mathbf{Su}[v] = \emptyset\text{, where u, v:1}\rightarrow\text{suj, and u} \neq \text{v} \end{cases}$$

If each service is represented by 1 port, **Su** will contain 2^{17} elements, which is the maximum size of **Su** in this restricted case.

The service **Su**[m] can be delivered by different hosts.
Let $\mathbf{Hd_{Su[m]}}$ represent the list of hosts that offer service **Su**[m].

$$\begin{cases} \mathbf{Hd_{Su[m]}} \subset \mathbf{Hmax} \text{ and } \mathbf{Hd_{Su[m]}}[x] \text{ contains a single host.} \\ \mathbf{Hd_{Su[m]}} \text{ contains unique and disjoint elements.} \\ \exists!\mathbf{Hd_{Su[m]}}[x] \text{ in } \mathbf{Hd_{Su[m]}}\text{for x:1}\rightarrow\text{hdm} \\ \mathbf{Hd_{Su[m]}}[u] \cap \mathbf{Hd_{Su[m]}}[v] = \emptyset\text{, where u, v:1}\rightarrow\text{hdmj, and u} \neq \text{v} \end{cases}$$

Combining hosts and services ($\mathbf{Hd_{Su[m]}}[x]$,**Su**[m]) where x:1→hdmj results in a list of tuples that are disjoint. This holds for all m:1→suj. At this point, all services and hosts delivering the services form disjoint tuples and may thus be used as a basis for creating an order-independent firewall rule base. $\mathbf{Cs_{Hd_{Su[m]}[x]}}$ is the list of clients that have access to service **Su**[m], defined on host $\mathbf{Hd_{Su[m]}}[x]$.
By using:

- **Su**[m] where m:1→suj, with suj=number of disjoint services offered on the network, for defining **R**.Port

- $\mathbf{Hd}_{\mathbf{Su}[\mathrm{m}]}$[x], x:→hdmj, with hdmj=number of hosts offering **Su**[m], for defining **R**.Destination
- $\mathbf{Cs}_{\mathbf{Hd}_{\mathbf{Su}[\mathrm{m}]}[\mathrm{x}]}$ being the list of clients requiring access to service **Su**[m] on host $\mathbf{Hd}_{\mathbf{Su}[\mathrm{m}]}$[x], of length = cjs, for defining **R**.Source
- "Allow", for **R**.action

Disjoint rules will be created, which are usable for an evolvable firewall rule base.

Example:
Suppose we have two network services: a DB and a WEB service. The DB service is on TCP-1200 and the WEB service is on TCP-8080. We have three hosts - H1,H2, and H3. H1 and H2 offer the DB service. H2 and H3 offer the WEB service. We have five clients, C1 to C5.

Su = [DB, WEB]
Hd = [$\mathbf{Hd}_{\mathbf{Su}[\mathrm{DB}]}$, $\mathbf{Hd}_{\mathbf{Su}[\mathrm{WEB}]}$]
– $\mathbf{Hd}_{\mathbf{Su}[\mathrm{DB}]}$ = [H1, H2]
– $\mathbf{Hd}_{\mathbf{Su}[\mathrm{WEB}]}$ = [H1, H3]
Cs = [$\mathbf{Cs}_{\mathbf{Hd}_{\mathbf{Su}[\mathrm{DB}]}[\mathrm{H1}]}$,$\mathbf{Cs}_{\mathbf{Hd}_{\mathbf{Su}[\mathrm{DB}]}[\mathrm{H2}]}$, $\mathbf{Cs}_{\mathbf{Hd}_{\mathbf{Su}[\mathrm{WEB}]}[\mathrm{H1}]}$, $\mathbf{Cs}_{\mathbf{Hd}_{\mathbf{Su}[\mathrm{DB}]}[\mathrm{H3}]}$]
– $\mathbf{Cs}_{\mathbf{Hd}_{\mathbf{Su}[\mathrm{DB}]}[\mathrm{H1}]}$ = [C1, C2, C3]
– $\mathbf{Cs}_{\mathbf{Hd}_{\mathbf{Su}[\mathrm{DB}]}[\mathrm{H2}]}$ = [C2, C3]
– $\mathbf{Cs}_{\mathbf{Hd}_{\mathbf{Su}[\mathrm{WEB}]}[\mathrm{H1}]}$ = [C1, C2, C3, C4, C5]
– $\mathbf{Cs}_{\mathbf{Hd}_{\mathbf{Su}[\mathrm{DB}]}[\mathrm{H3}]}$ = [C4]

A rule to allow access to service DB on H1 is to be created with the following components:
Su[DB] – a Service that is disjoint with respect to all other existing Services
$\mathbf{Hd}_{\mathbf{Su}[\mathrm{DB}][\mathrm{H1}]}$ = [H1] – a host that is disjoint with respect to all other Hosts offering the same hervice
$\mathbf{Cs}_{\mathbf{Hd}_{\mathbf{Su}[\mathrm{DB}]}[\mathrm{H1}]}$ – a group dedicated to the host/service combination.

4.2 The Green-Field Artifact

Information presented in the previous section needs to be translated into a solution that is usable in a real-world firewall. As discussed in Section 1.1.3, firewalls work with groups. Groups can be used to represent the concepts discussed in the previous sections.

According to requirement R4, it is preferential to implement a white-list approach, containing rules that explicitly allow traffic as well as explicitly deny traffic that does not have a matching rule. In combination with requirement R1 (only disjoint rules), this means that we require a rule base containing only disjoint explicit Allow rules and one default Deny rule. The default Deny rule is not disjoint with the explicit Allow rules and must be assigned to a location where it may not interact with the explicit Allow rules, i.e. at the end of the rule base. We can add new disjoint rules wherever we require, except for after the default Deny rule. The default Deny rule goes first in and last out, and all explicit Allow rules precede the default Deny. The default Deny rule is always evaluated as the final rule.

1. When starting from an empty firewall rule base **F**, the first rule to be added shall be the default Deny rule **F**[1]= $\mathbf{R}_{\text{default\_deny}}$ with

$$\begin{cases} \mathbf{R}_{\text{default\_deny}}.\text{Source = ANY,} \\ \mathbf{R}_{\text{default\_deny}}.\text{Destination=ANY,} \\ \mathbf{R}_{\text{default\_deny}}.\text{Port= ANY,} \\ \mathbf{R}_{\text{default\_deny}}.\text{Action = ``Deny''.} \end{cases}$$

2. For each Service offered on the network, create a group. All Service groups must be entirely disjoint from one another: the intersection between groups must be empty.
   Naming convention to follow:
   - **S**_*service.name*,
   - with *service.name* as the name of the Service.
3. For each Host offering the Service defined in the previous step, a group must be created containing only one item (being the Host offering that specific Service).
   Naming convention to follow:
   - **H**_*host.name*_**S**_*service.name*,
   - with *host.name* as the name of the Host offering the Service
4. For each Host offering the Service from the first step, a Client group must be created. That group will contain all Clients requiring access to the specific Service on the Specific host.
   Naming convention to follow:
   - **C**_**H**_*host.name*_**S**_*service.name*
5. For each (**S**_*service.name*, **H**_*host.name*_**S**_*service.name*) combination, create a rule **R** with:

$$\begin{cases} \mathbf{R}.\text{Source =}\mathbf{C\_H}\_\textit{host.name}\_\mathbf{S}\_\textit{service.name} \\ \mathbf{R}.\text{Destination = }\mathbf{H}\_\textit{host.name}\_\mathbf{S}\_\textit{service.name} \\ \mathbf{R}.\text{Port= }\mathbf{S}\_\textit{service.name} \\ \mathbf{R}.\text{Action = ``Allow''} \end{cases}$$

   Add those rules to the firewall rule base **F** above the default deny rule.
   The default rule $\mathbf{R}_{\text{default}}$ should always be at the end of the rule base.

Example:
Let's revisit the previous example. The different objects the artifact generates are:
S_DB – containing TCP-1200
S_WEB – containing TCP-8080
H_H1_**S**_DB – containing H1
H_H3_**S**_DB – containing H3
H_H1_**S**_WEB – containing H1
H_H2_**S**_WEB – containing H2
C_**H**_H1_**S**_DB – containing C1, C2, C3
C_**H**_H3_**S**_DB – containing C2, C3
C_**H**_H1_**S**_WEB – containing C1, C2, C3, C4, C5

C_**H**_H2_**S**_WEB – containing C1

All rules regarding the DB services are created with Service objects that are disjoint from each other and Destination objects that are disjoint from each other, both in name and content. They will always result in disjoint rules. This is also the case for WEB services.

Through the use of the artifact's design principles, group objects are created that form the building blocks for an evolvable rule base. Each building block addresses one concern.

If each Service of **Su** is comprised of only one Port, then the **Su** will contain the maximum number of elements, being2^{17} elements, resulting in maximum 2^{17} Service groups **S**_*service.name* being created. For each Host, maximum 2^{17} Services can be defined and expressed in **H**_*host.name*_**S**_*service.name* destination groups. The artifact stipulates that one rule per Host and per Service must be created. This reduces the rule base solution space from

$$2.\left(\sum_{a=1}^{cj}\binom{cj}{a}\right)\cdot\left(\sum_{a=1}^{hj}\binom{hj}{a}\right)\cdot\left(\sum_{k=1}^{2^{17}}\binom{2^{17}}{k}\right) \tag{4.1}$$

where cj = $fc(\mathbf{N})$ and hj = $fh(\mathbf{N})$
to:

$$fj = hdj.suj + 1 = hdj.2^{17} + 1 \tag{4.2}$$

with hdj = number of Hosts connected to the network.
hdj = $fh(\mathbf{N})$. The "+1" is the default Deny rule $\mathbf{R}_{\text{default\_deny}}$.

4.3 Green-Field Artifact Demonstration

In this section we will demonstrate the artifact via validation by instantiation. We will apply different generic changes on a generic rule base (add/remove rule) and on the components that make up a rule (add/remove a Service, add/remove a Host, add/remove a Client). We will also show the result of rule aggregation.

4.3.1 Add and Remove a Rule

Creating rules according to the artifact's design principles leads to rules that are disjoint from one another. Disjoint rules can be added and removed from the firewall rule base without introducing CEs.

4.3.2 Adding a New Service to the Network

A new Service is a Service that is currently not defined in **Su**. The new Service results in a new definition of a Service being added to **Su**. The artifact prescribes that a new group **S**_*service.name* must be created for the new Service. The group will contain the ports required for the Service. For each new Host offering the Service, the artifact prescribes to create a new group Destination **H**_*host.name*_**S**_*service.name*, and an associated Source group **C**_**H**_*host.name*_**S**_*service.name*. The Destination groups are populated with only one Host (the Host offering the Service). The Source groups are populated with all Clients requiring access to the Service of one specific Host.

All building blocks to create the disjoint Rules are now available. For each Host offering the new Service, a Rule must be created using the created groups. No CEs are introduced during these operations. Adding new Rules to the rule base does not introduce CEs (see Section 4.3.1).

4.3.3 Adding a New Host Offering Existing Services, to the Network

A new Host is a Host that is currently not defined in **Hd**. The new Host results in a new Host definition being added to **Hd**. The artifact prescribes that a new group **H**_*host.name*_**S**_*service.name* must be created for each Service delivered by the Host and a corresponding Source group **C**_**H**_*host.name*_**S**_*service.name* must be created as well. The Destination groups are populated by their corresponding Hosts. The Source groups are populated with all Clients requiring access to the Service on that Host. All building blocks to create the disjoint Rules are now available. For each Service offered by the new Host, a Rule must be created using the created groups. No CEs are being introduced during these operations. Adding the new Rules to the rule base does not introduce CEs (see Section 4.3.1).

4.3.4 Adding a New Host Offering New Services, to the Network

Sections 4.3.3 and 4.3.2 are in combination what is required to complete this type of change. The artifact prescribes that new Service groups must be created for new Services. An equal number of Destination groups must be created and each be populated by the new Host. The same number of Source groups must be created and populated by the Clients requiring access to one of the new Services on the new host. All building blocks to create the disjoint rules are now available. For each combination (new Host, new Service), a Rule must be created using the created groups. No CEs are being introduced during these operations. Adding the new Rules to the rule base does not introduce CEs (see Section 4.3.1).

4.3.5 Adding a New Client to the Network

Adding a new Client to the network requires neither the creation of new rule building blocks nor the addition of new Rules. The new Client only needs to be added to those Source groups that provide access to the required Services/Hosts combinations. No CEs are being introduced during these operations.

4.3.6 Removing a Service From the Network

Let **sr** be the Service that needs to be removed from the network. The name of the Service is **sr**.name=sremove. The Service is part of **Su**. The group corresponding with **sr** is **S**_sremove. The Hosts offering the Service correspond with the groups **H**_*host.name*_**S**_sremove. The Clients consuming the Service are defined in **C**_**H**_*host.name*_**S**_sremove. All building blocks to identify the Rules that require removal from the rule base are now available. For each Host offering **sr**, the corresponding rule

$$\left\{\begin{array}{l}\mathbf{R}.\text{Source} = \mathbf{C\_H}\_\textit{host.name}\_\mathbf{S}\_\text{sremove}\\ \mathbf{R}.\text{Destination}=\mathbf{H}\_\textit{host.name}\_\mathbf{S}\_\text{sremove}\\ \mathbf{R}.\text{Port}= \mathbf{S}\_\text{sremove}\\ \mathbf{R}.\text{Action} = \text{“Allow”}\end{array}\right.$$

must be removed from the rule base. No CEs are being introduced during these operations. Removing Rules from the rule base does not introduce CEs (see Section 4.3.1). The Service **sr** needs to be removed from **Su** as well as from the corresponding group **S**_remove in the firewall.

4.3.7 Removing a Host From the Network

Let **hr** be the Host that needs to be removed from the network. The name of the Host is **hr**.name=hremove. The Host is part of **Hd**. There will be as many Destination groups for **hr** as there are Services offered by **hr**. They are defined by **H**_hremove_**S**_*service_name*. The same holds for the Source groups, defined by **C**_**H**_hremove_**S**_*service.name*. All building blocks to identify the Rules that require removal from the rule base are available. For each Service offered by **hr**, the corresponding Rule

$$\begin{cases} \mathbf{R}.\text{Source = }\mathbf{C}\_\mathbf{H}\_\text{hremove}\_\mathbf{S}\_\textit{service.name} \\ \mathbf{R}.\text{Destination=}\mathbf{H}\_\text{hremove}\_\mathbf{S}\_\textit{service\_name} \\ \mathbf{R}.\text{Port= }\mathbf{S}\_\textit{service.name} \\ \mathbf{R}.\text{Action = ``Allow''} \end{cases}$$

must be removed from the rule base. No CEs are being introduced during these operations. Removing Rules from the rule base does not introduce CEs (see Section 4.3.1). The Host **hr** needs to be removed from **Hd** as must the corresponding groups **H**_remove_**S**_*service.name* in the firewall.

4.3.8 Removing a Service From a Host

Let **sr** be the Services with **sr**.name=sremove, which need removal from Host **hr** with **hr**.name = hremove. The Service is part of **Su**. The group corresponding with **sr** is **S**_sremove. The Destination group for Service **sr** on Host **hr**, is **H**_hremove_**S**_sremove. The corresponding Source group is **C**_**H**_hremove_**S**_sremove. All building blocks to identify the Rule

$$\begin{cases} \mathbf{R}.\text{Source = }\mathbf{C}\_\mathbf{H}\_\text{hremove}\_\mathbf{S}\_\text{sremove} \\ \mathbf{R}.\text{Destination=}\mathbf{H}\_\text{hremove}\_\mathbf{S}\_\text{sremove} \\ \mathbf{R}.\text{Port= }\mathbf{S}\_\text{sremove} \\ \mathbf{R}.\text{Action = ``Allow''} \end{cases}$$

requiring removal from the rule base are available. No CEs are being introduced during these operations. Removing Rules from the rule base does not introduce CEs (see Section 4.3.1). The Service **sr** does not need to be removed from **Su**, nor does the corresponding group as the Service continues to be offered on other Hosts.

4.3.9 Removing a Client From the Network

Let **cr** be a Client that needs to be removed from the network. The Client is part of **Cs**. Removing a Client from the network does not require removing Rules from the rule base. The Client needs to be removed from the different Source groups that provide the Client access to specific Services on specific Hosts. If the Services and Hosts to which the Client has access are known, then the Source group from which the Client needs to be removed are also known. If the Services and/or Hosts are not known, then an investigation of all the Source groups is required to see if the Client is part of the group or not. If part of the group, the Client needs to be removed. The Client also needs to be removed from **Cs**. Determining whether or not a Client is a member of a Source group can be considered as a CE as all Source groups require inspection.

4.3.10 Summary of the Demonstration

Table 4.1 provides a summary of the green-field artifact demonstration.

Type of Change	Impact of the Change
ADD a Rule	no CE
ADD a new Service	no CE
ADD a new Host with an existing Service	no CE
ADD a new Host with a new Service	no CE
ADD a new Client	no CE
REMOVE a Rule	no CE
REMOVE a Service	no CE
REMOVE a Host	no CE
REMOVE a Service from a Host	no CE
REMOVE a Client	CE at Client level

TABLE 4.1: Summary green-field artifact demonstration

4.4 The Impact of Aggregations

The artifact results in a fine-grained rule base, i.e. one that contains many rules. The urge to aggregate and consolidate rules into fewer and less granular rules will be a natural inclination of firewall administrators since many of them would (wrongfully) deem that the smaller rule base, the less complicated it is. However, any form of aggregation will result in information loss. Specifically because the artifact consciously enforces fine-grained information in the group naming and usages, therefore disjoint rules can be created and the ZT model can be enforced. If, due to aggregations, it can no longer be guaranteed that rules are disjoint, then neither can a CE-free rule base continue to be guaranteed. Aggregation will also lead to violations of the ZT model.
In the following subsections, we provide two examples of the impact of aggregations.

4.4.1 Aggregation at Host Level

All hosts offering the same service are aggregated into one Destination group. Such an aggregation excludes the possibility of specifying that a client needs access to a specific service on a particular Host. A client will have access to the service on all hosts offering the Service, whether desired or not. In such a configuration, ZT can no longer be guaranteed. As long as the Service groups are unique, disjoint rule can still be made and the rule base will remain evolvable. The moment that ZT and non-ZT rules are combined, CEs start creeping into the creation process of new rules.

Example

Assume the following setup; five hosts, H1 to H5, all offer service S1. If we apply the green-field artifact and aggregation at host level, we require the creation of a Destination group containing the five hosts. The name of the Destination group would be **H**_*_**S**_S1. The "*" represent the aggregation of the hosts. The group **C**_**H**_*_**S**_S1 contains all clients requiring access to S1 on all hosts.
Assume now that there is a new client that requires access to S1 but only on H1. We cannot use the existing groups to create the required rule as they give access to all hosts that offer S1. We need to create a new Destination group **H**_H1_**S**_S1 and a new Client group **C**_**H**_H1_**S**_S1. Together with **S**_S1, the required rule can be created.
Assume now that there is an existing client that requires access to S1 on a new host H6 and that H6 cannot be part of existing aggregations. We require groups **H**_H6_**S**_S1 and **C**_**H**_H6_**S**_S1 to be able to create such a rule.
The group **H**_*_**S**_S1 no longer represents its original intent as it no longer contains all hosts offering S1. This eliminates he anthropomorphic relation we want to enforce with the green-field artifact and which is required for an evolvable rule base. It becomes necessary to investigate the content of all aggregation Destination groups to understand their intent. This opens the door for the introduction of CEs, as adding clients and hosts to the network can no longer be expanded as foreseen in the green-field artifact, but requires analysis of the content of the Destination group(s). The green-field artifact anticipated all possible changes in a rule base. The introduction of aggregation in combination with more fine-grained filtering no longer makes it possible to anticipate all changes, as the changes can no longer be generalized. The combination of aggregation and ZT makes changes content- and context-specific, resulting in the introduction of CEs at run-time.

4.4.2 Aggregation at Service Level

All services offered on a host are aggregated into one Service group. The aggregation method excludes specifying that a client needs access to some of the services on the host. A client will have access to all services defined on the host, whether desired or not. In such a configuration, ZT can no longer be guaranteed. As long as the Destination groups are disjoint, disjoint Rules can still be created. The moment that ZT and non-ZT rules are combined, a non-disjoint Rule will emerge. The rule base can no longer be guaranteed CE-free.

Example
Assume the following setup; there are five services, S1 to S5, that are delivered by three hosts, H1 to H3. The services are aggregated in a service group **S**_S*, where "*" represents the aggregation of services. The Destination groups are **H**_H1_**S**_S*, **H**_H2_**S**_S* and **H**_H3_**S**_S* and the Source groups are **C**_**H**_H1_**S**_S*, **C**_**H**_H2_**S**_S* and **C**_**H**_H3_**S**_S*. As there is disjointness at the destination, a disjoint rule can be made.
Assume now that a client is only allowed access to S1 on H1. We need to create a new Service group **S**_S1 and Source group **C**_**H**_H1_**S**_S1. This Service group is no longer disjoint with **S**_S*.
Assume, for example, that a new service S6 is introduced at H3. We need to create a new Service Group **S**_6 and Source group **C**_**H**_H3_**S**_S6. The aggregated Service group **S**_S* no longer reflects its original intent as it does not contain S6. If it were it to contain S6, the filtering intent could not be correctly implemented.

We run into similar problems as those discussed previously for host level aggregations. Either we introduce non-disjoint groups, allowing the creation of non-disjoint rules, or the aggregations no longer represent their intent, breaking the anthropomorphic relation and potentially introducing CEs when rules are being created.

Chapter 5

Brown-Field Artifact Creation and Demonstration

In this chapter, we design an artifact — an algorithm this time — that will convert an existing rule base into an evolvable rule base. The algorithm is based on the Iterated Local Search (ILS) meta-heuristic, which is a classic way to solve an optimization problem. The optimization we are addressing is: maximize evolvability while minimizing the number of rules. Section 5.1 discusses the design of the algorithm, while Section 5.2 demonstrates the algorithm with firewall exports from Engie. We conclude with our findings in Section 5.3. The findings of this chapter have been published in [46] and [47].

5.1 Brown-Field Artifact Design

In this section we will discuss the different components that comprise the algorithm. We begin by rationalizing the choice for Iterated Local Search (ILS) as meta-heuristic [48] [49] [50]. We will discuss the nature of the initial solution, the set of feasible solutions, and the objective function associated with a solution. We continue by defining the move type, move strategy, perturbation and stop condition of the Iterated Local Search. The final part of this section begins with a discussion of the solution encoding and special operations performed in the algorithm, and concludes with the presentation of the algorithm.

5.1.1 Meta-Heuristic Selection

The objective is to disentangle/reshuffle the service definitions into a set of new service definitions that are disjoint but maximally large. The simplest solution is to create one service definition per port. However, some ports belong together to deliver a service. This filtering logic is embedded in the rule base and service definitions. It must be preserved.

Service definitions containing ports that appear in multiple service definitions must be split into non-overlapping service definitions. The result should be that the degree of overlap (or disjointness) of all service definitions decreases as more service definitions are split. Let us say that a user measures the degree of disjointness of the entirety of the service definitions (pre-change and post-change) and then observes a post-change improvement in the degree of disjointness. It would be correct to conclude that the change represents an improvement to the previous version.

A Local Search (LS) heuristic is a suitable method for organizing such gradual improvement processes. To avoid getting stuck in a local optimum (see further), the

Local Search will be upgraded to an Iterated Local Search. The Iteration component should result in avoiding becoming stuck in a local optimum where we can no longer perform splits and improve the disjointness. The Iteration component should perform a special kind of split called a "perturbation" that will allow the continuation of the search for improvement.

5.1.2 Initial Solution and Neighborhood

The initial solution is the rule base containing all of the service definitions. It is the rule base with all the service definitions. The set of all service definitions is our neighborhood. We will have to pick a service definition, confirm whether or not it is disjoint and, if not, split it and see how this affects the solution - that is whether or not disjointness has improved. The solution space (SP) for the service definitions consists of all possible combinations of ports. If the number of distinct ports in the service groups equals **P**, then the **SP** is:

$$\mathbf{SP} = \sum_{k=1}^{\mathbf{P}} \binom{\mathbf{P}}{k} \tag{5.1}$$

P can be max 2^{17}. We are looking to find a new solution that is part of the solution space, in which all service definitions are disjoint yet grouped within groups of maximum size.

5.1.3 Objective Function

To know whether or not the splitting of a service definition results in improving the solution, we need a mechanism to express the degree of disjointness of a service definition and of the total rule base.

Let $\boldsymbol{p}$ represent a service port.

Let **S** be a set of ports, representing a service definition.
$\mathbf{S} = \{ \boldsymbol{p}_1 ... \boldsymbol{p}_{nS} \}$
where$|\ \mathbf{S}\ | = nS$ = number of ports in the service definition.

Let σ be the set of all service definitions $\mathbf{S}_\mathrm{i}$ used in the firewall rule base.
$\sigma = \{\mathbf{S}_1 ... \mathbf{S}_{\mathrm{n}\sigma}\}$
where$|\ \sigma\ | = n\sigma$= number of service definitions

Let $\mathbf{PF}(\boldsymbol{p}_x)_\sigma$ be the port frequency of port $\boldsymbol{p}_x$ in σ, as the number of times $\boldsymbol{p}_x$ is used in services of σ.

$$\mathbf{PF}(\boldsymbol{p}_x)\sigma = \sum_{i=1}^{n\sigma} |\ \mathbf{S}_\mathrm{i} \cap \boldsymbol{p}_x\ | \tag{5.2}$$

We define the Disjointness Index $\mathbf{DI(S_\mathrm{x})}_\sigma$, of a service definition $\mathbf{S}_\mathrm{x}$, in σ as the sum of the port frequencies $\mathbf{PF}(\boldsymbol{p}_x)_\sigma$ of the ports $\boldsymbol{px}$ of $\mathbf{S}_\mathrm{x}$, divided by the number of ports in $\mathbf{S}_\mathrm{x}$.

$$\mathbf{DI}(S_\mathrm{x})\sigma = \frac{\sum_{i=1}^{nx} PF(p_x)_\sigma}{nx} \tag{5.3}$$

where $nx = |\ S_\mathrm{x}\ |$= number of ports in $\mathbf{S_x}$.

A disjoint service is a service whereby each port **p** appears in only one service definition. The **DI** of a disjoint service will have a value of **1** and a value greater then 1 if the service is not disjoint.

We define the Objective Function $\mathbf{OF}_\sigma$, in σ, as the sum of all $\mathbf{DI(S_x)}_\sigma$ and of all service definitions in σ.

$$\mathbf{OF}\sigma = \sum_{i=1}^{n\sigma} \mathbf{DI}(\mathbf{S}_i)\sigma \tag{5.4}$$

with $n\sigma$ the number of service definitions in the solution.

We define an Optimal Solution as a solution where $\mathbf{OF}\sigma$ equals the number of service definitions, as this means that all **DI** of all service definitions are equal to 1.

$$\mathbf{OF}\sigma = \mid \sigma \mid \tag{5.5}$$

An Optimal Solution is not necessarily a Global Optimum as making service definitions of one port would also yield an objective function value that is equal to the total number of service definitions.

5.1.4 Feasible Solutions

Whatever kind of splits we will be performing, the original filtering logic of the rule base must be maintained. When a service is split, all rules that contain this service must be modified. The original service must be replaced by the result of the split. As we want a rule to contain only one service definition, it may be required to split the rules containing the split result.
Example: R1 contains service Sx. Sx is split into Sx1 and Sx2. To reflect this, we replace Sx with Sx1 and Sx2 in rule R1.However, a rule must only contain one service. R1 needs to be split into R1.1 and R1.2, where R1.1 is a copy of R1 but with Sx being replaced by Sx1, and R1.2 is a copy of R1 but with Sx being replaced by Sx2. Both rules are put in consecutive locations in the rule base.

5.1.5 Move Type

Before we decide on the move type, we must first investigate the impact that splitting of service definitions has on the objective function. Based on this analysis, a selection of type of split (move type) will be made.

The Impact of Splitting Service Definitions on the OF

A service definition can:

- be a subset of existing service definitions.
- be the superset of existing service definitions.
- be partially overlapped with other service definitions.
- be a combination of the above.

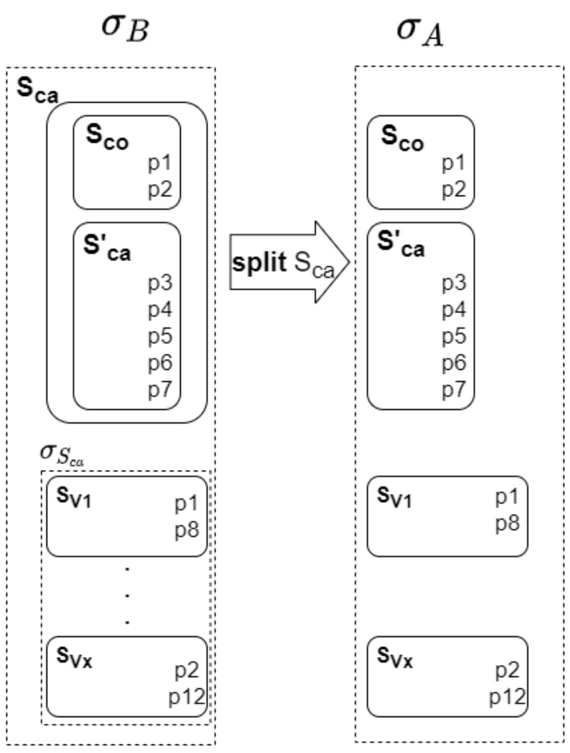


FIGURE 5.1: Split example

Let $\mathbf{S}_{\text{ca}}$ be the candidate service we will split.

$$\begin{cases} \mathbf{S}_{\text{ca}} = \{\boldsymbol{p}_{1} ... \boldsymbol{p}_{nca}\} \\ nca = \mid \mathbf{S}_{\text{ca}} \mid = \text{number of ports in the } \mathbf{S}_{\text{ca}} \end{cases}$$

Let $\mathbf{S}_{\text{co}}$ be an arbitrary set of ports that are part of $\mathbf{S}_{\text{ca}}$, making up the new service $\mathbf{S}_{\text{co}}$, that is to be extracted from $\mathbf{S}_{\text{ca}}$.

$$\begin{cases} \mathbf{S}_{\text{co}} = \{\boldsymbol{p}_{j} ... \boldsymbol{p}_{j+nco}\} \\ nco = \mid \mathbf{S}_{\text{co}} \mid = \text{number of ports in the } \mathbf{S}_{\text{co}}. \\ \mathbf{S}_{\text{ca}} \cap \mathbf{S}_{\text{co}} = \{\boldsymbol{p}_{j} ... \boldsymbol{p}_{j+nco}\} \\ |\mathbf{S}_{\text{ca}} \cap \mathbf{S}_{\text{co}}| = nco \end{cases}$$

Let $\mathbf{S'}_{\text{ca}}$ be the new service comprised of ports that are part of $\mathbf{S}_{\text{ca}}$ but not of $\mathbf{S}_{\text{co}}$. $\mathbf{S'}_{\text{ca}}$ is what is left of $\mathbf{S}_{\text{ca}}$, after splitting-up or carving-out $\mathbf{S}_{\text{co}}$

$$\begin{cases} \mathbf{S'}_{\text{ca}} = \mathbf{S}_{\text{ca}} \setminus \mathbf{S}_{\text{co}} = \{\boldsymbol{p}_{1} ... \boldsymbol{p}_{j-1}, \boldsymbol{p}_{j+nco+1}, ... \boldsymbol{p}_{nca}\} \\ \mid \mathbf{S'}_{\text{ca}} \mid = nca - nco \end{cases}$$

Let $\sigma_{\mathbf{S}_{ca}}$ be the set of services that contains ports that are also part of service $\mathbf{S}_{\text{ca}}$.

$$\begin{cases} \sigma_{\mathbf{S}_{ca}} = \{\mathbf{S}_{\text{V1}} ... \mathbf{S}_{\text{Vn}}\} \\ \forall \mathbf{S}_{\text{Vx}} \text{ x=1} \rightarrow \text{n} \mid \\ * \ |\mathbf{S}_{\text{ca}} \cap \mathbf{S}_{\text{Vx}}| \neq \emptyset \\ * \mid \mathbf{S}_{\text{Vx}} \mid = Vnx \\ * \ |\mathbf{S}_{\text{ca}} \cap \mathbf{S}_{\text{Vx}} \mid = \text{q}_{\text{x}} = \text{the amount of port overlap between } \mathbf{S}_{\text{ca}} \text{ and } \mathbf{S}_{\text{Vx}} \end{cases}$$

See Figure 5.1 for a visual representation of these definitions.

When the split or carve-out of **S**co from **S**ca is performed, the port frequencies, the **DI** and the **OF** change, depending on the effect of the split. We shall now investigate under which conditions the split will improve the objective function.

Let σ_B be the set of services before the split and σ_A be the set of services after the split. We want to know which conditions will improve the Objective Function, or

$$\begin{cases} \Delta\mathbf{OF} = \mathbf{OF}_{\sigma_B} - \mathbf{OF}_{\sigma_A} > 0 \\ \Delta\mathbf{OF} > 0 \text{ means } \mathbf{OF} \text{ improved (=lowered).} \\ \Delta\mathbf{OF} < 0 \text{ means } \mathbf{OF} \text{ deteriorated (=increased).} \end{cases}$$

$\mathbf{S}_{\text{co}}$ is a random subgroup of $\mathbf{S}_{\text{ca}}$, meaning not necessarily part of σ_B. Subsequent to a split $\mathbf{S}_{\text{ca}}$ becomes $\mathbf{S'}_{\text{ca}}$. Both $\mathbf{S'}_{\text{ca}}$ and $\mathbf{S}_{\text{co}}$ are part of σ_A.

There are three possible cases:

- $\mathbf{S'}_{\text{ca}}$ and $\mathbf{S}_{\text{co}}$ also exist in σ_B. The split results in two existing services. We merge them into the existing services — the split results in a reduction of the total number of services with 1.
  $| \sigma_A | - | \sigma_B | = -1$
- $\mathbf{S'}_{\text{ca}}$ or $\mathbf{S}_{\text{co}}$ exists in σ_B. The split results in a new service and an existing service. The existing service merges and the split results in an equal number of services.
  $| \sigma_A | - | \sigma_B | = 0$
- $\mathbf{S'}_{\text{ca}}$ and $\mathbf{S}_{\text{co}}$ do not exist in σ_B. The split results in two new services and the split results in an increase of the total number of services with 1.
  $| \sigma_A | - | \sigma_B | = +1$

We shall now investigate what kind of change in Objective Function value we can expect, based on the three following cases.

Case 1: $| \sigma_B | - | \sigma_A |$ **= -1**
$\mathbf{S'}_{\text{ca}}$ and $\mathbf{S}_{\text{co}}$ are elements of σ_A and σ_B.
$\mathbf{S}_{\text{ca}}$ only exists in σ_B.
$\mathbf{S}_{\text{ca}} = \{p_1 ... p_{nca}\}$
As $\mathbf{S}_{\text{ca}}$ is not part of σ_A, the port frequencies of all ports of S_{ca} decreased by 1 in σ_A.

$$\forall \mathbf{p} \in \mathbf{S}_{ca} \,|\, (\mathbf{PF}(\mathbf{p})_{\sigma_A} = \mathbf{PF}(\mathbf{p})_{\sigma_B} - 1 \tag{5.6}$$

See Figure 5.2 for a graphical representation.

As the port frequencies of all ports that are part of $\mathbf{S}_{\text{ca}}$ decrease, the **DI** of all groups that contain one or more port of $\mathbf{S}_{\text{ca}}$ are also impacted. These are all $\mathbf{S}_{\text{Vi}}$ service groups.

When calculating ΔOF, only the impacted service groups must be taken into account.

$\Delta\mathbf{OF} = \mathbf{OF}_{\sigma_B} - \mathbf{OF}_{\sigma_A} = [\,\mathbf{DI}(\mathbf{S}_{ca})_{\sigma_B} + \mathbf{DI}(\mathbf{S'}_{ca})_{\sigma_B} + \mathbf{DI}(\mathbf{S}_{co})_{\sigma_B} + \sum_{i=1}^{n} \mathbf{DI}(\mathbf{S}_{Vi})_{\sigma_B}] - [\,\mathbf{DI}(\mathbf{S'}_{ca})_{\sigma_A} + \mathbf{DI}(\mathbf{S}_{co})_{\sigma_A} + \sum_{i=1}^{n} \mathbf{DI}(\mathbf{S}_{Vi})_{\sigma_A}]$

$\Delta\mathbf{OF} = \mathbf{DI}(\mathbf{S}_{ca})_{\sigma_B} + [\,\mathbf{DI}(\mathbf{S'}_{ca})_{\sigma_B} - \mathbf{DI}(\mathbf{S'}_{ca})_{\sigma_A}] + [\,\mathbf{DI}(\mathbf{S}_{co})_{\sigma_B} - \mathbf{DI}(\mathbf{S}_{co})_{\sigma_A}] + [\,\sum_{i=1}^{n} \mathbf{DI}(\mathbf{S}_{Vi})_{\sigma_B} -$

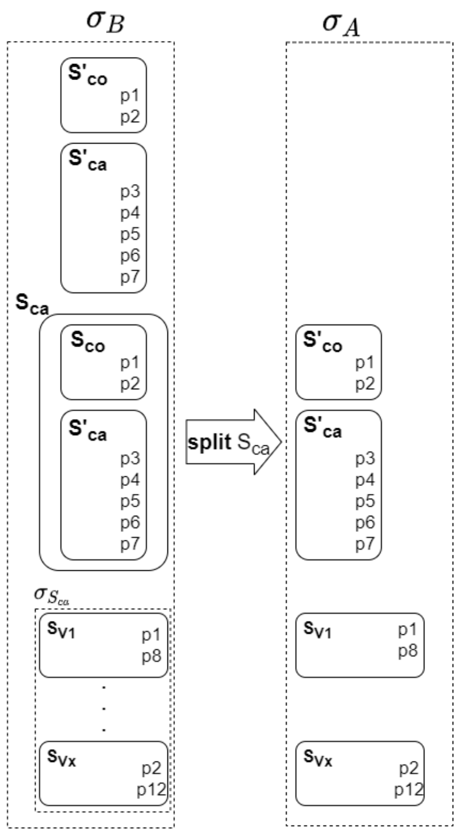


FIGURE 5.2: Split case 1

$\sum_{i=1}^{n} \mathbf{DI}(\mathbf{S}_{Vi})_{\sigma_A}]$

Taking (5.3) and (5.6) into account:

(a) $\mathbf{DI}(\mathbf{S'}_{ca})_{\sigma_A} = \frac{\sum_{i=1}^{nca-nco} \mathbf{PF}(\mathbf{p}_i)_{\sigma_A}}{nca-nco} = \frac{\sum_{i=1}^{nca-nco} \mathbf{PF}(\mathbf{p}_i)_{\sigma_B} - (nca-nco)}{nca-nco} = \mathbf{DI}(\mathbf{S'}_{ca})_{\sigma_B} - 1$

$\implies \mathbf{DI}(\mathbf{S'}_{ca})_{\sigma_B} - \mathbf{DI}(\mathbf{S'}_{ca})_{\sigma_A} = 1$

(b) $\mathbf{DI}(\mathbf{S}_{co})_{\sigma_A} = \frac{\sum_{i=1}^{nco} \mathbf{PF}(\mathbf{p}_i)_{\sigma_A}}{nco} = \frac{\sum_{i=1}^{nco} \mathbf{PF}(\mathbf{p}_i)_{\sigma_B} - (nco)}{nco} = \mathbf{DI}(\mathbf{S}_{co})_{\sigma_B} - 1$

$\implies \mathbf{DI}(\mathbf{S}_{co})_{\sigma_B} - \mathbf{DI}(\mathbf{S}_{co})_{\sigma_A} = 1$

(c) $\sum_{i=1}^{n} \mathbf{DI}(\mathbf{S}_{Vi})_{\sigma_B} - \sum_{i=1}^{n} \mathbf{DI}(\mathbf{S}_{Vi})_{\sigma_A} = \sum_{i=1}^{n} \frac{\sum_{j=1}^{nvi} \mathbf{PF}(\mathbf{p}_j)_{\sigma_B}}{nvi} - \sum_{i=1}^{n} \frac{\sum_{j=1}^{nvi} \mathbf{PF}(\mathbf{p}_j)_{\sigma_A}}{nvi}$

$= \sum_{i=1}^{n} \frac{\sum_{j=1}^{nvi} \mathbf{PF}(\mathbf{p}_j)_{\sigma_B}}{nvi} - \sum_{i=1}^{n} \frac{\sum_{j=1}^{nvi} \mathbf{PF}(\mathbf{p}_j)_{\sigma_B} - qvi}{nvi} = \sum_{i=1}^{n} \frac{qvi}{nvi}$

Putting (a), (b) and (c) into $\Delta\mathbf{OF}$ gives:

$\Delta\mathbf{OF} = \mathbf{DI}(\mathbf{S}_{ca})_{\sigma_B} + 2 + \sum_{i=1}^{n} \frac{qvi}{nvi}$

Conclusion: If | σ_B | - | σ_A | = -1, then ΔOF is always > 0 (all terms are positive), and the Objective Function always improves.

Case 2:* | σ_B | - | σ_A | *= 0

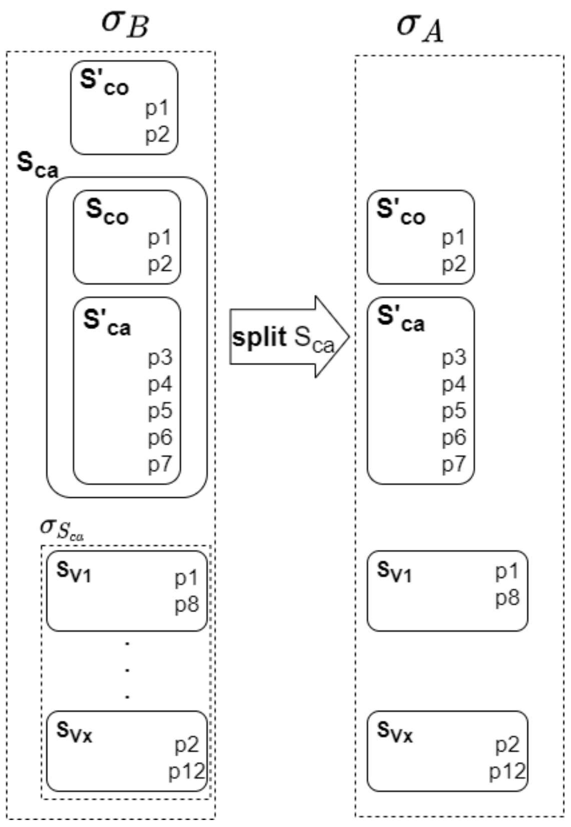


FIGURE 5.3: Split case 2

$\mathbf{S'}_{\text{ca}}$ or $\mathbf{S}_{\text{co}}$ are part of σ_A or σ_B (exclusive OR).
Assume $\mathbf{S}_{\text{co}}$ already exists in σ_B (carve-out of an existing group)
$\mathbf{S}_{ca}$ does not exist in σ_A, but $\mathbf{S'}_{\text{ca}}$ does exist in σ_A. The port frequencies of all ports of $\mathbf{S'}_{\text{ca}}$ do not change.

$$\forall \mathbf{p} \in \mathbf{S}_{ca} \setminus \mathbf{S}_{co} \mid \mathbf{PF}(\mathbf{p})_{\sigma_A} = \mathbf{PF}(\mathbf{p})_{\sigma_B} \tag{5.7}$$

$\mathbf{S}_{co}$ already exists in σ_B. The group cancels out in σ_B and the port frequencies of all ports in $\mathbf{S}_{co}$ decrease.

$$\forall \mathbf{p} \in \mathbf{S}_{co} \mid \mathbf{PF}(\mathbf{p})_{\sigma_A} = \mathbf{PF}(\mathbf{p})_{\sigma_B} - 1 \tag{5.8}$$

See Figure 5.3 for a graphical representation.

As the port frequencies decrease, the **DI** of all groups that contain one or more port of $\mathbf{S}_{\text{co}}$ are also impacted. These are all $\mathbf{S}_{\text{Vi}}$ service groups.

When calculating Δ**OF**, only impacted service groups must be taken into account.

$\Delta\mathbf{OF} = [\,\mathbf{DI}(\mathbf{S}_{ca})_{\sigma_B} + \mathbf{DI}(\mathbf{S}_{co})_{\sigma_B} + \sum_{i=1}^{n} \mathbf{DI}(\mathbf{S}_{Vi})_{\sigma_B}] - [\,\mathbf{DI}(\mathbf{S}_{ca})_{\sigma_A} + \mathbf{DI}(\mathbf{S}_{co})_{\sigma_A} + \sum_{i=1}^{n} \mathbf{DI}(\mathbf{S}_{Vi})_{\sigma_A}]$

$\Delta\mathbf{OF} = [\,\mathbf{DI}(\mathbf{S}_{ca})_{\sigma_B} - \mathbf{DI}(\mathbf{S'}_{ca})_{\sigma_A}] + [\,\mathbf{DI}(\mathbf{S}_{co})_{\sigma_B} - \mathbf{DI}(\mathbf{S}_{co})_{\sigma_A}] + [\,\sum_{i=1}^{n} \mathbf{DI}(\mathbf{S}_{Vi})_{\sigma_B} - \sum_{i=1}^{n} \mathbf{DI}(\mathbf{S}_{Vi})_{\sigma_A}]$

Taking (5.3) and (5,8) into account and using the same type of calculations as in Case 1:

(d) $\mathbf{DI}(\mathbf{S}_{ca})_{\sigma_B} - \mathbf{DI}(\mathbf{S'}_{ca})_{\sigma_A} = \mathbf{DI}(\mathbf{S}_{ca})_{\sigma_B} - \mathbf{DI}(\mathbf{S'}_{ca})_{\sigma_B}$

(e) $\mathbf{DI}(\mathbf{S}_{co})_{\sigma_B} - \mathbf{DI}(\mathbf{S}_{co})_{\sigma_A} = 1$

(f) $\mathbf{DI}(\mathbf{S'}_{ca})_{\sigma_A} + \mathbf{DI}(\mathbf{S}_{co})_{\sigma_A} + \sum_{i=1}^{n} \mathbf{DI}(\mathbf{S}_{Vi})_{\sigma_A} = \sum_{i=1}^{n} \frac{qvi}{nvi}$ (see case 1)

Putting (d), (e) and (f) into ΔOF

$\Delta\mathbf{OF} = \mathbf{DI}(\mathbf{S}_{ca})_{\sigma_B} - \mathbf{DI}(\mathbf{S'}_{ca})_{\sigma_B} + 1 + \sum_{i=1}^{n} \frac{qvi}{nvi} > 0$

The same result is obtained when the assumption is made that $\mathbf{S'}_{\mathrm{ca}}$ already exits in σ_B

Conclusion: If** | σ_B | **-** | σ_A | **= 0, then** Δ**OF (due to** $-DI(S'_{ca})_{\sigma_B}$**) can be < 0, and the Objective Function can thus deteriorate.

Case 3:** | σ_B | **-** | σ_A | **= 1

$\mathbf{S}_{\mathrm{ca}}$ splits into 2 mutually-exclusive new services ($\mathbf{S'}_{\mathrm{ca}}$ and $\mathbf{S}_{\mathrm{co}}$). Neither $\mathbf{S'}_{\mathrm{ca}}$ nor $\mathbf{S}_{\mathrm{co}}$ are part of σ_B.
$\mathbf{S'}_{\mathrm{ca}}$ and $\mathbf{S}_{\mathrm{co}}$ are both part of σ_A.
The port frequencies **PF** of any **p** do not change. No other services are impacted.

$$\forall \mathbf{p} \in \mathbf{S}_{ca} \mid \mathbf{PF}(\mathbf{p})_{\sigma_A} = \mathbf{PF}(\mathbf{p})_{\sigma_B} \tag{5.9}$$

The only factors playing a role in the calculation of Δ**OF** are $\mathbf{DI}(\mathbf{S}_{ca})_{\sigma_B}, \mathbf{DI}(\mathbf{S'}_{ca})_{\sigma_A}$, and $\mathbf{DI}(\mathbf{S}_{co})_{\sigma_A}$

Δ**OF**= $\mathbf{DI}(\mathbf{S}_{ca})_{\sigma_B} - \mathbf{DI}(\mathbf{S'}_{ca})_{\sigma_A} - \mathbf{DI}(\mathbf{S}_{co})_{\sigma_A}$

Δ**OF**= $\mathbf{DI}(\mathbf{S}_{ca})_{\sigma_B} - \mathbf{DI}(\mathbf{S'}_{ca})_{\sigma_B} - \mathbf{DI}(\mathbf{S}_{co})_{\sigma_B}$

Conclusion: If** | σ_B | **-** | σ_A | **= 1, then** Δ**OF can be < 0 (due to** $-DI(S'_{ca})_{\sigma_B} - DI(S_{co})_{\sigma_B}$**), and the Objective Function can thus deteriorate).

See Figure 5.1 for a visual representation of this case.

Only case 1, | σ_B | - | σ_A | = -1, provides full certainty of how **OF** will evolve. To gain more certainly, we shall also investigate the relationship between the **DI** of service $\mathbf{S}_{\mathrm{ca}}$ and the **DI**s of sub services $\mathbf{S'}_{\mathrm{ca}}$ and $\mathbf{S}_{\mathrm{co}}$.

Relationship between DIs

(1) $\mathbf{DI}(\mathbf{S}_{ca})_\sigma = \frac{\sum_{i=1}^{nca} \mathbf{PF}(\mathbf{p}i)_\sigma}{nca} = \frac{\sum_{i=1}^{1} \mathbf{PF}(\mathbf{p}i)_\sigma + \sum_{i=j+1}^{j+nco} \mathbf{PF}(\mathbf{p}i)_\sigma + \sum_{i=j+nco+1}^{nca} \mathbf{PF}(\mathbf{p}i)_\sigma}{nca}$

$nca.\mathbf{DI}(\mathbf{S}_{ca})_\sigma - \sum_{i=j+1}^{j+nco} \mathbf{PF}(\mathbf{p}i)_\sigma = \sum_{i=1}^{j} \mathbf{PF}(\mathbf{p}i)_\sigma + \sum_{i=j+nco+1}^{nca} \mathbf{PF}(\mathbf{p}i)_\sigma$

(2) $\mathbf{DI}(\mathbf{S'}_{ca})_\sigma = \frac{\sum_{i=1}^{nca} \mathbf{PF}(\mathbf{p}i)_\sigma + \sum_{i=j+nco+1}^{nca} \mathbf{PF}(\mathbf{p}i)_\sigma}{nca-nco}$

(3) $nco.\mathbf{DI}(\mathbf{S}_{co})_\sigma = \sum_{i=j+1}^{j+nco} \mathbf{PF}(\mathbf{p}i)_\sigma$

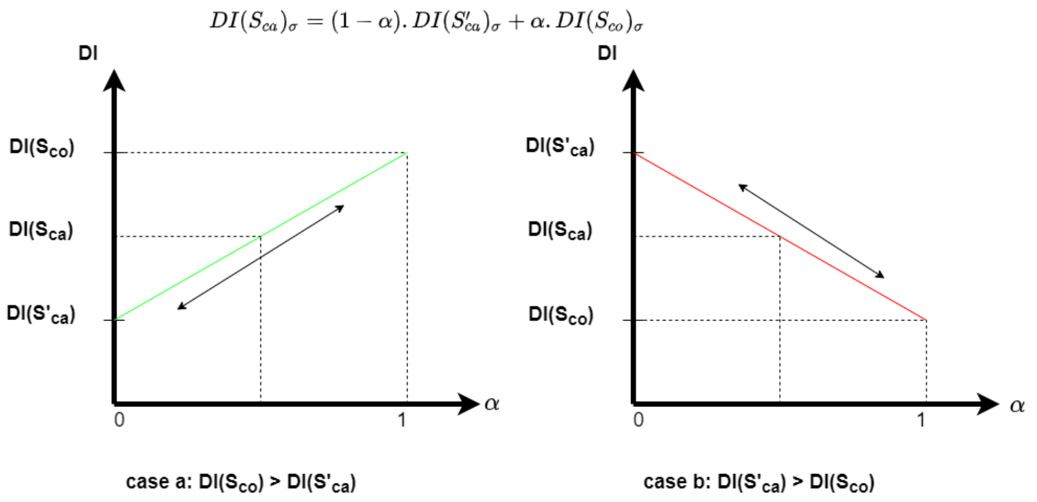


FIGURE 5.4: Linear interpolation

Putting (1), (2) and (3) together gives:

$\mathbf{DI}(\mathbf{S'}_{ca})_\sigma = \frac{nca.\mathbf{DI}(\mathbf{S}_{ca}) - \sum_{i=j+1}^{j+nco} \mathbf{PF}(\mathbf{p}i)_\sigma}{nca-nco}$

$\mathbf{DI}(\mathbf{S'}_{ca})_\sigma = \frac{nca}{nca-nco}.\mathbf{DI}(\mathbf{S}_{ca})_\sigma - \frac{nco}{nca-nco}.\mathbf{DI}(\mathbf{S}_{co})_\sigma$

$\frac{nca}{nca-nco}.\mathbf{DI}(\mathbf{S}_{ca})_\sigma = \mathbf{DI}(\mathbf{S'}_{ca})_\sigma + \frac{nco}{nca-nco}.\mathbf{DI}(\mathbf{S}_{co})_\sigma$

$\mathbf{DI}(\mathbf{S}_{ca})_\sigma = \frac{nca-nco}{nca}.\mathbf{DI}(\mathbf{S'}_{ca})_\sigma + \frac{nco}{nca}.\mathbf{DI}(\mathbf{S}_{co})_\sigma$

Let $\alpha = \frac{nco}{nca}$ be the split-factor, where $0 \leq \alpha \leq 1$

Then

$$\mathbf{DI}(\mathbf{S}_{ca})_\sigma = (1-\alpha).\mathbf{DI}(\mathbf{S'}_{ca})_\sigma + \alpha.\mathbf{DI}(\mathbf{S}_{co})_\sigma \tag{5.10}$$

This formula expresses $\mathbf{DI}(\mathbf{S}_{ca})_\sigma$ as the linear interpolation between $\mathbf{DI}(\mathbf{S'}_{ca})_\sigma$ and $\mathbf{DI}(\mathbf{S}_{co})_\sigma$, with α as the interpolation factor. See Figure 5.4 for a visualization of this linear interpolation function.

Two cases are possible:

- Case a: $\mathbf{DI}(\mathbf{S'}_{ca})_\sigma > \mathbf{DI}(\mathbf{S}_{co})_\sigma$
- Case b: $\mathbf{DI}(\mathbf{S}_{co})_\sigma > \mathbf{DI}(\mathbf{S}_{ca})_\sigma$

Based on the relationship between DIs, we may conclude that:

If | σ_B | - | σ_A | = 1
then Δ**OF**= $\mathbf{DI}(\mathbf{S}_{ca})_{\sigma_B} - \mathbf{DI}(\mathbf{S'}_{ca})_{\sigma_B} - \mathbf{DI}(\mathbf{S}_{co})_{\sigma_B}$ < 0
as according to (5.10) either $\mathbf{DI}(\mathbf{S'}_{ca})_{\sigma_B}$ or $\mathbf{DI}(\mathbf{S}_{co})_{\sigma_B}$ is $>$ $\mathbf{DI}(\mathbf{S}_{ca})_{\sigma_B}$.
The Objective Function thus deteriorates when | σ_B | - | σ_A | = 1.

If | σ_B | - | σ_A | = 0
then $\Delta\mathbf{OF} = \mathbf{DI}(\mathbf{S}_{ca})_{\sigma_B} - \mathbf{DI}(\mathbf{S'}_{ca})_{\sigma_B} + 1 + \sum_{i=1}^{n} \frac{qvi}{nvi}$ can be < 0
if $\Delta\mathbf{OF} < 0$ then $\mathbf{DI}(\mathbf{S}_{ca})_{\sigma_B} < \mathbf{DI}(\mathbf{S'}_{ca})_{\sigma_B}$ must be < 0

Taking (5.10) into account, we can rewrite $\Delta\mathbf{OF}$ as:

$(1-\alpha).\mathbf{DI}(\mathbf{S'}_{ca}) + \alpha.\mathbf{DI}(\mathbf{S}_{co}) - \mathbf{DI}(\mathbf{S'}_{ca}) + 1 + \sum_{i=1}^{n} \frac{qvi}{nvi}$

$-\alpha.(\mathbf{DI}(\mathbf{S'}_{ca}) - \mathbf{DI}(\mathbf{S}_{co}) + 1 + \sum_{i=1}^{n} \frac{qvi}{nvi} > 0$

$\implies \Delta\mathbf{OF} > 0 \text{ if } \alpha < \frac{1+\sum_{i=1}^{n} \frac{qvi}{nvi}}{\mathbf{DI}(\mathbf{S'}_{ca}) - \mathbf{DI}(\mathbf{S}_{co})}$

Thus, a smaller α gives a higher probability of an **OF** improvement.

Split Selection

From the preceding, we conclude that carving-out subgroups has a high likelihood to result in an improvement of the Objective Function. We will even go a step further and define our move type as the carving-out of all subgroups of a service definition. We call our split operator the full-carve-out move. Example: A service definition **S**={1,2,3,4,5,6,7}. There also exists service definitions $\mathbf{S_1}$={1,2} and $\mathbf{S_2}$={5,7}. Carving out $\mathbf{S_1}$ and $\mathbf{S_2}$ from **S** gives, $\mathbf{S_1}$= {1,2}, $\mathbf{S_2}$={5,7} and **S'**={3,4,6}

This move type, however, is unable to handle partial overlapping service definitions. It is expected, therefore, that when all carve-outs are done, there will be a number of overlaps remaining that require a different type of operation.

5.1.6 Move Strategy

All services with a **DI** greater than one are candidates for splitting. It seems logical to begin by splitting the service with the largest **DI**. If that service cannot be split (no subgroups), then the second-largest **DI** is taken, etc. If a group can be split, the impact of the split is calculated. When **OF** improves (=descends), the move is accepted and executed. If not, the next service in the sorted service **DI** list is chosen. The move-strategy is a variant of the First Improvement strategy of the ILS metaheuristic; a variant as we first order the service **DI** list and take the top element from the list.

5.1.7 Perturbation

The carve-out of subgroups cannot remove all forms for non-disjointness. Correlated (partially overlapping) service definitions cannot be split this way. This creates a requirement for a new split operator when no additional carve-outs are possible. The operator will determine if a service definition overlaps with another service definition. If it does, the intersection is split-off. By splitting off this intersection, a new services definition will be created. Splitting off an intersection will result in $\mid \sigma_B \mid$ - $\mid \sigma_A \mid$ = 1. In the previous section we observed that the Objective Function will deteriorate. This is a transitory situation, due to the fact that the newly-formed service definitions may be subgroups of the existing service definitions. We consciously allow the **OF** temporary deterioration so that a better optimum may be found in the next Local Search iteration. We consider this kind of split as the perturbation.

5.1.8 Stop Conditions

Once all possible carve-outs and perturbations are complete, then there are no more inclusively matching and correlated rules. All port frequencies are equal to one, all service group **DI**'s are equal to one, and **OF** will equal the number of service definitions. The solution cannot be additionally improved.

Figure 5.5 shows how we expect the Objective Function to evolve over time, via consecutive local searches (doing full-carve-out moves) and perturbations (doing intersection-carve-out-moves), until the end condition is reached (i.e., full services are disjoint).

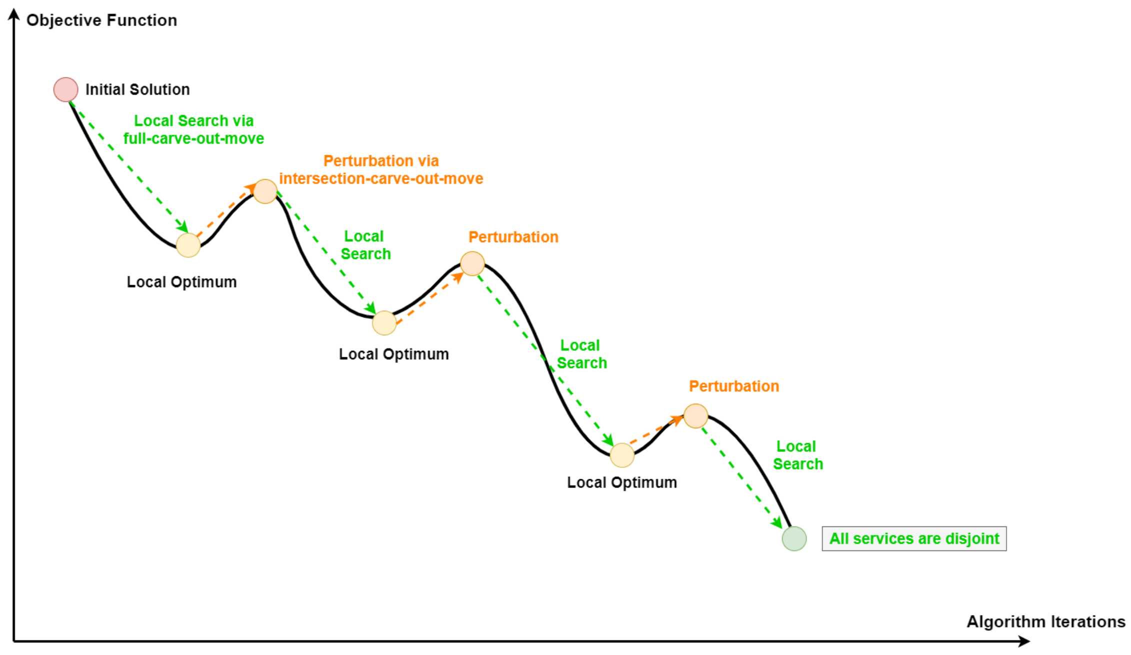


FIGURE 5.5: Expected evolution of the Objective Function

5.1.9 Solution Encoding

The algorithm has been implemented in JAVA. The different components of the solution are implemented as JAVA classes. We attempted to stay as true as possible to NS principles by defining data classes, which only contain data and convenience methods to get and set the data, and task classes used to perform actions and calculations on the data objects.

Port

Services contain ports. A port is linked to a protocol (TCP or UDP). PortRange is the class representing a range of ports with an associated protocol.

```
public class PortRange
{
private String protocol;
private int begin;
private int end;
}
```

For a single port, begin = end.

Port Frequency

Within a solution, each port will have a frequency that is equal to the number of service definitions in which this port appears. PortFrequency is the class representing the port frequency and the service definitions containing that port.

```
public class PortFrequency
{
private int portnumber;
private int frequency;
private ArrayList<String> group_occurancelist;
}
```

Port Frequencies List

The PortFrequencies class is the list of all ports existing in a solution, and for each port you identify and store its port frequency value in the solution. The i^{th} element of the array represents port i.The i^{th} element contains the port frequency information of port i. As there are TCP and UDP ports, two arrays are required for a full port frequency list.

```
public class PortFrequencies
{
private PortFrequency TCP_portfrequency[]=
   new PortFrequency[65536];
private PortFrequency UDP_portfrequency[] =
   new PortFrequency[65536];
}
```

Service

The Service class represent a service definition and contains all port ranges, UDP and TCP, associated with the service.

```
public class Service
{
private String name;
private ArrayList<PortRange> udp_ranges;
private ArrayList<PortRange> tcp_ranges;
}
```

ServiceList

The ServiceList class is the list of all service definitions of a solution.

```
public class ServiceList
{
private String name;
private ArrayList<Service> servicelistitems;
}
```

Service_DI

For each service definition, the disjointness index must be calculated and stored. The disjointness index is stored in the Service_DI class.

```
public class Service_DI
{
private Service service;
private double disjointness_index;
}
```

ServiceDIList

The ServiceDIList class is a list of all DIs of all service definitions of a solution. This list represents the neighborhood as this list will be used to iterate over. The service DI list is an ordered list, with the service with the highest DI as the first element of the list.

```
public class Service_DI_List
{
private ArrayList<Service_DI> service_DI_list;
}
```

5.1.10 Operations

The algorithm contains a number of tasks that perform actions on and with the data classes. The most important and relevant ones are listed in the following subsections.

PortFrequenciesConstructor

The PortFrequenciesConstructor will calculate the port frequencies of all ports used in all services. It takes the current servicelist as input. The result — a PortFrequenciesList — is accessible via a get-method.

Service_DI_List_Creator

The Service_DI_List_Creator will calculate the DI of all services. The inputs are the current service list and portfrequencieslist and the result — a ServiceDIList — is accessible via a get-method.

Service_Split_Evaluator

The Service_Split_Evaluator will perform a full-carve-out-split. The inputs are the service to split, the current servicelist, and the current portfrequencieslist. The result of the split, i.e. a new ServiceList, a new ServiceDIList, a new PortFrequenciesList, and the value of the objective function, are accessible via get-methods.

Service_Perturbation

The Service_Perturbation will check if a perturbation is possible and, if so, performs it. The inputs are the current servicelist and the portfrequencieslist. The results of the split, being the new ServiceList, new ServiceDIList, new PortFrequenciesList, and the value of the objective function, are accessible via get-methods.

5.1.11 The Iterated Local Search Algorithm

Algorithm 1 (see Figure 5.6) is the ILS algorithm designed according to the components described in previous sections.
The important variables are:

- sl = the service list.
- pfl = the portfrequencies list.
- sdil = the service DI list.
- of = objective function value of a solution.
- fully_disjoint = Boolean indicating if the solution is fully disjoint.
- end_of_neighborhood = Boolean indicating if the full neighborhood has been searched.
- objective_function_improvement = Boolean indicating if the objective function has improved.
- neighborhoodpointer = index of an element in the sorted neighborhood.
- service_to_split = service of the neighborhood that will be evaluated for splitting.

5.2 Brown-Field Artifact Demonstration

The artifact outlined in the previous section will be applied to firewall rule bases provided by Engie. Before an export from a firewall can be used as input for the algorithm, some pre-processing is required. We start this section by explaining these operations. We continue by discussing the components we added to the algorithm that allow the adjustments to the rule base and thus measure the impact of service disjointness on the size of the rule base. The different demonstration sets will be elucidated before they become subject to the algorithm. We conclude with a summary of the algorithm's results and a description of some in-depth behavioral characteristics of the algorithm.

5.2.1 Firewall Export Pre-Processing

The pre-processing consists of a number of steps: loading the export files in data structures, creating the Rule History List, creating the Service History List, replacing the service groups with their members, adjusting the rule base to conform to the design criteria that one rule should contain only one service, searching for identical (content wise) services and adjusting the service list and rule base accordingly, and the versioning of all services and rules. Each of these steps is elaborated in the next subsections.

```
Algorithm 1: ILS for service list normalization
sl = load_initial_solution(filename);
pfl = portfrequencies_list_constructor(sl).get_portfrequencies_list();
sdil = service_di_list_creator(sl, pfl).get_service_di_list();
of = service_di_list.get_objective_function();
fully_disjoint = FALSE;
end_of_neighbourhood = FALSE;
objective_function_improvement = FALSE;
while NOT full_disjoint AND NOT end_of_neighbourhood do
    neighbourhood = sdil.sort;
    neighbourhood_pointer = 1 (top of list)
    objective_function_improvement = FALSE;
    while NOT improvement_objective_function AND NOT fully_disjoint AND NOT end_of_neighbourhood do
        servicer_to_split = neighbourhood.get_element(neighbourhood_pointer);
        service_split_evaluator(service_to_split, sdil, pfl);
        objective_function_improvement = service_split_evaluator.get_objective_function_improved();
        if objective_function_improvement = TRUE then
            sl= service_split_evaluator.get_service_list();
            pfl = service_split_evaluator.get_portfrequencies_list();
            sdil = service_split_evaluator.get_service_di_list();
            fully_disjoint = service_di_list.is_fully_disjoint_check();
        else
            neighbourhood_pointer ++
        end
        end_of_neighbourhood = sdil.end_of_list_check(nieghbourhood_pointer);
    end
    if end_of_neighbourhood then
        service_perturbation_exists = service_perturbation.perturbation exists(sl,pfl);
        if service_perturbation_exists then
            sl= service_split_evaluator.get_service_list();
            pfl = service_split_evaluator.get_portfrequencies_list();
            sdil = service_split_evaluator.get_service_di_list();
            fully_disjoint = service_di_list.is_fully_disjoint_check();
            end_of_neighbourhood = FALSE;
        else
            end_of_eighbourhood = TRUE;
        end
    end
end
if fully_disjoint then
    PRINT "Probably the Global Optimum has been found";
else
    PRINT "Local Optimum found";
end
PRINT "Solution = " + sl.get_overview();
```

FIGURE 5.6: ILS-based algorithm

Step 1: Loading the Export Files

The firewall export consists of a set of CSV files. Not every element of those files is valuable. Via manual operations, we have removed the non-relevant data in order to limit our focus to:

- for services: the name, protocol and ports
- for service groups: the name and service members
- for rules: the number, name, source, destination, services/service groups and action.

The end result is a set of ";" delimited files.
The services file, service group file and rule base file are loaded into their corresponding data structures and can now be pre-processed.

Step 2: Preparing the Rule History

The Rule History will track all changes made to the original rule base, such as the replacement of service groups by their member services, the splitting of rules to adhere to the design rule that one rule should contain only one service, and the splitting of rules due to the splitting of a service and name changes (versioning) throughout the algorithm's run-time. The Rule History has a tree structure, wherein each level in the tree represents the changes made to the previous level.

Step 3: Preparing the Service History

The Service History will track all changes made to the original services, such as the merger of a service with another identical service (different name, but same content), the split of a service into two or more new services (after a carve-out or perturbation), and name changes (versioning) throughout the algorithm's run-time. The Service History has a tree structure, wherein each level in the tree represents the changes made to the previous level.

Step 4: Replacing the Service Groups in the Rule Base

The Service Groups aggregate services. As discussed in Chapter 2 and 4, those are sources of evolvability issues and, as such, we need to eliminate them. In all rules belonging to the rule base that contain a Service Group, we replace the Service Group by the individual members (Services) of the Service Group. Given that the rules shall change during this operation, all changes are tracked in the Rule History.

Step 5: Applying the One-Service-Per-Rule Design Criterion

From the green-field artifact, it follows that applying SoC at the rule level means that a rule should only contain one service. The previous step will certainly have resulted in a violation of this principle and there may also be additional rules that combine different services in one rule. The rule base is scanned, in search of violations of our design criteria, and rules are split accordingly. All changes are reflected in the Rule History.

Step 6: Looking for Identical Services

As firewalls allow the creation of identical service definitions with different names (examples: HTTPS, TCP, 443 and https, TCP, 443), we must scan through all the services to locate and consolidate identical services in unique service definition. Failure to do this would violate SoC on the basis that the same concern is addressed by different objects, thus hampering evolvability. As services get consolidated, rules containing those services must be adjusted. Both Service History and Rule History are updated.

Step 7: Initial Versioning of the Services

During the algorithm phase, services will be split. To ensure that each new service has a unique name, we must version the services. This is done via a simple renaming mechanism whereby the child services (i.e., services resulting from the split) of a parent service (service to split), are named as the concatenation of the name of the parent and ".x", where x is the child number. To check whether or not child services already exist in the service list, it is the content (protocol, port) that is compared, not the name. The initial versioning of the services simply adds a "V.0" to the original service name. As in a rule, the name of the services is used. All rules are adjusted according to the initial versioning. All changes to services and rules are reflected in the Service and Rule History.

5.2.2 Adjusting the Rules

Each time a sub-service gets carved-out or an intersection between two services gets split off, adjustments to the rules base are required. All the rules containing the original service must be adjusted to reflect the result of the split. The rules must also be split on the basis that rules must adhere to our SoC design criteria. Adjustments to the rule base occur at two instances of the algorithm: when a successful carve-out is performed and when a successful perturbation is performed. Algorithm 2 (see Figure 5.7) is essentially the same as Algorithm 1, with the further inclusion of pre-processing, rule adjustments, and rule/service history tracking, in bold text (while aggregating details of Algorithm 1 in a textual description).

5.2.3 Demonstration Data Sets

Engie provided exports from 15 Palo Alto firewalls in use within Belgium- and Paris-based data centers. The data centers contain multiple firewalls with different filtering strategies. We requested firewall exports that would represent the different types of filtering strategies. Additional contextual information for each firewall can be found below.

- AIMv2: Firewall used to filter in- and outbound traffic of Internet-exposed resources.
- AdminBE: Firewall used to filter traffic between data center client hosting zones and a shared management zone containing services as backup, and monitoring and system management tools. The firewall is located in the Belgium-based data center.
- AdminFR: Idem as AdminBE but for a firewall located in the Paris-based data center.
- AWSDCN: Firewall that acts as a filter between the Engie backbone network and the AWS Direct Connect (dedicated connection to AWS cloud data center in Dublin).
- HOSTING-BE-EBL: Firewall protecting the client hosting zone for Electrabel (a business unit of the Engie Group) in the Belgium-based data center.

Algorithm 2: Algorithm 1 + pre-processing, rulebase adjustments and hisotry tracking

```
//load rule base//
initial_solution = preProcess(initial_solution)
//initialize//
rb = initial_solution.getRulebase();
sh = servicerhistory.initialize();
rbh = rulebasehistory.initialize();
while NOT full_disjoint AND NOT end_of_neighbourhood do
    //reset neighbourbood pointer//
    while NOT improvement_objective_function AND NOT
     fully_disjoint AND NOT end_of_neighbourhood do
        //Select service from neighbourhood via neighbourhood
         pointer//
        //evaluate splitting of the serivce//
        if objective_function_improvement = TRUE then
            //adjust service list, port frequency list, service DI list//
            //evaluate fully_disjoint//
            sh =servicehistory.adjust(sh, sl, servicer_to_split,
             service_split_evaluator.getSplitResult);
            rb = rule_base_updater. adjust(rb, servicer_to_split,
             service_split_evaluator.getSplitResult);
            rhb = rulebasehistory.adjust((rbh, rb,
             servicer_to_split,
             service_split_evaluator.getSplitResult);
        else
            neighbourhood_pointer ++
        end
        end_of_neighbourhood =
         sdil.end_of_list_check(nieghbourhood_pointer);
    end
    if end_of_neighbourhood then
        service_perturbation_exists = service_perturbation.perturbation
         exists(sl,pfl);
        //Look for possible perturbations (service intersections)//
        if service_perturbation_exists then
            //adjust service list, port frequency list, service DI list//
            sh =servicehistory.adjust(sh, sl, servicer_to_split,
             service_split_evaluator.getSplitResult);
            rb = rule_base_updater. adjust(rb, servicer_to_split,
             service_split_evaluator.getSplitResult);
            rhb = rulebasehistory.adjust((rbh, rb,
             servicer_to_split,
             service_split_evaluator.getSplitResult);
        else
            end_of_eighbourhood = TRUE;
        end
    end
end
if fully_disjoint then
    PRINT “Probably the Global Optimum has been found”;
else
    PRINT “Local Optimum found”;
end
PRINT “Solution = “ + sl.get_overview();
```

FIGURE 5.7: Rule adjustments and rule/service history tracking in the ILS

- HOSTING-BE-ORES: Firewall protecting the client hosting zone for ORES (a former part of Electrabel, no longer part of the Engie Group), in the Belgium-based data center.

- HOSTING-BE-RAS: Firewall protecting Remote Access Resources, in the Belgium-based data center.
- HOSTING-BE-SHARED: Firewall protecting resources that are shared between various business units of the Engie Group, in the Belgium-based data center.
- HOSTING-BE-TRACTEBEL: Firewall protecting the client hosting zone for TRACTEBEL (a business unit of the Engie Group), in the Belgium-based data center.
- HOSTING-FR-COFELY: Firewall protecting the client hosting zone for COFELY (a business unit of the Engie Group), in the Paris-based data center.
- HOSTING-FR-GRDF: Firewall protecting the client hosting zone for GRFD (a former business unit of the GDF, no longer part of the Engie Group), in the Paris-based data center.
- HOSTING-FR-RAS: Firewall protecting Remote Access Resources, in the Paris-based data center.
- HOSTING-FR-SHARED: Firewall protecting resources that are shared between various business units of the Engie Group, in the Paris-based data center.
- IAF: Firewall protecting access between the resources of the user network and data center, via Identify Aware filtering rules.
- IoT-BE: Firewall protecting IoT related resources in the Belgium-based data center.

The demonstration data set also contains an artificially-created rule base entitled Demoset which was used to test the algorithm. Demoset contains as many anomalies as possible to test special conditions that could occur but that are difficult to filter out of the given exports.

5.2.4 Demonstration Results

In the following subsections, we review the demonstration environment, the summary table of the demonstrations, and the relationship between the Objective Function and the number of rules in the rule base. We continue with a discussion of the impact of the algorithm on the number of service definitions, and have a closer look at the evolution of the Objective Function during the algorithm's execution. We conclude with an example of how rule and service definition changes are tracked during algorithm execution.

Demonstration Environment

The algorithm is written in JAVA using JAVA SDK 1.8.181, developed in the NetBeans IDE V8.2. The demonstration ran on an MS Surface Pro (5th Gen) Model 1796 i5 - Quad Core @ 2.6 GHz with 8 GB of memory, running Windows 10.

Demonstration Overview

The algorithm results for the different rule bases can be found in Table 5.1 which contains the following information:

- Initial Number of Rules (NoR): number of rules as read from the firewall export files.
- Initial Number of Services (NoS): number of services as read from the firewall export files.
- Initial Number of Service Groups (NoSR): number of service groups as read from the firewall export files.
- Pre-Processing Number of Rules (NoR): number of rules after pre-processing.
- Pre-Processing Number of Unique Services (NoUS): number of unique services after pre-processing.
- Pre-Processing **OF**: the value of the Objective Function, after pre-processing and thus at start of the algorithm.
- Final Number of Rules (NoR): the number of rules after applying the algorithm.
- Final Number of Services (NoS): the number of service definitions after applying the algorithm.
- Final **OF**: the value of the Objective Function after applying the algorithm.

The algorithm performance indicators can be found in Table 5.2.

- Algorithm execution time: time required to disentangle the services and adjust the rules.
- Total execution time: time required to perform the data loading, pre-processing, disentanglement, to print the end result and log, and for the result to be displayed on the screen.
- Level 1 Iterator: number of times the outer loop of the algorithm has run.
- Level 2 Iterator: number of times the inner loop of the algorithm has run.

Rule Base	**Initial**			**After Pre-Processing**			**Final**		
	NoR	***NoS***	***NoSG***	***NoR***	***NoUS***	**OF**	***NoR***	***NoS***	**OF**
AIMV2	207	250	6	498	226	577.9	1,263	228	228
AdminBe	461	597	41	1,443	547	3,234.1	8,994	547	547
AdminFR	655	717	46	2,584	669	4,469.3	29,377	668	668
AWSDCN	13	13	1	13	13	14.9	22	13	13
Demoset	21	24	8	104	21	44.9	249	34	34
HOSTING-BE-EBL	350	304	10	759	259	877.6	3,841	256	256
HOSTING-BE-ORES	462	336	13	1,306	274	1,205.6	4,936	267	267
HOSTING-BE-RAS	20	16	0	28	16	17.5	29	16	16
HOSTING-BE-SHARED	107	120	7	223	10 7	188.7	360	106	106
HOSTING-BE-TRACTEBEL	10	5	1	10	5	5	10	5	5
HOSTING-FR-COFELY	10	9	1	16	9	9	16	9	9
HOSTING-FR-GRDF	118	46	4	213	42	50	223	40	40
HOSTING-FR-RAS	21	16	1	29	16	17,5	30	16	16
HOSTING-FR-SHARED	198	139	6	359	126	250.2	509	127	127
IAF	32	10	0	34	10	10	34	10	10
IOT-BE	23	28	0	38	25	36.5	47	24	24

TABLE 5.1: Overview demonstration results

Rule Base	**Execution Information**			
	ILS (ms)	Total (ms)	L1 Iterations	L2 Iterations
AIMV2	187,227	396,000	25	1,507
AdminBe	520,944	824,000	135	13,609
AdminFR	820,847	1,242,000	154	23,165
AWSDCN	811	18,000	2	3
Demoset	1,358	120,000	22	430
HOSTING-BE-EBL	76,039	265,000	34	2,016
HOSTING-BE-ORES	193,436	632,000	59	2,587
HOSTING-BE-RAS	99	1,000	2	3
HOSTING-BE-SHARED	35,139	202,000	12	427
HOSTING-BE-TRACTEBEL	63	1,000	2	2
HOSTING-FR-COFELY	54	1,000	2	2
HOSTING-FR-GRDF	122	1,000	3	6
HOSTING-FR-RAS	96	1,000	2	36
HOSTING-FR-SHARED	78,503	210,000	19	929
IAF	68	1,000	2	2
IOT-BE	28,731	130.000	3	19

TABLE 5.2: Performance of the algorithm

Objective Function and the Number of Rules

In Table 5.3 and Figure 5.8, we represent the relationship between the % of **OF** improvement and the number of initial rules in the rule base (NoR). Three out of the sixteen firewalls contain fully disjoint service definitions: HOSTING-BE-TRACTEBEL, HOSTING-FR-COFELY and IAF. Those are also the firewalls with the fewest rules and service definitions. The algorithm detects the full disjointness and leaves the service definitions and rule base as is.

Six out of the sixteen firewalls are fairly close to having disjoint service definitions: AWSDCN, Demoset, HOSTING-BE-RAS, HOSTING-FR-GRDF and IOT-BE. Those firewalls have a number of rules and services definitions that are below 100 (HOSTING-FR-GRDF having a number of rules a bit above 100). The total improvement of the **OF** is limited to about 25 %.

The remaining eight firewalls contain many more rules and service definitions and the value of the difference between the initial and final value of the **OF** is at least 50 %, with a maximum of 85 %. These numbers confirm that, without proper rule design criteria, the probability of getting a non-evolvable rule base drastically increases with the size of the rule base. The trend between the size of the rule base and the percentage of **OF** improvement (a good indicator for the status of the initial evolvability), should be an asymptotic function trending toward 100 %. A logarithmic regression provides a good fit.

In Table 5.4 and Figure 5.9, we represent the relationship between the % **OF** improvement and the % of extra rules (growth rule base) due to the service disentanglement algorithm. We will revisit this relationship in Section 7.3.2.

Rule Base	**Initial NoR**	%**OF** Improvement
HOSTING-BE-TRACTEBEL	10	0%
HOSTING-FR-COFELY	10	0%
AWSDCN	13	10%
HOSTING-BE-RAS	20	9%
Demoset	21	24%
HOSTING-FR-RAS	21	9%
IOT-BE	23	34%
IAF	32	0%
HOSTING-BE-SHARED	107	44%
HOSTING-FR-GRDF	118	20%
HOSTING-BE-SHARED	198	49%
AIMv2	207	61%
HOSTING-BE-EBL	350	71%
AdminBE	461	83%
HOSTING-BE-ORES	462	78%
AdminFR	655	85%

TABLE 5.3: %**OF** Improvement vs initial Number of Rules (NoR)

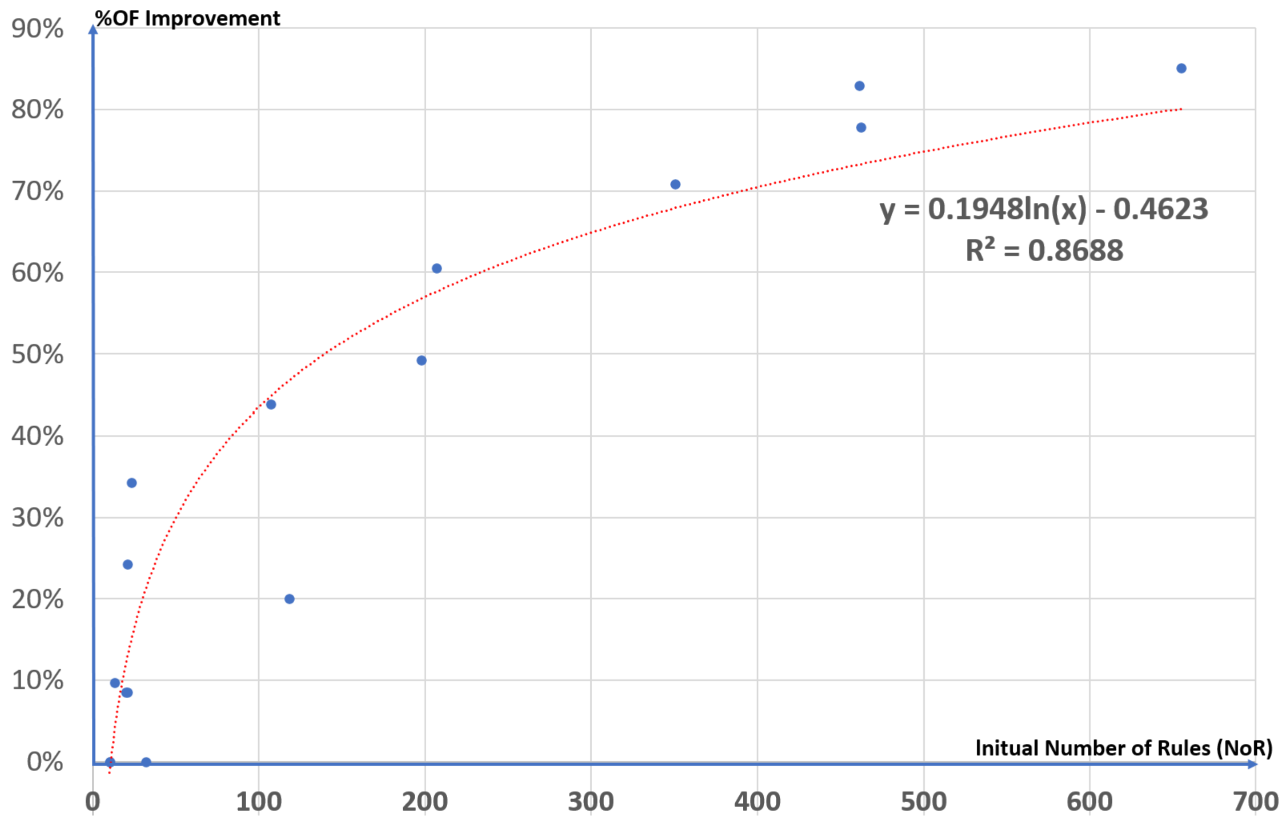


FIGURE 5.8: %**OF** Improvement vs number of rules in the rule base

Rule Base	%OF Improvement	%Growth Rule Base
HOSTING-BE-RAS	9%	4%
HOSTING-FR-RAS	9%	3%
AWSDCN	10%	69%
HOSTING-FR-GRDF	20%	5%
Demoset	24%	139%
IOT-BE	34%	24%
HOSTING-BE-SHARED	44%	61%
HOSTING-FR-SHARED	49%	42%
AIMv2	61%	154%
HOSTING-BE-EBL	71%	406%
HOSTING-BE-ORES	78%	278%
AdminBE	83%	523%
AdminFR	85%	1037%

TABLE 5.4: %Growth Rule Base vs % **OF** Improvement

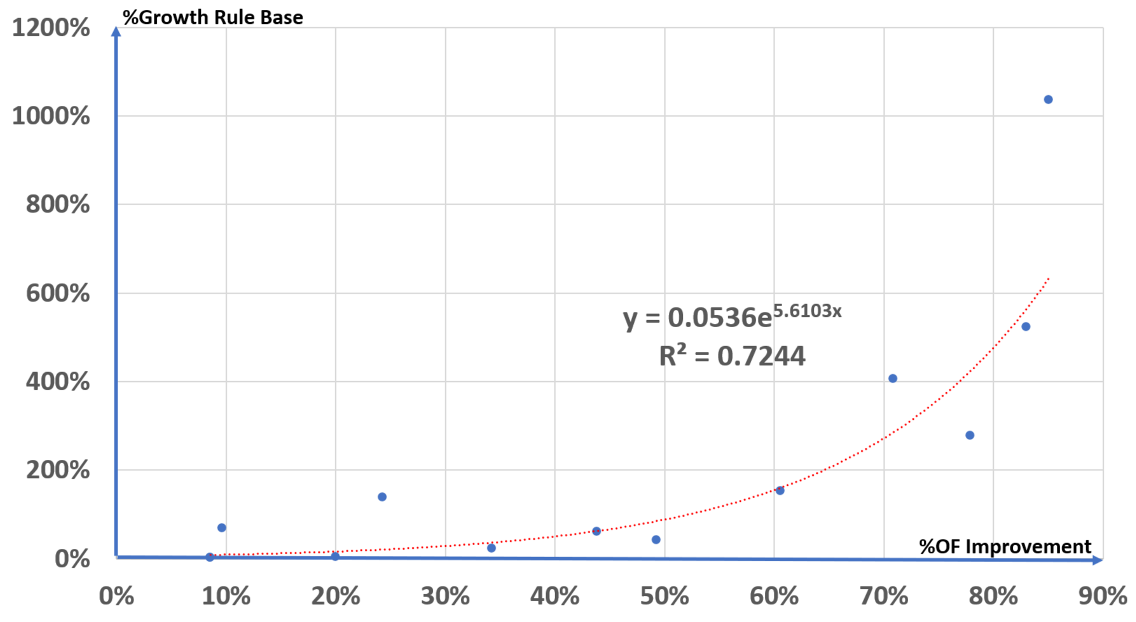


FIGURE 5.9: % extra rules vs %Δ**OF**

Impact of the Algorithm on the Number of Service Definitions

Upon examination of the number of service definitions at the end of the algorithm, we note that splitting the service definitions does not have a large impact on the total number of services. See Figure 5.10 for an overview. There is even a tendency toward the total number of service definitions decreasing slightly. It seems to be that the algorithm rearranges the ports into more suitable groups, without having the number of service definitions proliferate.

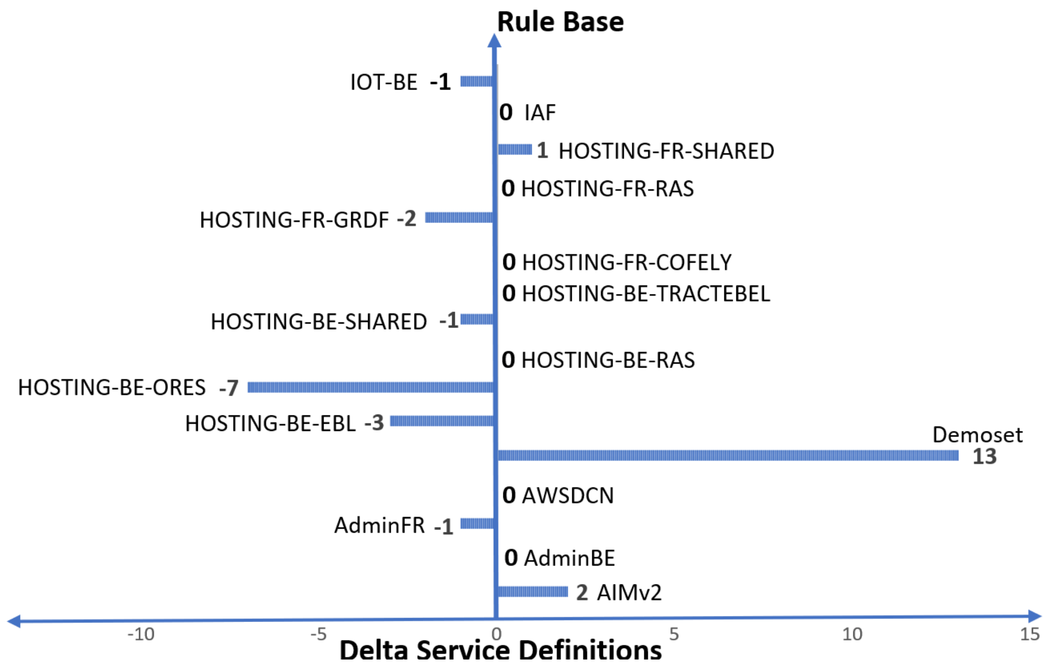


FIGURE 5.10: Impact of the algorithm on the number of services.

Evolution of the Objective Function During Algorithm Execution

To visualize what occurs during the algorithm execution, three indicators are tracked: the **OF**, the outer loop iterations, and the inner loop iterations. The "Level 1 Indicator" is the number of times that the outer **DO** loop of the algorithm has run. The indicator measures the number of times a perturbation or successful carve-out is done. The "Level 2 Indicator" (L1I) is the number of times the inner **DO** loop of the algorithm runs within a given number of level 1 iterations. Each time the "Level 1 Iterator" increments, the "Level 2 Iterator" (L2I) is reset. We plot the evolution of these three indicators against the cumulative number of level 2 iterations for two of the firewalls with a number of rules below 100, in Figure 5.11 and Figure 5.12. In Figure 5.13 and Figure 5.14 we show the evolution of the three indicators for two firewalls containing in excess of 100 rules.

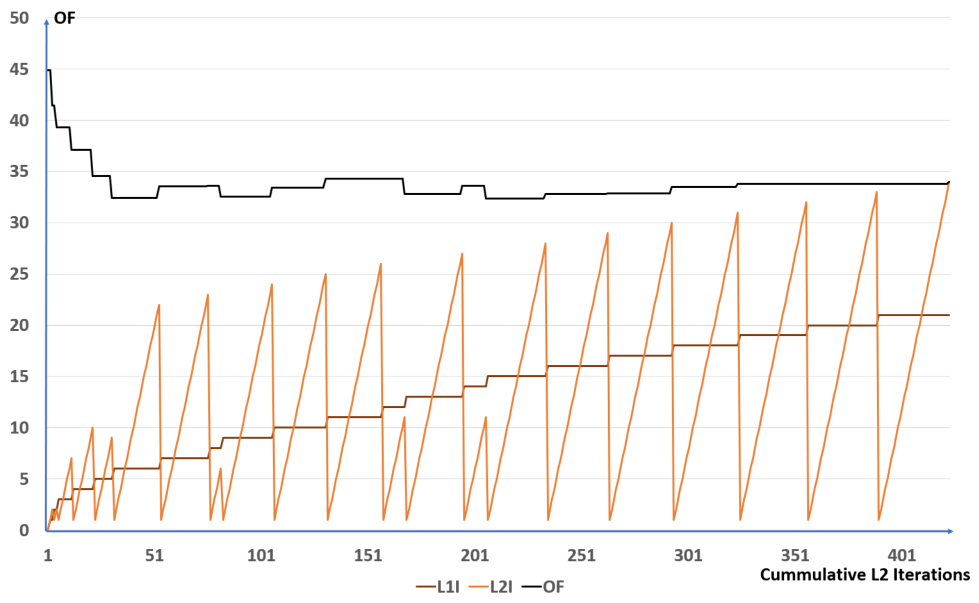


FIGURE 5.11: **OF**, L1I and L2I for the Demoset firewall.

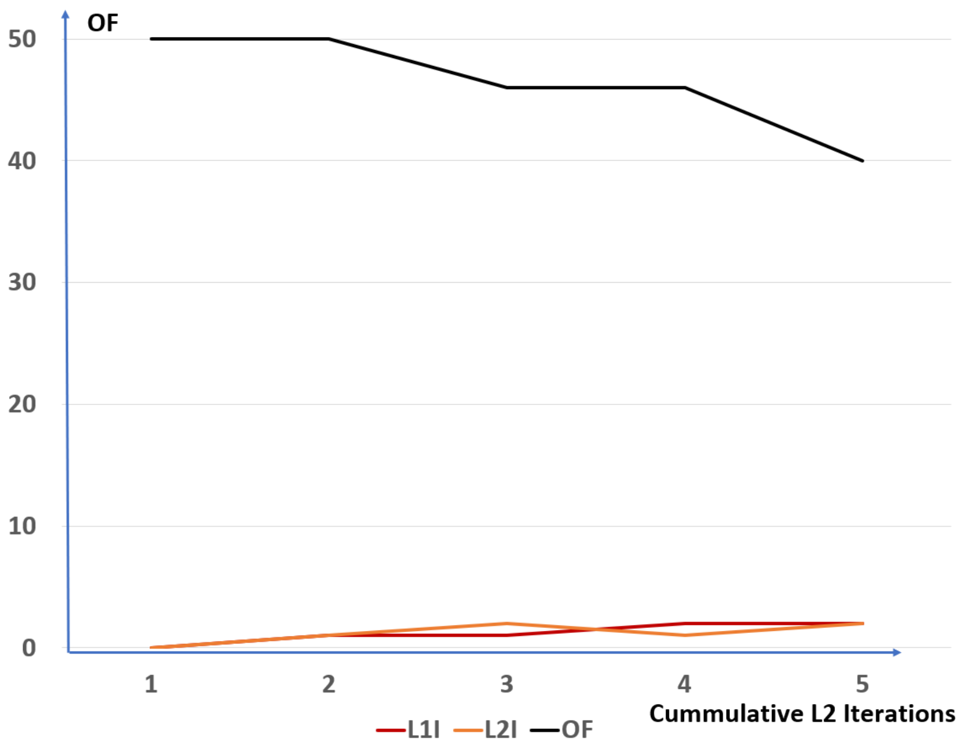


FIGURE 5.12: **OF**, L1I and L2I for the HOSTING-FR-GRDF firewall.

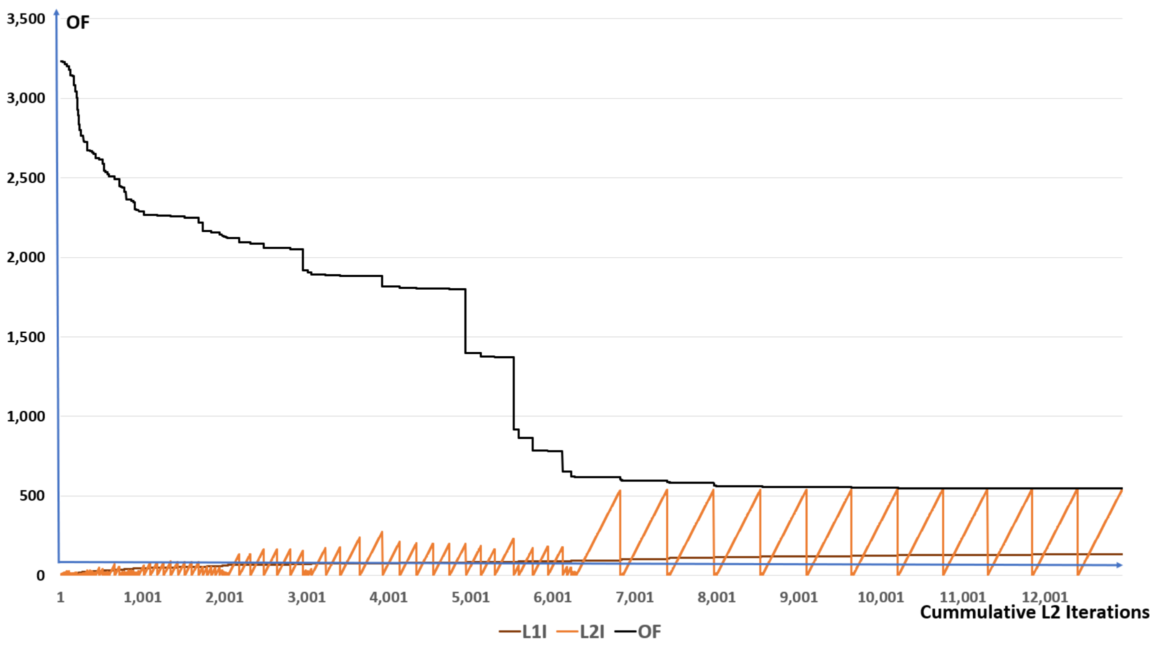


FIGURE 5.13: **OF**, L1I and L2I for the AdminBE firewall.

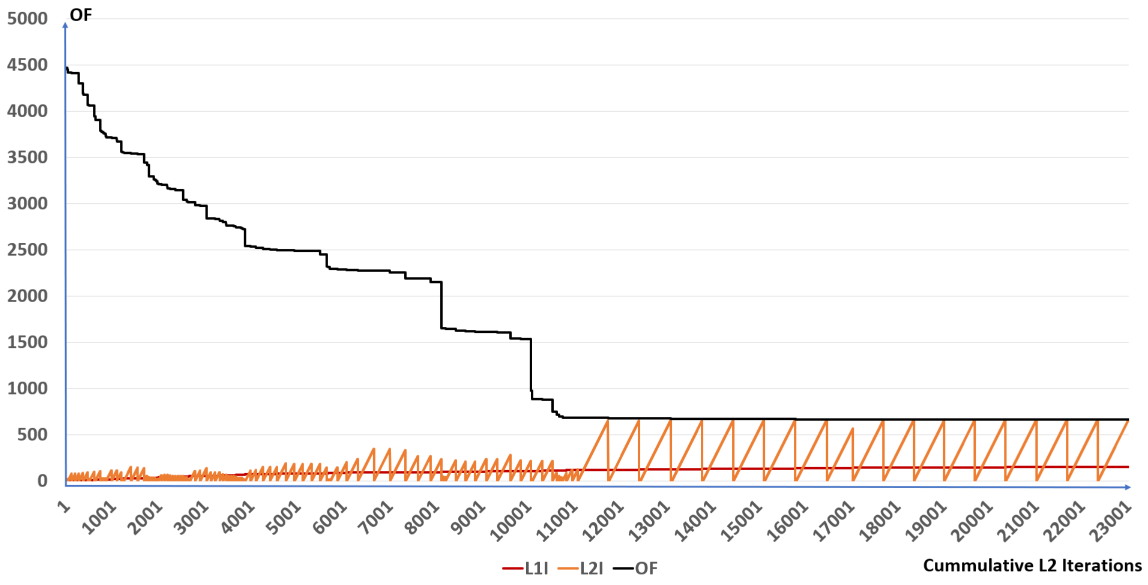


FIGURE 5.14: **OF**, L1I and L2I for AdminFR firewall.

Tracking of Rule and Service Definition Changes

The algorithm tracks all changes that are made to the rules. As an example, the log excerpt below shows the evolution of rule nr 6 from the Demoset, as provided by the algorithm at end of execution.

- In pre-processing Step 4, the Service Group "SERVICE25", is replaced by its members "SERVICE17" and "SERVICE19". The rule now has 6.1 as identifier.
- In pre-processing Step 5, the rule is split into two rules, 6.1.1 and 6.1.2 since rule 6.1 was contained in two service definitions.
- In pre-processing Step 7, rules 6.1.1 and 6.1.2 get the versioned service definitions. At this point, rule 6 is replaced by 6.1.1.1 and 6.1.2.1.
- During the ILS, the service "SERVICE17 V0" gets split into "Service 17 V0.2" and the existing service "SERVICE19 V0", and the rule 6.1.1.1 splits into 6.1.1.1.1 and 6.1.1.1.2.

```
6;R6;SERVICE25,
*6.1;R6.1;SERVICE17,SERVICE19,
**6.1.1;R6.1.1;SERVICE17,
***6.1.1.1;R6.1.1.1;SERVICE17 V0,
****6.1.1.1.1;R6.1.1.1.1;SERVICE19 V0,
****6.1.1.1.2;R6.1.1.1.2;SERVICE17 V0.2,
**6.1.2;R6.1.2;SERVICE19,
***6.1.2.1;R6.1.2.1;SERVICE19 V0,
```

In summary, rule 6 was replaced by rules 6.1.1.1.1, 6.1.1.1.2 and 6.1.2.1.

The evolution of the services is tracked in a similar manner. In the log excerpt below, the evolution of "SERVICE17" and "SERVICES19" is shown (versioning, splitting).

```
SERVICE17;UDP;40-41
*SERVICE17 V0;UDP;40-41
**SERVICE19 V0;UDP;40
**SERVICE17 V0.2;UDP;41

SERVICE19;UDP;40
*SERVICE19 V0;UDP;40
```

5.3 Discussion and Conclusion

We can conclude that an algorithm based on an ILS meta-heuristic disentangles service definitions and is able to adjust the rule base accordingly. The algorithm is an essential building block in a solution that can convert an existing firewall rule base into a rule base that is fully compliant with the green-field artifact.

It is possible that fully-overlapping rules emerge during the algorithm execution.
Example:
Take a rule R1 that has C1 as source, H1 as destination, and S1 as service.
Now take a rule R2 that has C1 as source H1 as destination, and S2 as service.
Let's consider that S1 and S2 overlap. The overlapping service is S3
Applying the algorithm would give:
– R1: C1 H1 S'1

– R2: C1 H1 S3
– R3: C1 H1 S’2
– R4: C1 H1 S3
As can be seen, R2 and R4 become identical rules which still need to be filtered out.

The demonstration has provided insight into how the Objective Function evolves during algorithm execution, as well as into the relationships between the number of initial rules in the rule base and the corresponding value of the objective function, and the number of rules at the end of the algorithm execution and the change in Objective Function.

Chapter 6

Implications of the Artifact

In this chapter we investigate the implications of the artifact (green- and brown-field) on different aspects related to the firewall. In Section 6.1, we start by looking at the impact of the green-field artifact on the firewall ontology model. Section 6.2 looks at the impact of different filtering strategies (other than Zero Trust) on the green-field artifact. As a network rarely contains only one firewall, we investigate in Section 6.3, the impact of the green-field artifact on a network with multiple firewalls. In Section 6.4, we discuss the concept of Software-Defined Network and Software-Defined Firewall. In Section 6.5 we continue examining the firewall scaling options resulting from the green- and brown-field artifact. We end this chapter with Section 6.6, in which we put forth the idea of a new artifact that allows the management of the firewall in accordance with and using the artifacts presented in this dissertation. The findings of this chapter have been published in [28].

6.1 Impact on the Ontological and Implementation Model

Application of the green-field artifact leads to a CE-free rule base with respect to a given set of anticipated changes. Adding rules to the rule base due to the activation of a new host, the activation of a new service on a host, and the addition of a new client requiring access to a service on a host all become free of CE when the artifact is used. Removal of rules due to removing a host or a service is also free of CE.

The green-field artifact enforces a new ontology for the firewall rule base. Applying a more restrictive ontology will also mean that the implementation model will be more restrictive.

A DEMO FACT model taking those restrictions into account can be found in Figure 6.1. As the model is the basis for the implementation mode, the restrictions will also be present there and will favor the usage of disjoint components and rules.

The green-field artifact enforces an implementation model that is shown in Figure 6.2. The restrictions required to ensure disjoint rules are installed. If a firewall were to use this implementation mode, it would favor the usage of disjoint rules and be evolvable under change.

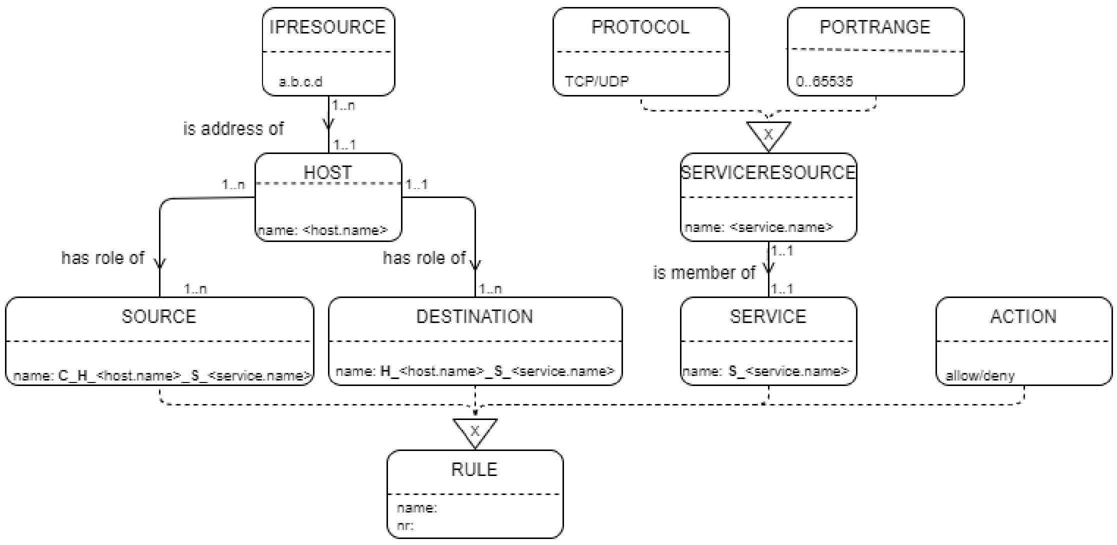
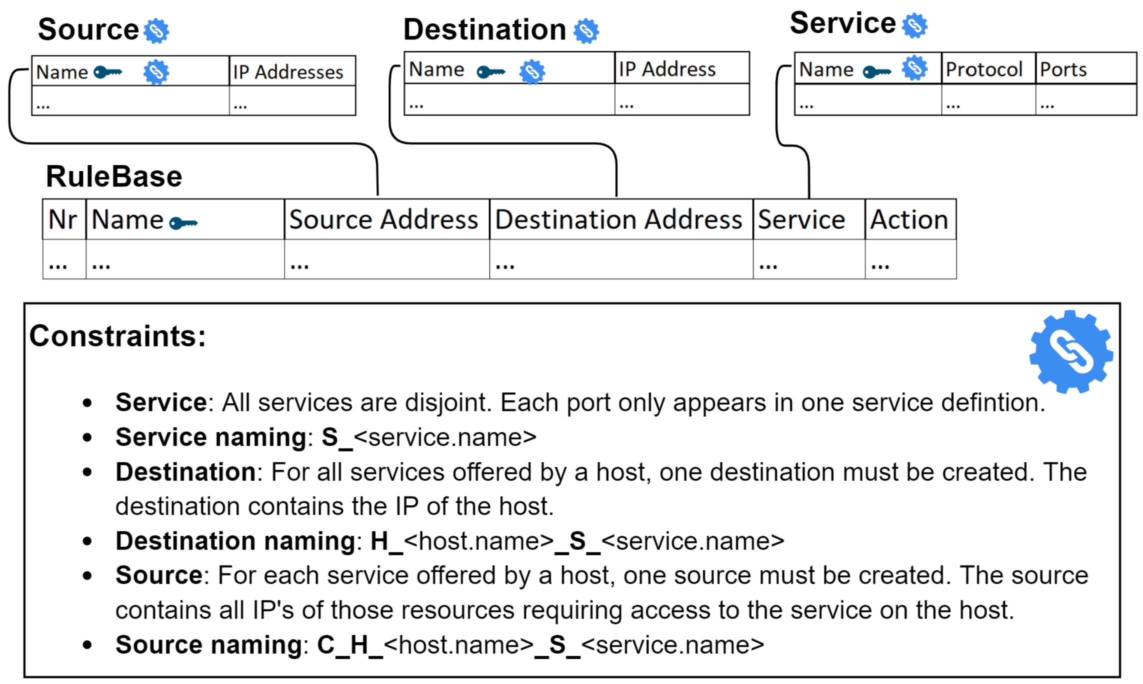


FIGURE 6.1: Ontology of an evolvable rule base


Source

Name	IP Addresses
...	...

Destination

Name	IP Address
...	...

Service

Name	Protocol	Ports
...	...	...

RuleBase

Nr	Name	Source Address	Destination Address	Service	Action
...	...	...	...	...	...

Constraints:

- **Service**: All services are disjoint. Each port only appears in one service defintion.
- **Service naming**: **S_**<service.name>
- **Destination**: For all services offered by a host, one destination must be created. The destination contains the IP of the host.
- **Destination naming**: **H_**<host.name>**_S_**<service.name>
- **Source**: For each service offered by a host, one source must be created. The source contains all IP's of those resources requiring access to the service on the host.
- **Source naming**: **C_H_**<host.name>**_S_**<service.name>

FIGURE 6.2: Implementation model of an evolvable rule base

6.2 Impact of the Filtering Strategies

6.2.1 Interconnect Filtering Strategy

In Section 3.4, we discussed the ZT filtering strategy. This strategy is useful to protect individual resources. However, firewalls are not only used to protect individual resources. Firewalls are also used to interconnect various parts of the network and regulate the traffic that is allowed to travel within the network. A ZT filtering strategy is not recommended in this case as the number of resources requiring individual protection would be too large. An Interconnect Filtering (ICF) strategy is better suited for this job. The focus is on traffic between network segments, such as VLANs, groups of VLANs or subnets, and not on the individual resources (such as hosts) connected to those network segments. The rules are different as compared to ZT rules. The level of granularity is a subcomponent of the network, not the resource. Filtering does not occur at port/service level. This means that there is one fewer parameter to enforce disjointness between the rules.

The proposed green-field artifact can still be used to create an ICF strategy-based rule base. The group objects used in an ICF strategy rule base would represent the following:

- Destination group: a group containing the IP addresses, expressed in subnets (VLAN's), that make up a logical part of the network.
- Source group: a group of IP addresses expressed in subnets (VLAN's), that comprise a logical part of the network.

The VLANs can be organized in various ways, including according to a physical location or organizational department. In the former case, there is a VLAN per building floor, and the sum of all VLANs represents the building. In the latter case, there are VLANs per organizational unit that are grouped in different areas of the building. The sum of all VLAN-based organizational units in the building represents the full building.

In ZT-based filtering, the port is the most fine-grained component where filtering is performed. Conversely, in IC-based filtering, the VLAN is the most fine-grained component where filtering is performed. As a result, for ICF, the design of the rule base needs to be structured around the VLAN.

Applying the same philosophy as in the green-field ZT artifact, we can make a green-field ICF artifact:

- Begin with an empty firewall rule base **F**. Add as the first rule the default Deny rule.
- For each VLAN requiring access control, create a destination group. Populate the group with the relevant IP address ranges representing the VLAN. The intersection between all groups must be empty. A VLAN cannot be present in two different logical parts of the network and thus cannot be present in two groups. The naming convention of those groups is as follows: **D**_VLAN-LogicalName-VLANnr

- For each VLAN requiring access control (defined in the previous step), create a source group. Populate the source group with the VLANs that require access (client VLANs representing the sources). The naming convention of these groups is as follows: **S_D**_VLAN-LogicalName-VLANnr.
- For each VLAN requiring protection create a rule whereby:
  - Source: **S_D**_VLAN-LogicalName-VLANnr
  - Destination: **D**_VLAN-LogicalName-VLANnr
  - Protocol: ANY
- Add this rule at the top of the rule base.

The **D**_VLAN-LogicalName-VLANnr groups will enforce the disjointness of the rules in the rule base. Adding and removing operations on a rule base created according to this green-field ICF artifact is compliant with the evolvability conditions.

It should be clear that this kind of filtering cannot be combined with ZT-based filtering. The disjointness of a rule cannot be guaranteed if ZT- and ICF-based rules are used in the same firewall rule base:

- Protocol: violates disjointness as ICF rules have "any" as protocol.
- Destination: ZT rules will be a subset of ICF rules and thus violate disjointness.
- Source: not used to enforce disjointness.

An example of an ICF strategy use case is the merger between two companies, each with their own network. As long as the security policies are not aligned between both companies, there is a good reason not to interconnect the two networks directly. The interconnection is best done via a firewall. The firewall will filter between IP ranges, for instance, allowing traffic between the two headquarters, but not yet between remote sites (this simplified example does not consider potential IP range overlap, NATing etc.).
Given that change is the only constant in companies, ICF-based filtering is complicated. Moves between buildings, reorganization within buildings, addition and removal of sites, and organizational changes all complicate upfront and stable segmentation of network. Segmentation rules changes and segmentation principles are mixed, and logical network segments no longer become disjoint. The result will be evolvability issues in the rule base(s) and unforeseen consequences arising from the changes.

Until this point in our thesis, we have addressed the ICF problem using a network-centric approach. As network segmentation and company reorganization can result in implementation conflicts, solutions such as identity-based firewalls emerged. In those solutions, ICF-based filtering happens based on the identity of the user. When a user tries to connect to certain parts of the network and hits an identity-based firewall enforcing the ICF strategy, the firewall will confirm the identity of the user and will filter based on this identity. This only works if:

- The firewall can establish the identity of the user associated with the source (who is working on PC with IP = x.y.z.u).
- The firewall has access to a DB containing the identities and has mechanisms to validate the identity.

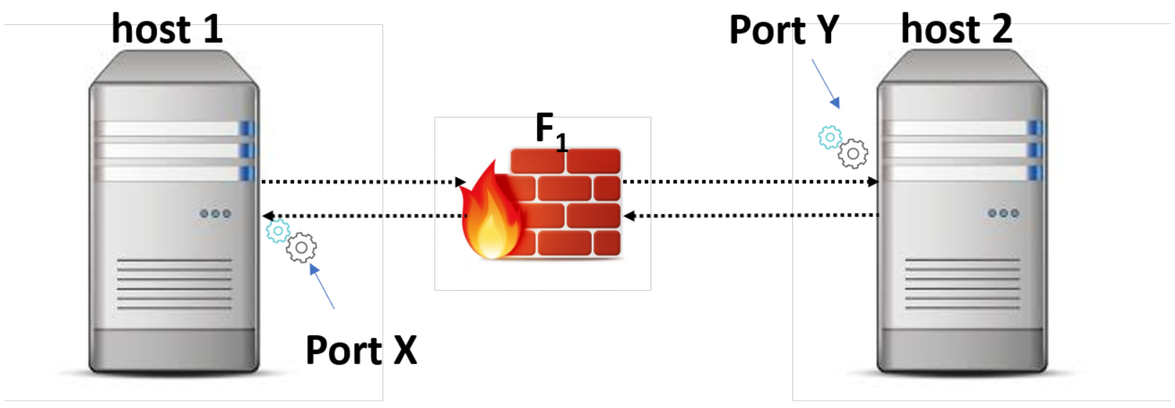


FIGURE 6.3: Inbound and outbound on a single firewall

- The firewall has a set of rules stating which identity has access to which destinations.

Such a setup is more user-centric. Access to the network is linked to the identity of the user and not the building or organizational layout. As elegant as this solution may seem, in reality, it simply shifts the problem from the network space to the identity space. Further investigation of this dimension is beyond the scope of this research. However, it is worth pointing out that user identities, identify verification (authentication), identity authorization, identity definition, identity implementation, identity and HR policies, and identity synchronization solutions are among the most complex IS systems within an IT landscape. Researching the associated evolvability issues and proposing solutions is worthy of separate PhD research. In Section 5.2.3 we have encountered one such Identity-Aware Firewall - the IAF firewall. The rules within that firewall were very simple because the actual complexity is not located in the firewall rules, but rather within the firewall identify awareness configuration.

6.2.2 Inbound and Outbound Filtering Strategy

An inbound filtering strategy — of which the ZT strategy is an example — will filter traffic close to the destination. The outbound filtering strategy will filter close to the source. From a security perspective, it makes sense to stop the traffic as early as possible on the network. On a single firewall, the notion of inbound and outbound is relative. A firewall rule base is not aware of inbound or outbound. It only knows source and destination and both can be located on either side of the firewall.

The artifact we propose was initiated based on a scenario wherein all sources are located on the lefthand side and all destination are located on the righthand side of the firewall, thus effectively implementing an inbound filtering strategy. The same artifact can be used in a single firewall setup wherein sources and destinations are located on both sides of the firewall. As long as the artifact is strictly followed, all rules will remain disjoint.
There are, however, some cautions required. Consider the case described in Figure 6.3 where a host1, located on the lefthand side of the firewall, needs to access a host2 on the righthand side. Host2 also requires access to a service offered by host1. According to the green-field artifact, the following two rules would be created:

- **R1**: **C**_**H**_host2_**S**_Y, **H**_host2_**S**_Y, **S**_Y, Allow

  - traffic from left to right
  - **H**_host2_**S**_Y contains host2
  - **C**_**H**_host2_**S**_Y contains host1

- **R2**: **C**_**H**_host1_**S**_X, **H**_host1_**S**_X, **S**_X, Allow
  - traffic from right to left
  - **H**_host1_**S**_X contains host1
  - **C**_**H**_host1_**S**_X contains host2

What the firewall will do internally is consider the content of the groups, not the group names themselves, and the rules are internally translated as

- **R1**: host1, host2, Y, Allow
- **R2**: host2, host1, X, Allow

Both host1 and host2 are members of different groups. Interchanging those groups will result in rules which do not follow the logic of the green-field artifact but that do represent the same rules inside the firewall.

- **R1**: **H**_host1_**S**_X, **C**_**H**_host1_**S**_X, **S**_Y, Allow
  - **R1**: host1, host2, Y, Allow
- **R2**: **H**_host2_**S**_Y, **C**_**H**_host2_**S**_Y, **S**_X, Allow
  - **R2**: host2, host1, Y, Allow

Group objects are used to increase the manageability of rule bases. The above makes it clear that, if used incorrectly, manageability will decrease. Groups created to represent destinations cannot be used to represent sources in rules, and the converse is also true. This is a manifestation of Separation of Concern. Representing sources and destination are different concerns and should not be mixed.

Inbound and outbound filtering are also two different concerns. In the above scenario, both are mixed on one firewall yet no immediate negative impact surfaces. The impact will become visible once there are multiple firewalls in the network. This will be discussed in the next section.

6.3 Multiple Firewalls

Until now we have largely examined networks containing one firewall. In this section we investigate the impact of multiple firewalls between the source and the destination. We begin by introducing the serial firewall filtering function and continue to investigate the validity of applying rules on all or only on certain firewalls within a multi-firewall environment. We conclude this section by revisiting outbound/inbound filtering within the context of a multi-firewall setup.

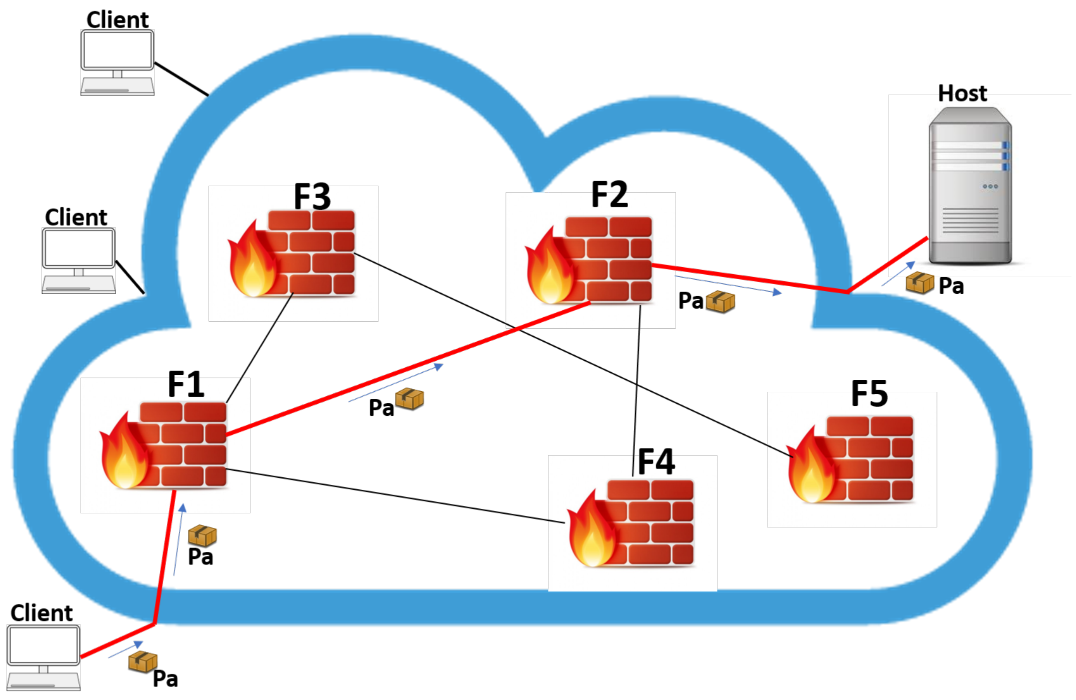


FIGURE 6.4: Multiple firewalls in a network

6.3.1 The Serial Firewall Filtering Function

Let **Pa** be a package traveling over the network.

- **Pa**.source = the IP address of the source sending package **Pa**.
- **Pa**.destination = the IP address of the destination for package **Pa**.
- **Pa**.port = the **Port** targeted on destination **Pa**.destination.

Let $\phi_{\mathbf{f}}(\mathbf{F_f},\mathbf{Pa})$ be the firewall filtering function that takes rule base $\mathbf{F_f}$ and package **Pa** as input.

$$\begin{cases} \phi_{\mathbf{f}}(\mathbf{F_f},\mathbf{Pa}) = 0, \text{ if the package is blocked} \\ \text{–> there is no rule } \mathbf{R} \text{ in } \mathbf{F_f} \text{ such that the package is allowed} \\ \phi_{\mathbf{f}}(\mathbf{F_f},\mathbf{Pa}) = 1, \text{ if the package is allowed} \\ \text{–> there is a rule } \mathbf{R} \text{ in } \mathbf{F_f} \text{ such that the package is allowed} \end{cases}$$

Let $\mathbf{f}_{\text{total}}$ be the number of firewalls in a given network.
Let $\Phi^{\mathbf{s}}{}_{\mathbf{fw}}$ be the serial firewall filtering function for a network path containing a series of fw consecutive firewalls.

$$\Phi^{\mathrm{s}}{}_{\mathrm{fw}}\ (\mathbf{Pa}) = \prod_{\mathbf{f}=1}^{\mathbf{f}=fw} \phi_{\mathbf{f}}(\mathbf{F_f},\mathbf{Pa})$$

Where:

$$\begin{cases} fw : 1 \rightarrow \mathbf{f}_{\text{total}} \\ \Phi^{s}_{fw}(\mathbf{Pa}) = 0, \text{ if } \mathbf{Pa} \text{ is blocked by at least one of the } \mathbf{fw} \text{ firewalls} \\ \Phi^{s}_{fw}(\mathbf{Pa}) = 1, \text{ if } \mathbf{Pa} \text{ is allowed by all } \mathbf{fw} \text{ firewalls} \end{cases}$$

See Figure 6.4 for a graphical representation of these concepts.

6.3.2 Applying the Rules on Some Firewalls

Within a given network, fw and $\Phi^{s}{}_{fw}$ will differ from the location of the source, destination and the internal routing of the network. Let us assume that in such a network, all firewalls have an evolvable rule base according to the proposed artifact. The addition of a new resource, called "host_new", offering service "S_new", requires the addition of new rules $\mathbf{R_{new}}$, such that "host_new" is protected according to the ZT filtering strategy.

Let us assume that $\mathbf{R_{new}}$ is only implemented on the firewalls in the path between the initially-identified sources (members of **C**_**H**_host_new_**S**_new), and destination host_new.

As time passes, the initially-identified sources require modification: a new client needs to access the host, or a client is removed from the network. According to our original green-field artifact, adding or removing a client is just a question of adding and removing the client from the group **C**_**H**_host_new_**S**_new. In our current scenario, however, this is no longer the case. If a new client has a different network path towards the host_new compared to the path in which the rule $\mathbf{R_{new}}$ was initially implemented, then the rule $\mathbf{R_{new}}$ must now be implemented on all firewalls on the path between the new client and host_new. In addition, the source group must be updated on all firewalls in all paths between all current clients and host_new. As the possible network paths are a function of the network, and the network can grow infinitely, a CE is being introduced. This is the worst kind of CE, as we will not know upfront where adjustments are required, and a full investigation of the network is required. An example of the described scenario can be found in Figure 6.5.

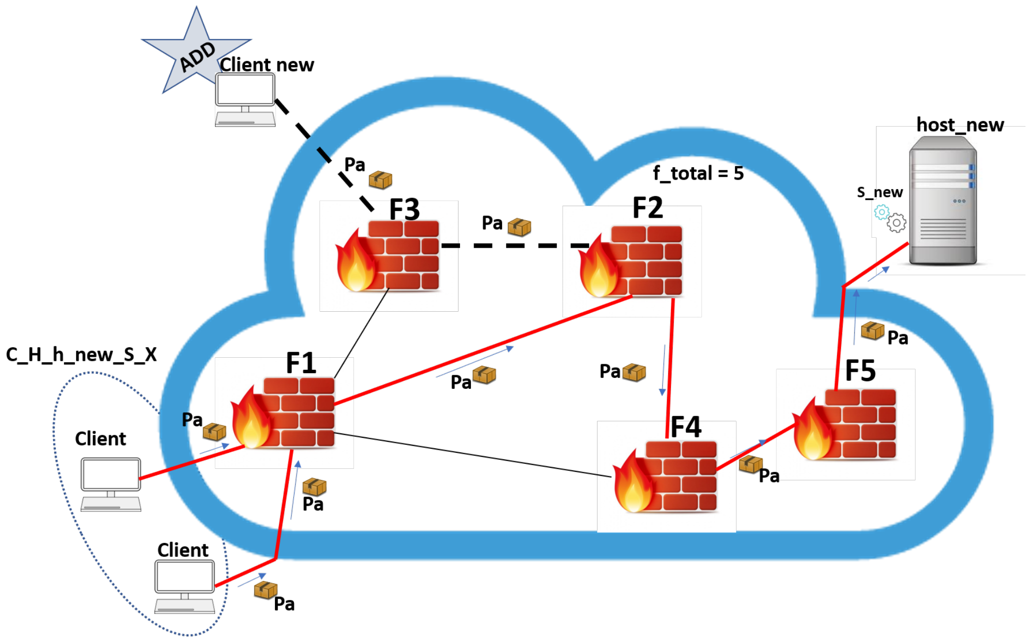


FIGURE 6.5: Apply the rules on some firewalls

6.3.3 Applying the Rules on all Firewalls

The only way to avoid the problem described in the previous section is to ensure that all firewalls contain the same rule base. All manipulations of rules must be executed on all firewalls simultaneously. As the network grows, so will the number of firewalls, and again, a CE is being introduced. This action of this CE is less aggressive, since it is now known that the manipulations are required on all firewalls. We have already discussed the impact of rule base size on the firewall. Having to duplicate all rules across the entire network will render the rule base even larger and less coherent. Rules would be added to firewalls which will never be activated, and groups would contain objects that are irrelevant to the context of that specific firewall. With such an approach, firewall manageability would decrease. In the above scenario, all firewalls address all of the same concerns. Normalized Systems theory specifies that this will have a negative impact on evolvability, as can be concluded from the above.

6.3.4 Restricting Inbound Traffic Filtering

The academic paper entitled "Minimizing the Maximum Firewall Rule Set in a Network with Multiple Firewalls" [51] is closely related to the problem we are attempting to solve. According to [51], the inclusion of firewalls in a network such that the rule base is minimal is an NP-complete problem requiring a heuristics-based solution. Although applying the heuristic-base algorithm described in [51] may minimize the rule base over all firewalls, the evolvability of those rule bases is not discussed.

In Section 6.2.2, we mentioned that a network with one firewall combines both

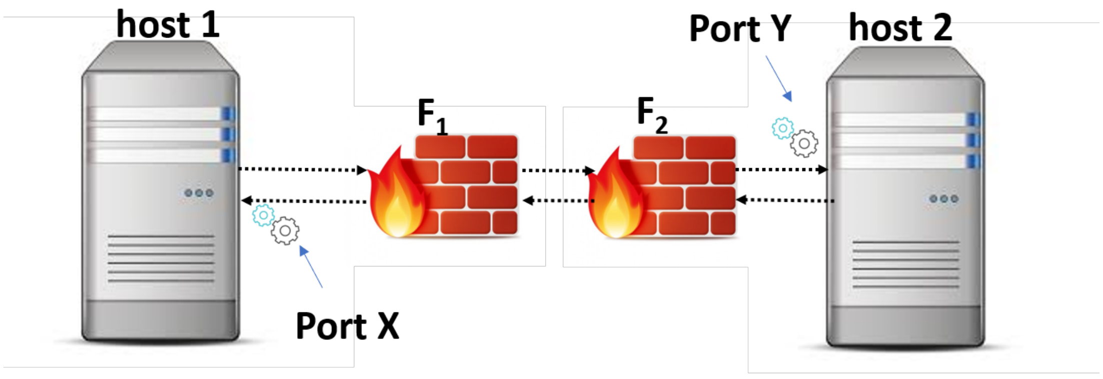


FIGURE 6.6: Back-to-back firewalls

inbound and outbound filtering rules. In a network with two firewalls that are connected in a back-to-back configuration — meaning that the firewalls are directly interconnected and no resources are located in this interconnection — inbound and outbound traffic filtering can be separated. This can be done by adding a new default rule, which states that all outbound traffic is allowed. Figure 6.6 illustrates the setup, while Figure 6.7 and Figure 6.8 illustrate the construction of the rule bases of $\mathbf{F}_1$ and $\mathbf{F}_2$. The rules **R**1 on both firewalls are disjoint with respect to the rule base in which they are located as:

- on $\mathbf{F}_1$: **C_H_**F$_1$Any**_S_**Any - represents all hosts protected by inbound traffic by $\mathbf{F}_1$

```
Rules Firewall $F_1$
    R1: C_H_$F_1$Any_S_Any, H_$F_1$Any_S_Any,
     S_Any, Allow
    R2: C_H_host1_S_X, H_host1_S_X, S_X, Allow
    R3: Any, Any, Any, Deny
    with group contents
        in R1: H_$F_1$Any_S_$F_1$Any: any
        in R1: S_$F_1$Any: any
        in R2: C_H_host1_S_X: all hosts needing
         access to host1, host2 in this case
        in R2: H_host1_S_X: host1
        in R2: S_X: port X
```

FIGURE 6.7: Rules on firewall **F1**

```
Rules Firewall $F_2$
    R1: C_H_$F_2$Any_S_Any, H_$F_2$Any_S_Any,
     S_Any, Allow
    R2: C_H_host2_S_Y, H_host2_S_Y, S_Y, Allow
    R3: Any, Any, Any, Deny
    with group contents
        in R1: H_$F_2$Any_S_$F_2$Any: any
        in R1: S_$F_2$Any: any
        in R2: C_H_host2_S_Y: all hosts needing
         access to host2, host1 in this case
        in R2: H_host2_S_X: host2
        in R2: S_Y: port Y
```

FIGURE 6.8: Rules on firewall **F2**

- on **F**$_2$: **C_H_**F$_2$Any_**S**_Any - represents all hosts protected by inbound traffic by **F**$_2$
- **C_H_**F$_1$Any_**S**_Any ∩ **C_H_**F$_2$Any_**S**_Any = ∅

and

- All source groups on **F**$_1$ are subsets of **C_H_**F$_2$Any_**S**_Any - represents all hosts protected by inbound traffic by **F**$_2$
- All source groups on **F2** are subsets of **C_H_**F$_1$Any_**S**_Any.

Thus, on both **F**$_1$ and **F**$_2$, the default outbound rule is disjoint with all other groups. We again herein note the appearance of Separation of Concern. The concern of protecting a resource is only assigned to one firewall. If assigned to multiple firewalls, evolvability issues will occur. This leads to the following design criteria:

- A firewall shall be clearly assigned to protect a set of resources. Those resources are protected by the firewall via the inbound ZT traffic filtering strategy.
- The firewall shall allow all outbound traffic from the set of resources it protects.
- If all firewalls are protecting their resources, there is no need for outbound filtering.

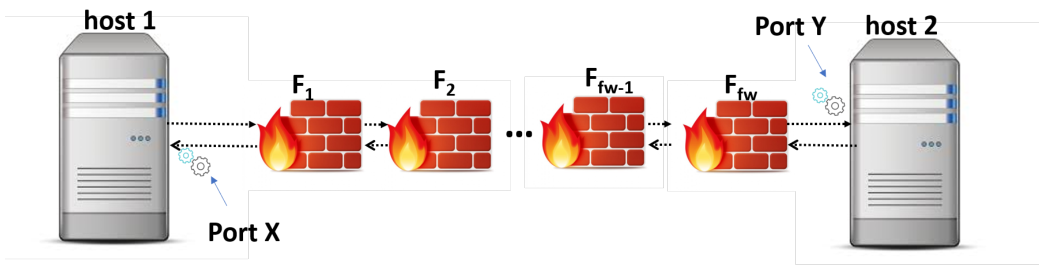


FIGURE 6.9: Path with multiple firewalls

As illustrated, our artifact can be made compliant with such a setup, simply by adding the "default Allow" rule and the creation of some additional groups.

The inverse of the above-described approach also applies: by default you shall allow all inbound traffic and filter on outbound traffic. Separation of Concerns would be respected. The artifact would need to be revised as disjointness would need to be enforced based on the combination of Service and Source rather than of Service and Destination. The same reasoning applies for the inbound default Allow rule. Although technically possible, this filtering strategy would be confusing. By comparison, consider the following scenario: A city needs to close an entry road due to construction works. Traffic will be blocked as close to the construction site as possible (inbound filtering). It is impossible to block all roads, which could potentially lead to the city (outbound filtering).

6.3.5 Apply Inbound Traffic Filtering to More Than Two Firewalls

What happens when there are more than two firewalls between two resources? Figure 6.9 illustrates the setup. When we apply the design criteria from the previous section, we must conclude that $\mathbf{F}_2$ to $\mathbf{F}_{\text{fw-1}}$ are not allowed to filter inbound traffic. Those concerns are already assigned to $\mathbf{F}_1$ and $\mathbf{F}_{\text{fw}}$. Firewall $\mathbf{F}_2$ to $\mathbf{F}_{\text{fw-1}}$ must handle other concerns such as:

- **choke-point**: Use a firewall as a kind of valve: Allow all or Deny all. This comes in handy in case of network intrusions, and traffic needs to be blocked asap in a simple way, without impacting existing routing.
- **Interconnect filtering strategy**: use those firewalls to control connectivity between network segments (see Section 6.2.1).

Note that for the Interconnect Filtering strategy, Separation of Concern must also be respected. An interconnect firewall should be assigned to handle the interconnection of assigned ranges, and no other interconnect firewall should filter on the same ranges. This again can quickly become complex and evolve into an NP-complete problem. Measured advice would be to refrain from the usage of interconnect and choke-point firewalls, and limit the number of firewalls in any network path to the greatest possible extent.

6.4 Software-Defined Network/Firewall

Extending the inbound filtering strategy discussed in the previous section to its logical limit equals providing each resource with its own firewall. This is what occurs in a Software-Defined Network (SDN) combined with a Software-Defined Firewall (SDF). In a SDN, the network layer is virtualized inside a virtualization layer called the hypervisor. The SDN is decoupled from the actual underlying physical network. In the hypervisor layer, network components such as routers, switches, VLANs, load balancers, and firewalls are all defined entirely in software. To each virtual host defined in/on the hypervisor, a virtual firewall can be attached. A package does not enter the network layer of the virtual hosts unless it successfully crosses the firewall. SDF is better compared to an Operating System (OS)-based firewall (such as IP tables or Windows Group Policies). OS-based firewalls can only perform their filtering function if the package is already "inside" the host.

For an SDF, the rule base is configured via policies. A policy defines the protocol and port that can pass though the firewall. The policies are assigned to the firewall. Given that the firewall is attached to only one host, disjointness for the destination, by default, is guaranteed. However, multiple policies can be attached to one host, and within those policies, overlaps and conflicts of protocols/ports and actions may exist. Again, the conscious restriction of design space is required.

The previously-proposed artifact can be adjusted for use within an SDN context by creating policies for Software-Defined Firewalls. The policies are the equivalent of the Service Groups. They must be as fine-grained as possible. For each service exposed on a host, a policy must be created. Policies may not overlap. Rather than creating a destination group, the policies are attached to the host. As many policies are attached to the host as there are services offered by the host. Access to the host is provided by allowing explicit access of a client to the host. This corresponds to creating a client group as defined in the artifact. Group membership means you can access the host, and the policy attached to the host will authorize protocols and ports.

A Software-Defined Firewall in a Software-Defined Context is the best way to guarantee the ZT filtering strategy. SDF also offers the most evolvable architecture. Add/remove of hosts to the hypervisor automatically adds/removes the associated host firewalls. Add/remove of rules means add/remove of policies and/or attach/detach of policies. If the policies are created according to the proposed artifact, evolvability is guaranteed.

6.5 Implication of the Artifact on Firewall Scaling

In an evolvable rule base, all the rules are disjoint from one another and each network package can only hit one rule. This rule can be located in the beginning or near the end of the rule base. As there is only one rule that can be hit, the rule base may be split into multiple pieces and distributed in parallel across different firewalls.

Let **F** be a firewall rule base containing only disjoint rules created according to the green-field artifact. As visualized in Figure 6.10, **F** can be split into **f**w sub rule bases, which are spread over **f**w parallel firewalls. Each of the **f**w rule bases conclude with the “Default Deny” rule.

A network package will attempt to traverse each firewall, but only one of the firewalls has a rule it can hit.

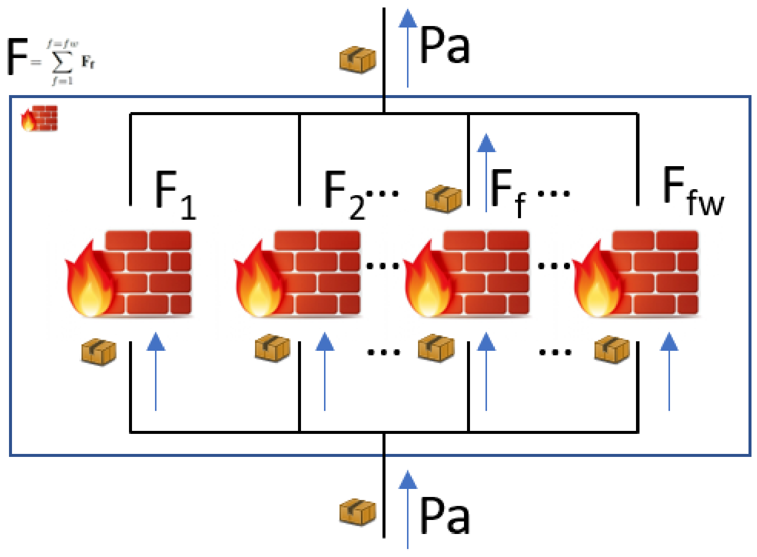


FIGURE 6.10: Scaling of firewalls with an evolvable rule base

$\mathbf{F} = \sum_{f=1}^{f=fw} \mathbf{F_f}$

Let $\phi_\mathbf{f}(\mathbf{F_f},\mathbf{Pa})$ be the firewall filtering function that takes rule base $\mathbf{F_f}$ and package **Pa** as input.

- $\phi_\mathbf{f}(\mathbf{F_f},\mathbf{Pa}) = 0$ if the package is blocked - there is no rule **R** in $\mathbf{F_f}$ such that the package is allowed
- $\phi_\mathbf{f}(\mathbf{F_f},\mathbf{Pa}) = 1$ if the package is allowed - there is a rule **R** in $\mathbf{F_f}$ such that the package is allowed

Let $\Phi^\mathbf{P}{}_\mathbf{fw}$ be the parallel firewall filtering function. Then:

$\Phi^\mathbf{P}{}_\mathbf{fw}(\mathbf{PA}) = \sum_{f=1}^{f=fw} \phi_\mathbf{f}(\mathbf{F_f},\mathbf{Pa})$

Where:

- $\Phi^p_{fw}(\mathbf{Pa}) = 0$, if **Pa** is blocked by all of the fw firewalls.
- $\Phi^p_{fw}(\mathbf{Pa}) = 1$ if **Pa** is allowed by one of the fw firewalls.
- $\exists ! F_f \in F$ for $f = 1 \rightarrow fw \mid R \in F_j$

A rule base that exclusively contains disjoint rules can scale horizontally (i.e., employ parallel firewalls). Firewalls with a non-evolvable rule base can only scale vertically (i.e., employ a larger firewall). Scaling, however, does not come without significant cost. Modern firewalls allow virtualization, but each virtual instance comes at a cost as well. In addition to the horizontal scaling possibilities of an evolvable rule base, the performance of an evolvable rule base can be boosted by moving the most frequently used rules to the top. Check Point, a firewall vendor, suggests locating the rules that are most frequently hit (and applied) at the top of the firewall table. In a rule base that is order-sensitive, this is a real issue. In a rule base that is not order-sensitive, one could monitor the firewall to determine which rules are hit most and then prioritize those rules without having to worry about the potential impact to other rules. Doing this dynamically would be even more powerful as the firewall would be able to reorganize its rules according to variable daily traffic.

6.6 The Firewall Rule Base Analyser and Normalizer System

As the firewall provides considerable design freedom which could potentially lead to evolvability issues, firewall management should be undertaken outside of the firewall, ideally using a specialized tool that incorporates the artifacts discussed in this dissertation. We characterize this tool as a Firewall Rule Analyser and Normalizer System or FRANS (see Figure 6.11). Such a tool would ideally have the following features:

- Enforces the usage of the green-field artifact.
- Analyzes an existing rule base — measures disjointedness levels — with the brown-field artifact.
- Converts an existing rule base into an evolvable rule base using the brown-field artifact.
- Will centrally manage all definitions: services, sources, destinations, rules.
- Provides full traceability on all changes performed on the definitions.
- Makes firewalls scale horizontally.
- Changes the rule order dynamically to increase performance.

All firewall rule management activities are done in the tool as opposed to via firewall management consoles. As modern firewalls publish their management functionalities via APIs, the tool can use these APIs to change rules and objects.

The creation of the fine-grained rule base by humans is an issue. The green-field artifact defines criteria for groups and rules that need to be followed strictly. The creation of a catalog of all possible services is required. For standard services and tools, lists of assigned ports/protocols and international standardization organizations related to the Internet (e.g., iana.org) exist and may be reused. Note that the Palo Alto firewall includes one such standard list of Application Objects (see Section 5.2.3). Unfortunately, the same Palo Alto firewall allows the overwriting and double definitions via Service Objects.

FRANS should expand the firewall rules in the fine-grained format, in accordance with the naming conventions. Checks must also be performed against the group definitions and content in accordance with the green-field artifact and via a user-friendly interface. With this configuration, the tool could then push the rules towards the firewall, which would effectively separate the management from the implementation of rules. Such tools exist on the market. Examples include Algosec, Tufin, Firemon. However, none of those tools consciously restrict the design space for the purpose of enforcing the creation of an evolvable rule base.

While defining a rule for each service may be considered cumbersome, it is possible to create roles such as "monitoring and management" (i.e., establishing which is a grouping of smaller, disjoint services) in order to mitigate this. In this example,

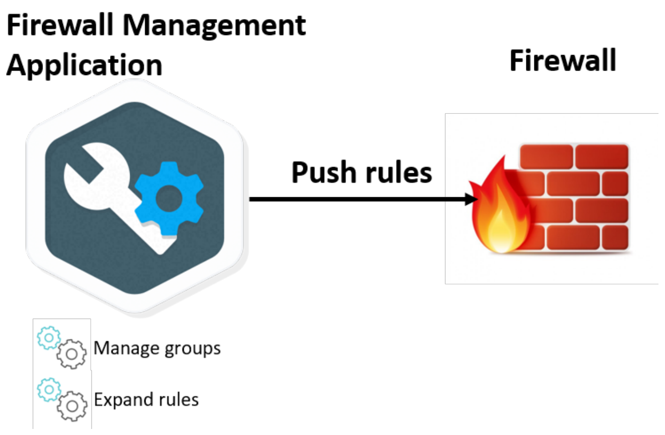


FIGURE 6.11: Firewall management tool

the firewall administrator could create a rule specifying this "monitoring and management" role to express that the server needs to allow access to all monitoring and management services. The tool would ideally expand these roles into the individual rules for each disjoint service. Examples:

- "Monitoring and Management" = SSH + SFTP + FTP + SMTP + TELNET
- Host = x
- Rule : C_Hx_SMaM; Hx_S_MaM; S_MaM; allow
- Will be expanded to :
  - C_Hx_S_SSH; Hx_S_SSH; S_SSH; allow
  - C_Hx_S_SFTP; Hx_S_SFTP; S_SFTP; allow
  - C_Hx_S_FTP; Hx_S_FTP; S_FTP; allow
  - C_Hx_S_SMTP; Hx_S_SMTP; S_SMTP; allow
  - C_Hx_S_TELNET; Hx_S_TELNET; S_TELNET; allow

The brown-field artifact should be included in the tool to read and analyze an existing rule base. The Disjoint Index of all groups and the total value of the Objective Function can be calculated. These are important indicators for the level of evolvability and the impact the firewall normalization process will have on the size of the rules base. Highly non-evolvable rule bases may require additional firewall infrastructure to allow horizontal scaling. The tool could create new firewall instances on a virtual infrastructure or spin up new cloud based firewalls on a cloud platform.

FRANS should convert existing rule bases into evolvable rule bases and deploy those on the firewall infrastructure.

As FRANS should be a central firewall management platform, it could compare at all times the defined policy in the tool to the active policy on the firewall. This would allow detection of rule adjustments made directly on the firewall and even make firewall rule bases immutable. In FRANS, additional information reflecting why rules are deployed and links with application management tools could be made in order to allow centralized and easily understandable security documentation.

Chapter 7

Evaluation and Discussion

In this chapter we evaluate and discuss the artifacts. The Design Science Framework is used as the reference model, in combination with classic Research Methodology practices as promoted by Campbell, Cook and Shadish [52]. We start by assessing the results and the performance of the artifacts and point out existing shortcomings of the artifacts in Section 7.1 . We continue in Section 7.2 by pointing out the applica-bility and required evolution of the artifacts for the environment. In Section 7.3, we evaluate and discuss the Rigor Cycle, by applying the principles of validity as pre-sented in [52]. We end this chapter by outlining our contributions to the knowledge base, concluding the evaluation and discussion of the Rigor Cycle in Section 7.4, and by positioning the work as Enterprise Engineering instruments in Section 7.5.

7.1 The Artifacts

7.1.1 Green-Field Artifact Limitations

At its core, the green-field artifact represents a method and a set of design criteria for use when setting up a firewall. Even if one has the luxury of setting up a brand-new firewall, it would be difficult to manually apply the green-field artifact. This needs to be accompanied by tooling, as explained in Section 6.6.

As outlined in Section 6.2, the green-field artifact can be adjusted to support dif-ferent filtering strategies.

7.1.2 Big O of the Brown-Field Artifact

The **Big O** of an algorithm expresses the algorithm's complexity, calculated based on the worst-case scenario in terms of the number of operations required in function of the size of the problem to be solved. This formula reflects the worst-case effort required to complete the algorithm execution.
The algorithm contains two nested loops that both can iterate over the full neighborhood, meaning the algorithm will be quadratic with respect to the size of the neighborhood.
The number of operations performed in the innermost loop, such as Service_DI_list_Creator, Service_split_Evaluator are also proportional to the size of the neighborhood.
We may thus conclude that the **Big O** of the complete algorithm is cubic - $O = n^3$, where **n** is the size of the neighborhood (= size of the solution = the number of service definitions).

Rule Base	Number of Unique Services (NoUS)	Algorithm Execution Time (ms)
HOSTING-BE-TRACTEBEL	5	63
HOSTING-FR-COFELY	9	54
IAF	10	68
AWSDCN	13	811
HOSTING-BE-RAS	16	99
HOSTING-FR-RAS	16	96
Demoset	21	1,358
IOT-BE	25	28,731
HOSTING-FR-GRDF	42	122
HOSTING-BE-SHARED	107	35,139
HOSTING-FR-SHARED	126	78,503
AIMv2	226	187,227
HOSTING-BE-EBL	259	76,039
HOSTING-BE-ORES	274	193,436
AdminBE	547	520,944
AdminFR	699	820,847

TABLE 7.1: Brown-field artifact performance

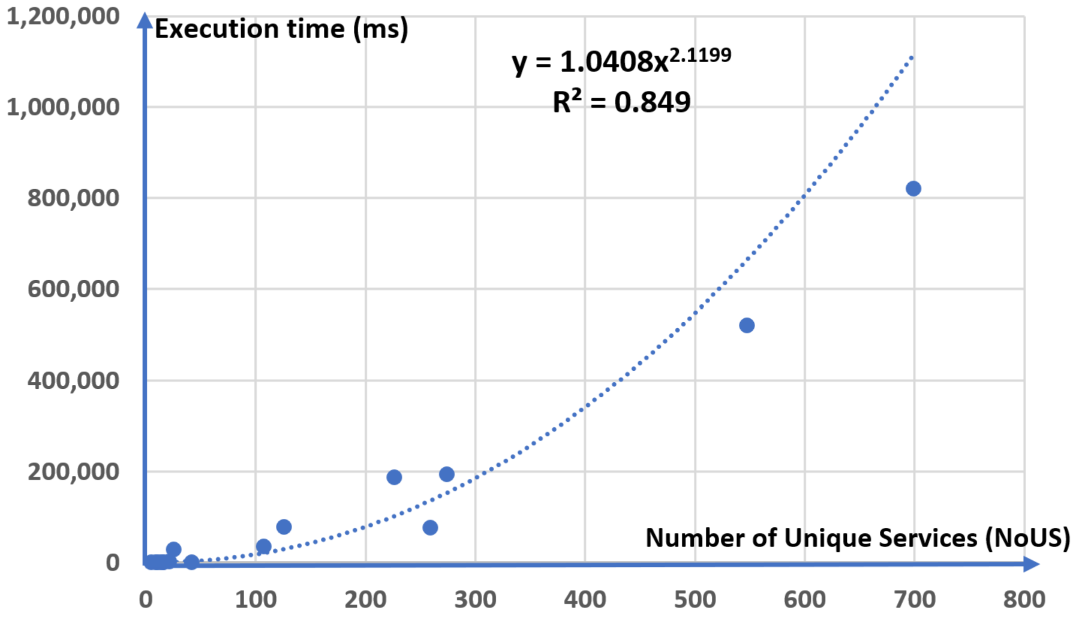


FIGURE 7.1: Brown-field artifact performance

7.1.3 Performance of the Brown-Field Artifact

Algorithm execution time is measured as the time it takes to disentangle the services after pre-processing. Figure 7.1 shows the relationship between the initial size of the neighborhood (number of unique services) and algorithm execution time. The exponent of the power function is a bit above two. This is consistent with the **Big O**, where we expected a worst-case exponent of three.

Measures could be taken to ensure better algorithmic performance. The innermost loop iterates over all services until it locates one that contains subgroups. All services that already have a **DI** of 1 should not be further investigated. As the neighborhood is sorted from high to low **DI** at the start of the inner loop, the inner loop could stop as of the first service where a **DI** of 1 is encountered. According to meta-heuristics, this value represents a form of algorithmic memory, indicating parts of the neighborhood that can no longer improve and should thus not be investigated.

7.1.4 Global Optimum

Does the heuristics-based algorithm establish the Global Optimum? It is quite difficult to formally prove that heuristic algorithms always provide the most optimal solution. After all, the full solution space of all possible groups combining all possible ports is exponential (see combinatorics) and quickly becomes impossible to fully search.
We do think that, given the initial solution, we have succeeded in converging on the most optimal solution. Sub-optimal solutions always will have either subgroup and/or overlapping groups. The algorithm filters out all subgroups in the inner loop and, if no additional subgroups are found, it searches for overlaps, after which it again scans for subgroups. As both inner and outer loop iterations search the entire neighborhood, all possible subgroups and overlaps are located and eliminated. While we are not presently able to provide formal proof, we nonetheless believe that, from a given initial solution, the set of services that are disjoint and maximum in size is found.

7.1.5 Brown-Field Artifact Limitations

Naming of the Services

The brown-field artifact tracks all changes in the service definitions by means of continuously changing the name of the services via a versioning mechanism. Although the end result is disjoint services according to the green-field artifact, the naming of those services is not compliant with the naming convention put forward in the green-field artifact. A mechanism to generate meaningful names is currently lacking.

Non-Existing Service Definitions

During simulations involving the brown-field artifact, it was detected that the rule base contained services that were not part of the service definitions. On the Palo Alto firewall, those were "any" and "application-default" and on the other firewalls only "any". The "any" service contains all ports and is a special case. To be compliant with the green-field artifact, there should be only one service-related rule containing "any", and that rule must be located at the end of the rule base. An elegant solution for how to treat the rules containing "any" as a service is not yet a dimension of the brown-field artifact.

When a service is set to "application-default" on a Palo Alto firewall, the firewall will search the ApplicationGroup list for the content of the "application-default" group, but will not search the Service or ServiceGroup list (see Section 2.5). The brown-field artifact does not integrate the Application and ApplicationGroup definitions. All rules containing "application-default" as service, are currently left untouched.

Destinations and Sources

The brown-field artifact does not transform the destination and source definitions in accordance with the green-field artifact. Recall that, for each service, there should be as many destination groups created as there are hosts offering the service and as many source groups as there are destinations offering the service. To realize this, scanning of the existing rule base is required in order to extract this information.

This is not currently a dimension of the brown-field artifact.

Strict application of the green-field artifact dictates that rules with multiple destinations are subject to splitting whereby there is one rule per destination. This is also not a current dimension of the brown-field artifact.

We have seen that destinations can not only be hosts but also subnets/VLANs. We have also seen that mixing filtering at different levels in one rule base (ZT level at the Host or ZT level at the VLAN/subnet) is not advantageous from an evolvability point of view. Different filtering strategies must be split among different firewalls, which can then be configured in a serial-chain pattern, ranging from coarse- to fine-grained filtering. The current brown-field artifact does not include detection of differences in filtering strategy, nor does it split those rules in a new rule base.

7.2 The Application to the Environment

The previous section discussed current shortcomings of the artifacts. These limitations make the artifacts not yet production-ready. Even if an artifact such as FRANS would be available, could we be sure that it delivers the value that is assumed? Evaluating this question requires a different research path. The artifact would become a socio-technical system and additional research aspects would come into play (adoption, acceptance, integration in existing processes, change management, etc.). Addressing these dimensions is beyond the scope of this dissertation.

What we may however know with certainty is that the impact of the artifact on the size of the rule base will evoke resistance to its implementation from firewall administrators. A large rule base is associated with complexity and performance issues, similar to the widespread belief that a fine-grained modular software structure is complex and thus must be accompanied by performance issues. We argue that these arguments are flawed. The complexity argument can be disproved by the counter argument that something large-but-structured is less complex than something small-but-unstructured. A useful metaphor to illustrate this is that of a wall that is made of brick and arranged using a simple pattern that repeats over and over. There is no complexity. Similarly, the rule structure we propose in the green-field artifact is a simple brick in a firewall that is repeated over and over. It is possible to know what can be expected from each rule. This is not the case, however, when there is no conscious design behind the rule base and an allowance for unstructured growth. In the latter case, the second law of thermodynamics predicts that the system would continue to evolve towards greater entropy.
The second argument of performance is disproved by the fact that an evolvable rule base can scale horizontally while a non-evolvable rule base cannot. The counter argument could then be that additional firewalls would increases the cost of security, however this reasoning is incorrect. Rather, the larger evolvable rule base will reflect a Zero-Trust policy, while the same cannot be said about a non-evolvable rule base. High-level security policies require a specific level of filtering. Not applying that level of filtering translates to non-compliance with the policy and an active erosion of security. The evolvable rule base will thus reveal the true cost of higher-level security policies.

7.3 The Usage of Existing Knowledge and Methodologies

In this section, we elaborate on two aspects of the Rigor Cycle: the usage of existing knowledge bases and the usage of methodologies.

7.3.1 Existing Knowledge Base

This research uses available prior research on the relationships between rules and firewall a nomalies. To the best of our knowledge, previous academic publications have unfortunately not addressed the issue of evolvability in a structured way. The findings of this work are not entirely unique in the sense that anomaly-avoidance knowledge exists. Despite this, clear derived design criteria and pro-active design is lacking. The Normalized Systems theory grounds the concept of evolvability and we were able to translate this concept into measurable indicators: the **DI** and **OF**. The field of meta-heuristics provides guidance on how the optimization problem associated with the brown-field artifact should be handled.

7.3.2 Methodologies

The Design Science Methodology allows and even promotes the integration of other methodologies as a means of improving the rigor of the research. In Chapter 4 we demonstrated the green-field artifact by instantiation. The brown-field artifact demonstration is an actual experiment which aims to validate the correct functioning of the algorithm and assess the impact the artifact has on the size of the rule base.

The construction of good experiments is well documented in the research methodologies used across the social sciences. More specifically, we refer to the work of Campbell, Cook and Shadish [52] concerning the types of validity related to an experiment. In the following subsections, we shall investigate the different validity types relevant to our experiment in order to properly establish the limitations of the experiment.
For this purpose, all definitions of validity types and validity threats derive from [52].

Experiment Description

We will now examine the question "What is the impact of the service group disjointness level of a rule base on the size of the aforementioned rule base after application of the brown-field artifact?". We shall define the Services Disjointness Index (**SDI**) as the ratio between the value of the objective function **OF** and the number of services **S**.

$$\mathbf{SDI} = \frac{\mathbf{OF}}{\mathbf{S}}$$

SDI is 1 in a rule base exclusively containing disjoint services and greater then 1 if the rule base contains non-disjoint services. We would like to know whether or not we may determine the increase in number of rules as a result of the application of the brown-field artifact, based on the initial value of **SDI**.

The **SDI** is a fairly accurate measure for the statistical entropy of a rule base. The macro-state is the number of services in a rule base, the micro-states being the number of possible services within a rule base. An evolvable and perfectly stable rule

Rule Base	**SDI**	**RNIR**
HOSTING-BE-TRACTEBEL	1	0
HOSTING-FR-COFELy	1	0
IAF	1	0
HOSTING-BE-RAS	1.0938	0.0357
HOSTING-FR-RAS	1.0938	0.035
AWSDCN	1.1069	0.6923
HOSTING-FR-GRDF	1.1905	0.0469
IOT-BE	1.4600	0.2368
HOSTING-BE-SHARED	1.7632	0.6143
HOSTING-FR-SHARED	1.9853	0.4178
Demoset	2.1377	1.3942
AIMv2	2.5573	1.5361
HOSTING-BE-EBL	3.3882	4.0606
HOSTING-BE-ORES	4.3999	2.7795
AdminBE	5.8759	5.2328
AdminFR	6.3938	10.3688

TABLE 7.2: **RNIR** VS **SDI**

base would have a ratio of micro-states to macro-state equalling 1. There are multiple configurations of services that deliver a statistical entropy of 1. We are aware of at least two: one port per service, and the one we discovered with the brown-field algorithm by disentangling the services. The **SDI** is, however, an imperfect representation of the statistical entropy of the rule base. Indeed, we have demonstrated that the brown-field algorithm may result in shadowing rules. An additional operationalization to measure this would be required in order to fully express the statistical entropy of a rule base.

In our experiment, the independent variable is **SDI** and the dependent variable is the relative increase in the number of rules due to application of the treatment (i.e., application of the brown-field artifact). The relative increase in the number of rules (**RINR**) is calculated as the difference between the number of rules (**NR**) after and before application of the artifact, divided by the number of rules before application of the artifact.

$$\mathbf{RINR} = \frac{NR_{after} - NR_{before}}{NR_{before}}$$

The result can be found in Table 7.2 and Figure 7.2. The correlation between the independent and dependent variable is 0.9257.

Construct Validity

According to [52], construct validity is the degree to which an operational definition of a specific concept under observation matches the actual concept. In our case, the

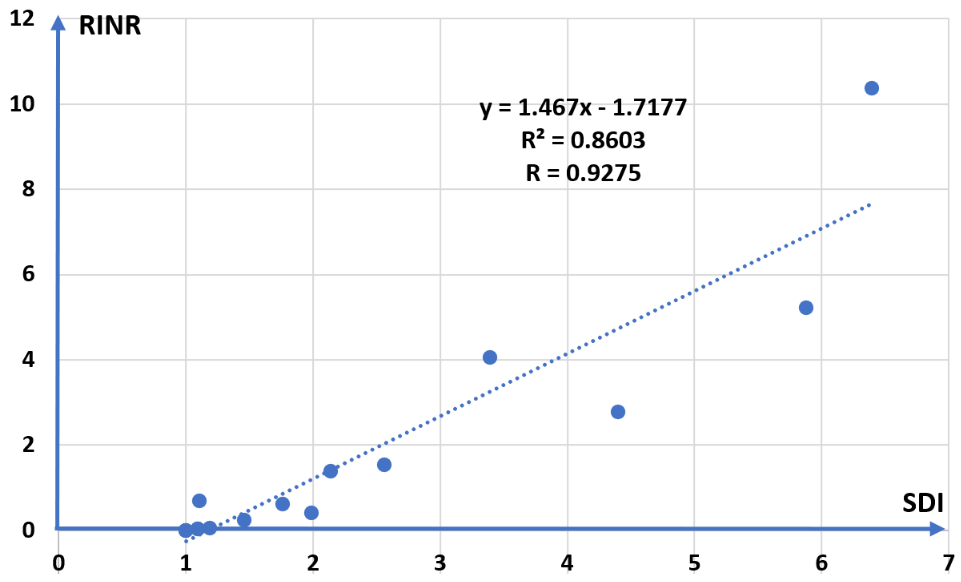


FIGURE 7.2: **RINR** vs **SDI**

concept we attempt to observe is the growth rate of the rule base that is attributable to the application of the artifact, in relation to the initial state of the disjointness of the services. We operationalized these concepts via **SDI** and **RINR**.

According to [52], construct validity faces a number of threats. Those potentially relevant to the current analysis are listed and assessed below.

- **Inadequate explanation of constructs**: we believe that we have sufficiently explained what we sought to achieve. We also maintain that we have adequately described the role of the artifact. While we contend that application of the green-field artifact results in an evolvable rule base, we state that the current version of the brown-field artifact properly addresses a necessary (disjoint service definitions) but insufficient (possible shadowing rules and anthropomorphic service definition naming) condition for an evolvable rule base.

- **Construct confounding**: Construct validity can be threatened by failure to describe all constructs, resulting in incomplete construct inferences. We do not have confounding constructs and we have fully operationalized each of our concepts.

- **Mono-operation bias**: This type of bias may occur when there is only a single operationalization of the construct (i.e., disjointness of all services in our experiment) that both under-represents the construct of interest and measures irrelevant constructs, thus complicating the capacity for inference. Constructs and their operationalizations are calculated based on the directly-manipulated variables within the algorithm. These exactly represent what we intended to measure. The mono-operational bias is thus irrelevant with respect to our experiment.

- **Mono-method bias**: This type of bias may occur when operationalizations are used in only one method of measurement. As part of this dissertation, we have developed a single type of brown-field artifact. That artifact is the method by which the operationalizations occur. This is a potential weakness which may be eliminated by using an alternative meta-heuristic (e.g., a construct base heuristic [48]) that measures the same operationalizations and is able to produce disjoint services with associated rules.

- **Confounding constructs with levels of constructs**: The inferences about the construct that best represent study operations may fail to describe the limited level of the constructs that were actually studied. For the creation of disjoint services, there are no confounding constructs other than the ones that are defined. As such, this threat is not applicable to our experiment.

In [52], additional construct validity threats are discussed, including: treatment-sensitive factorial structure, reactive self-reporting changes, reactivity to the experimental situation, experimenter bias, novelty and disruption effects, compensatory equalization, competence rivalry, resentful demoralization, and treatment diffusion. These dimensions are linked to the units that administer or receive the treatment. In the social sciences, these are humans and/or social systems. In our experiment, however, the units are firewall exports. Human agency is not relevant and so neither are these construct validity threats.

Statistical Conclusion Validity

Could there be reasons for drawing invalid inferences about the existence and size of covariations between variables?

In our case, there is no doubt about the covariance and causality of the variables. The brown-field algorithm actively manipulates the number of rules when it lowers the **SDI**. We could not determine a formula that expresses **RINR** in function of **SDI**. We know that the full-carve-out operations can generate more rules than the intersect carve-out. The value of **SDI** is a function of the number of overlaps (subgroups or intersections), but the distribution between the overlap types is firewall-dependent. To confirm these details, one must actively run the algorithm and count the number of times the different carve-outs are performed.

As such, we are confident in our conclusion that the various potential threats to statistical validity (e.g., low statistical power, violated assumptions of statistical tests, fishing, the error rate problem, unreliability of measurement, extraneous variance between experimental settings, restriction in range, unreliability of treatment implementation, and inaccurate effect size estimation) are not applicable to our experiment.

Internal Validity

Internal validity concerns the extend to which the observed co-variation between A and B actually reflects a causal relationship between A and B. A and B are said to have a causal relationship when: a) cause precedes the effect, and b) cause is positively correlated to the effect such that the effect would fail to occur in the absence of the cause.

In our experiment, we observed the relationship between the cause (i.e., the variation of **SDI**) and the effect (i.e., the variation **RINR**). For a given rule base, lowering the **SDI** means splitting services such that the overall disjointness of the services declines. Splitting services requires splitting of rules. There is thus no question about the causal relationship — cause and effect has in fact been programmed into the algorithm.

Below we review the various potential threats to internal validity and assess whether they may or may not have relevance to our experiment.

- **Ambiguous temporal presence**: This concerns the lack of clarity regarding what is cause and what is effect and if the former precedes the latter. As outlined above, for the present study, the relationship between the two has been fixed inside the algorithm.
- **Selection bias**: This bias may occur when there is a systematic difference in characteristics between the units that could also exert influence over the effect. We assert that the present study does not introduce this bias type. Rather, differences in the initial value of the **SDI** are welcomed as this is the independent variable we are manipulating. The differences in the initial value of the number of rules are immaterial as we measure the relationship between cause and effect as the relative difference between pre- and post-application of the brown-field artifact, irrespective of absolute values.
- **History**: This threat may be introduced within experimental scenarios where events occurring concurrently with treatment cause or contribute to the observed effect. During the execution of the brown-field artifact, there are two reasons why a rule is split. The first is to be found in the pre-processing phase, wherein the rule base is prepared such that one rule contains only one service. The second reason occurs during the splitting of service definitions and associated rule adjustments. The extra rules generated due to pre-processing are not taken into account when measuring the effect. We measure between start and end of the ILS, not between start and end of the brown-field artifact. We thus conclude that we may exclude the history effect.
- **Maturation**: This occurs when the treatment effect is confused with naturally-occurring changes over time. The firewall exports were all taken at a certain point in time (March 2021) and were not subsequently refreshed. Natural evolution due to new rules or policy changes are thus unable to exert influence within the experiment.
- **Regression artifacts**: When units are selected for their extreme scores, they will often have less extreme scores on other variables, and such occurrences may be confused with a treatment effect. The firewall exports used in the present study were all provided by a firewall administrator. We requested and received a set of firewall exports that protect different kinds of data center zones and thus have different filtering strategies. We observed considerable variation in the provided firewalls with respect to initial size and initial value of the **SDI**. We are thus confident that the firewall administrator randomly selected the firewalls.
- **Attrition**: Attrition, or experimental mortality, refers to the fact that fewer subjects complete the treatment relative to the number of initial participants and, as such, not all participant data is represented in the final results. Initially, due to the firewall export format and issues with unexpected characters in the export and export conversion files, not all exports resulted in a successful run of the algorithm. This prompted our introduction of the pre-processing phase

into the brown-field algorithm as an attempt to eliminate those issues. This measure resulted in usage of all provided exports, thus eliminating any concerns of attrition.

- **Testing**: This threat may occur when testing of the treatment can influence the final experiment and the scores. This bias was not present in the current study, as multiple runs of the algorithm utilizing identical data resulted in the same result across runs.
- **Instrumentation**: Changes in measurement instruments over time, even in the absence of treatment, can mimic a treatment effect. Running the brown-field artifact on a Windows or a Linux machine did not have any effect on the end result. Only processing speed could have been influenced (but only marginally).
- **Additive and interactive effects of multiple validity threats**: Interactions of the above-mentioned threats may impact observed outcomes. As we have compensated for all relevant threats, and other threats are not applicable, we conclude the additive effect was not a factor in our experiment.

External Validity

External validity relates to the extent to which a causal relationship holds across variations in units, settings, treatments, and outcomes. In our experiment, units varied (e.g., different firewall functions in the data center) while settings, treatments and outcomes remained constant.

- **Interaction of the causal relationship with units**: The relationship between **SDI** and **RINR** is clear in the scope of our experiment. We do however refrain from making larger claims about this relationship and the size of the relationship, as we used only a limited amount of firewalls housed in Engie data centres. Data about additional firewalls, inside and outside of Engie, could be useful for making statements of a more general nature.
- **Interaction of the causal relationship over treatment variations**: We only created one kind of brown-field artifact. We cannot make inferences as to what other types of artifacts would produce in terms of outcomes. What is clear is that the ILS-based algorithm does allow the disentanglement of the service groups and thus provides a working solution to achieve service group disjointness.
- **Interaction of the causal relationship with outcomes**: The outcomes of the algorithm are the same.
- **Interaction of the causal relationship with settings**: The settings have no impact on the outcome of the algorithm result. A different development and execution environment (e.g., programming language, OS, hardware) can exert an impact on the total execution time, but not on the result.
- **Context-dependent mediation**: there are no mediators in the relationship between **SDI** and **RINR**.

Validity Summary

Based on the preceding, we are able to make the following claim regarding the relationship between **SDI** and **RINR**:

In the context of data delivered by Engie from data center firewalls based in Belgium and Paris, the Iterated Local Search-based brown-field algorithm successfully disentangles the services into disjoint services, and adjusts the rule base accordingly. The success of the disentanglement is measured through SDI - the Services Disjointness Index - which has a value greater than one (1) at start the algorithm and ends with value of one (1) once the algorithm has been run. There is a causal relationship between the value of ***SDI*** *at start of the algorithms and the relative increase in terms of the number of rules, expressed as* ***RINR****, resulting from the application of the algorithm. The data does not currently allow for significant statements to be made regarding the complexity (linear, polynomial, exponential) of the relationship between* ***SDI*** *and* ***RINR****.*

7.4 The Additions to the Knowledge Base

In this section, we restate our contributions to the knowledge base related to the domain of firewalls. To the best of our knowledge, our conclusions are not found within the existing academic or industry literature.

7.4.1 Size of the Problem

The solution/problem space of all possible firewall rules that can be made on a firewall given a set of client and hosts is mind-bogglingly large.

$$fj_{\max} = 2.\left(\sum_{a=1}^{cj}\binom{cj}{a}\right)\cdot\left(\sum_{a=1}^{hj}\binom{hj}{a}\right)\cdot\left(\sum_{k=1}^{2^{17}}\binom{2^{17}}{k}\right) \tag{7.1}$$

where cj = $f_c(\mathbf{N})$ and hj = $f_h(\mathbf{N})$

Executing this formula using relatively small numbers, such as ten clients, five hosts and five services, already results in astronomically-big numbers (see example in Section 2.3.2). The probability is thus small that a firewall administrator would consistently select those rules out of the potential design/solution/problem space which reliably result in an evolvable rule base. As such, conscious restriction of the design space is required.

7.4.2 The Green-Field Artifact

The green-field artifact provides design guidance to create a rule base that will be free of CEs and thus evolvable. The design criteria are hardly a surprise and confirm heuristic knowledge. The demonstration of the artifact makes it clear that any kind of aggregation at service or host level immediately opens the door to CEs.

7.4.3 The Brown-Field Artifact

The ILS-based brown-field artifact is a first component within a larger solution which converts existing non-evolvable rule bases into evolvable rule bases.

7.4.4 Impact of the Brown-Field Artifact on the Size of the Rule Base

As SoC has been meticulously applied, the choice of a fine-grained rule base is unsurprising. The relationship between the level of service disjointness and extra rules has been investigated. Additional runs of the algorithm with firewalls from different companies would provide further insight into the complexity of this relationship (and establish whether it is linear, polynomial or exponential).

7.4.5 Measuring the Evolvability of a Firewall

We were able to operationalize one aspect of the evolvability of a firewall, namely the need for disjoint services. Independent from the meaning and functions of the various service ports within the rule base, the **SDI** is an important indicator for the evolvability of the firewall. If **SDI** is greater than 1, the door is left open to the creation of non-evolvable rules. The **SDI** represents the statistical entropy of the service configurations in a rule base and is a good proxy for the statistical entropy of the rule base.

To measure all aspects of evolvability and statistical entropy in accordance with the green-field artifact, a second index concerning the destinations would need to be developed.

7.4.6 Firewall Scaling

The artifacts produce a fine-grained rule base. A large number of rules in a rule base will have a detrimental impact on performance. But creating an evolvable rule base also provides the answer to this problem, given that only an evolvable rule base will scale horizontally.

7.4.7 Multi-Firewall Design Guidance

Design guidance for one firewall is lacking, and this is true to an even greater extent for multiple firewall configurations. Although not worked out on the same level of detail of single firewall design criteria for evolvability, we do provide alternative ways of thinking about multiple firewall setups.

7.4.8 Contribution to NS Theory

This work started with the question of whether or not NS theory may be used to solve evolvability issues of IT infrastructure components. We took the example of the firewall and have successfully demonstrated that a notoriously non-evolvable IT infrastructure component can be stabilized under change, by meticulously applying SoC to the design of firewall rules.
There are other IT infrastructure systems that would benefit from a similar approach, including Identify and Access Management (IAM), applications of Windows policies, application of AWS policies and IAM roles. These are just a few systems that also have some form of embedded rules embedded with potential to conflict with each other, resulting in unexpected behavior and CEs. NS theory can help investigate the problems of evolvability and provide the means to solve them.

7.5 Artifacts as Enterprise Engineering Instruments

The work presented in the doctoral thesis is not far away from practical usage in an organization. The discipline of Enterprise Engineering (EE) takes that premises that organization and their way of work, are consciously created and organized, and can thus be consciously re-created and re-organized. Recall that evolvable rule bases are actually a necessity for enterprises as they allow continuous change without impacting existing systems and without the erosion of cybersecurity. Positioning our artifacts as Enterprise Engineering instruments seems thus relevant.
EE promotes the usage of grounded methodologies and theories to understand what is going on in an enterprise and engineer/re-engineer the organization accordingly. We position this work in the so called Five Way Framework (see Figure 7.3). The framework aids in discussing and evaluating a methodology and we will briefly position the proposed methodology in this work (create evolvable rule bases), into this framework.

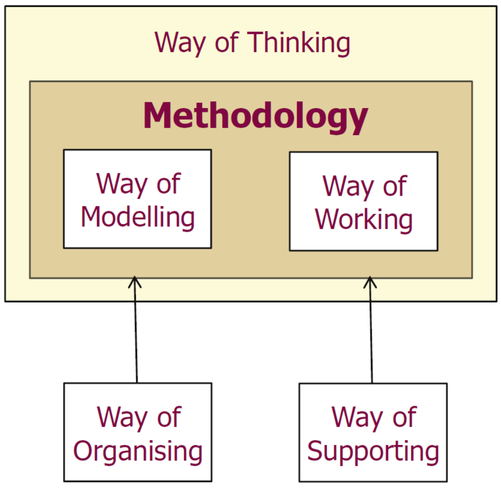


FIGURE 7.3: The Five Way Framework (from [44])

7.5.1 Way of Thinking (WoT)

Evolvability is the main concern of this work and is fully supported by the Normalized Systems theory. The theoretical foundations about system stability and statistical entropy are the paradigms used to study the evolvability issues of the firewall rule base.

7.5.2 Way of Modeling (WoM)

We used ontological modeling techniques and data modeling techniques to demonstrate that current firewalls offer no protection against the creation a non-evolvable rule base. They motivate the usage of additional constraints that need to be enforced by means of artifacts, to obtain evolvable rule bases. Mathematical modeling has been used to express non-evolvability in calculable parameters (**DI**, **OF**, **SDI**) and the proposed brown-field artifact activity manipulates the rule base to improve the values of those parameters towards values corresponding with an evolvable rule base.

7.5.3 Way of Working (WoW)

The green-field artifact dictates a new way of working with regards to the creation of rules. They provide the necessary criteria to follow to obtain evolvable rule bases.

7.5.4 Way of Supporting (WoS)

This work proposes a tool, such as FRANS, to support the application of the different artifacts. Additional work is required to convert and integrate the green- and brown-field artifacts into such tool. The groundwork has been done, it is more a matter of organizing and presenting the functionalities in a user-friendly application.

7.5.5 Way of Organizing (WoO)

Introducing and implementing a tool such a FRANS into an organization is a completely different subject than making a tool that answers to the requirements. FRANS becomes a socio-technical system, that introduces conceptual changes (create evolvable rule bases) and organizational changes (new change process, new ways to express the changes, new ways to implement the changes). It must be accompanies by change management and its success must be measured using IS Research Methodologies. The WoO was not in scope of this work.

Chapter 8

Conclusion and Future Work

This final chapter reiterates the work that has been undertaken, followed by a summary of this work's contribution to the fields of firewall management and NS theory application. We end this monograph by summarizing its limitations and pointing towards future research.

8.1 Conclusion

In Chapter 1 we introduced the context for this research. The concepts and functioning of the TCP/IP firewall and Normalized Systems theory were introduced. We then outlined the research problem: how to create an evolvable rule base and how to convert the firewall from a non-evolvable system to an evolvable system. We provided a rationale for the usage of the Design Science approach as methodology for our research and formulated our research objective accordingly.

In Chapter 2, the problem was discussed and explained. A summary of existing literature that relates to the firewall issues was provided, followed by the introduction of the two root causes of evolvability issues: relationships between rules and rule-order sensitivity. To further explain the complexity of the problem, we used combinatorics (based on a network containing clients, hosts and services) to calculate the size of the firewall solution/problem space.Based on the exchange of essential information required to create a firewall rule, an ontological model of the firewall rule base was proposed. In order to understand how such an ontological model is implemented in commercial firewalls, we reverse-engineered the data model of the rule base of three different firewall vendors, based on firewall exports. Based on the ontological and implementation models, we concluded that firewalls do little to avoid relationships between rules, and offer no protection against the creation of a non-evolvable rule base.

In Chapter 3, we defined the requirements for two artifacts: a green-field artifact aimed at providing design criteria for an evolvable rule base, and a brown-field artifact allowing the conversion of an existing non-evolvable rule base into a rule base that implements one of the green-field artifact's necessary conditions, i.e. disjoint services.

In Chapter 4, the green-field artifact was developed, which employs NS-related SoC, and was demonstrated via validation-by-instantiation.

In Chapter 5, the brown-field artifact (which disentangles services and adjusts the rule base accordingly) was created employing the Iterated Local Search meta-heuristic. We explained the various components of the ILS, and where possible, provided mathematical proof. The brown-field artifact was demonstrated on firewall exports from Engie data centers, and was created using the Iterated Local Search metaheuristic.

Chapter 6 investigated the artifact implications. The green-field artifacts impacts the original ontological model and thus prompt to propose a new one. The green-field artifact is impacted by filtering strategies and we have shown how they can be adjusted accordingly. As networks rarely contain only one firewall, we examined the relation between the green-field artifact and multiple firewalls. We found that applying the green-field artifact will result in a fine-grained rule base containing more rules compared to a rule base containing aggregations at service and destination definitions. A larger rule base potentially impacts firewall performance. However, we showed that an evolvable rule base is the only kind of rule base that can scale horizontally, thus effectively eliminating its own drawback as horizontal scaling can improve performance.

Chapter 7 evaluated and discussed the artifacts by means of considering potential limitations and weaknesses of the artifacts and the experiments conducted with the brown-field artifact. Particular attention was given to the evaluation of the experiments, which in addition to exploring the correct functioning of the brown-field algorithm, also studied the relationship between the level of service entanglement of a rule base, and the rule base growth that can be expected due to applying the brown-field artifact. We meticulously positioned all of our work within the Relevance and Rigor Cycles of the Design Science Framework, as well as the various components comprising the framework.

8.2 Contributions

The issues associated with firewalls have been known for a long time, and although considerable research has been undertaken on the topic in the beginning of the 21st century, common observations of what occurs "in the swamp of practice" suggests that little actionable guidance is available. Industry-security reports describe the need for complexity reduction, as well as for transparency and automation, but these rarely provide actionable guidance for converting an existing rule base into an evolvable rule base and pro-actively maintaining evolvability. How can a ZT policy be implemented if the rules that are created are not fine-grained? How can automation occur in the absence of knowing what is being automated? How can automation occur in a system that becomes increasingly non-evolvable over time? We strongly assert that the issue is not resolved and that the fundamental problems behind firewall evolvability are being ignored by the industry and practitioners .

In this work we re-positioned these issues and contributed to their deeper understanding and resolution. We calculated the size of the solution/problem space of a firewall within a given network and observed that it grows exponentially. In the absence of a meticulously-followed set of design rules, expecting to always select those rules which lead to evolvable rule bases is like hoping to win the lottery on a daily basis. Because humans are not machines, meticulously following a set of rules is better left to machines and algorithms, which would follow the criteria as proposed in the green-field artifact. NS theory provided us with the theoretical background on evolvability and we were able to operationalize indicators for measuring the degree of evolvability of a component that plays an essential role in the evolvability of a rule base: disjoint services. Using meta-heuristics, we developed an artifact that is able to disentangle the services from a rule base and adjust the rule base accordingly, thus bringing the rule base a step closer to evolvability. We studied the relationship between the degree of service disjointness and the growth of the rule base attributable to the application of our brown-field artifact. We observed an increase in rule numbers which might be considered harmful to firewall performance. However, this may be countered by the fact that an evolvable rule base is the only type of rule base with an infinite capacity to horizontally scale and dynamically reorder rules in order to increase performance. As networks rarely contain only one firewall, we also provided guidance for a multi-firewall setups.

This work build on NS theory and domain knowledge about firewalls. It contributes to the value of NS theory as a design theory and as an excellent instrument to study evolvability issues across a multitude of domains.

8.3 Limitations and Future Research

This work is an important yet incomplete step toward the evolvable TCP/IP firewall. The green-field artifact needs to be converted into software that will take a high-level security requirement as input, "expand" it into the required fine-grained rules, and push the rules to a firewall. The brown-field artifact needs to be extended to re-organize the destinations and sources in accordance with the green-field artifact, and requires a solution to naming services such that they are in line with the green-field artifact. These would all be components of the FRANS artifact.
While the requisite groundwork has been established, the remainder needs to be built. We thus regard this not as future research, but rather as future work.

Future research is however indeed required in order to better position the complexity of the relationship between **SDI** and **RINR**. We also posit that this work provides a base-line from which to study the evolvability of other IT infrastructure rule-based systems, such as Identity and Access Management, or various types of policy mechanisms used in cloud-based infrastructures (such as AWS IAM roles and policies). A further line of inquiry would be to investigate how other rule-base systems (such as taxing rules or subsidy attribution) could benefit from a more evolvable design.

Bibliography

[1] P. De Bruyn, *Generalizing Normalized Systems Theory : Towards a Foundational Theory for Enterprise Engineering*. PhD thesis, University of Antwerp, 2016.

[2] P. Huysmans, *On the Feasibility of Normalized Enterprises: Applying Normalized Systems Theory to the High-Level Design of Enterprises*. PhD thesis, University Antwerp, 2011.

[3] D. Van Nuffel, *Towards Designing Modular and Evolvable Business Processes*. PhD thesis, University of Antwerp, 2011.

[4] E. Vanhoof, P. Huysmans, W. Aerts, and J. Verelst, "Evaluating accounting information systems that support multiple gaap reporting using normalized systems theory," in *Enterprise Engineering Working Conference*, pp. 76–90, Springer, 2014.

[5] G. Oort, *Design of Modular Structures for Evolvable and Versatile Document Management Based on Normalized Systems Theory*. PhD thesis, University of Antwerp, 2019.

[6] G. Haerens, "Investigating the applicability of the normalized systems theory on it infrastructure systems," in *Workshop on Enterprise and Organizational Modeling and Simulation*, pp. 123–137, Springer, 2018.

[7] J. D. Day and H. Zimmermann, "The osi reference model," *Proceedings of the IEEE*, vol. 71, pp. 1334–1340, 12 1983.

[8] A. Tanenbaum and D. Wetherall, *Computer Networks - 5th edition*. Pearson, 2013.

[9] W. Stevens, *TCP/IP Illustrated - Volume 1 - the Protocols*. Addison-Wesley Publishing Company, 1994.

[10] Firemon, "Firewall cleanup recommendations," *Firemon whitepaper*, https://www.firemon.com/resources/ 2018.

[11] Firemon, "2017 state of the firewall," *Firemon whitepaper*, https://www.firemon.com/resources/ 2017.

[12] Firemon, "2018 state of the firewall," *Firemon whitepaper*, https://www.firemon.com/resources/ 2018.

[13] Firemon, "2019 state of the firewall," *Firemon whitepaper*, https://www.firemon.com/resources/ 2019.

[14] M. Bennet, "Zero trust security: A cio's guide to defending their business from cyberattacks," *Forrester Research*, June 2017.

[15] H. Shel and A. Spiliotes, "The state of network security: 2017 to 2018," *Forrester Research*, November 2017.

[16] H. Mannaert, J. Verelst, and P. De Bruyn, *Normalized Systems Theory: From Foundations for Evolvable Software Toward a General Theory for Evolvable Design*. Koppa, 2016.

[17] H. Mannaert, J. Verelst, and K. Ven, "The transformation of requirements into software primitives: Studying evolvability based on systems theoretic stability," *Science of Computer Programming*, vol. 76, no. 12, pp. 1210–1222, 2011.

[18] H. Mannaert, J. Verelst, and K. Ven, "Towards evolvable software architectures based on systems theoretic stability," *Software: Practice and Experience*, vol. 42, no. 1, pp. 89–116, 2012.

[19] P. Huysmans, G. Oorts, P. De Bruyn, H. Mannaert, and J. Verelst, "Positioning the normalized systems theory in a design theory framework," in *International Symposium on Business Modeling and Software Design*, pp. 43–63, Springer, 2012.

[20] H. Mannaert, P. De Bruyn, and J. Verelst, "On the interconnection of cross-cutting concerns within hierarchical modular architectures," *IEEE Transactions on Engineering Management*, 2020.

[21] Algosec, "Firewall management - 5 challenges every company must address," *Algosec whitepaper*, https://www.algosec.com/resources/.

[22] H. A. Simon, *The Sciences of the Artificial*. MIT press, 2019.

[23] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *Management Information Systems Quarterly*, vol. 28, no. 1, p. 6, 2008.

[24] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of management information systems*, vol. 24, no. 3, pp. 45–77, 2007.

[25] S. Gregor and A. R. Hevner, "Positioning and presenting design science research for maximum impact," *MIS quarterly*, pp. 337–355, 2013.

[26] P. Johannesson and E. Perjons, *An Introduction to Design Science*. Springer, 2014.

[27] G. Haerens and P. De Bruyn, "Using normalized systems to explore the possibility of creating an evolvable firewall rule base," in *The 11th International Conferences on Pervasive Patterns and Applications (PATTERNS 2019)*, pp. 7–16, May 2019.

[28] G. Haerens and H. Mannaert, "Investigating the creation of an evolvable firewall rule base and guidance for network firewall architecture, using the normalized systems theory," *International Journal on Advances in Security*, vol. 13, no. 1&2, pp. 1–16, 2020.

[29] G. Haerens, "Ontological analysis of the evolvability of the network firewall rule base," in *Proceedings of the 20th CIAO! Doctoral Consortium, and Enterprise Engineering Working Conference Forum 2020*, vol. Vol-2825, 2020.

[30] E. Al-Shaer and H. Hamed, "Design and implementation of firewall policy advisor tools," *DePaul University, CTI, Tech. Rep*, 2002.

[31] P. Eronen and J. Zitting, "An expert system for analyzing firewall rules," in *Proceedings of the 6th Nordic Workshop on Secure IT Systems (NordSec 2001)*, vol. 11, pp. 100–107, 2001.

[32] M. Abedin, S. Nessa, L. Khan, and B. Thuraisingham, "Detection and resolution of anomalies in firewall policy rules," *Proceedings of the IFIP Annual Conference Data and Applications Security and Privacy*, pp. 15–29, October 2006.

[33] Y. Bartal, A. Mayer, K. Nissim, and A. Wool, "Firmato: A novel firewall management toolkit," *ACM Transactions on Computer Systems (TOCS)*, vol. 22, no. 4, pp. 381–420, 2004.

[34] A. Wool, "Architecting the lumeta firewall analyzer.," in *USENIX Security Symposium*, pp. 85–97, 2001.

[35] S. Hinrichs, "Policy-based management: Bridging the gap," in *Proceedings 15th Annual Computer Security Applications Conference (ACSAC'99)*, pp. 209–218, IEEE, 1999.

[36] A. Mayer, A. Wool, and E. Ziskind, "Fang: A firewall analysis engine," in *Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000*, pp. 177–187, IEEE, 2000.

[37] S. Hazelhurst, "Algorithms for analysing firewall and router access lists," *arXiv preprint cs/0008006*, 2000.

[38] E. S. Al-Shaer and H. H. Hamed, "Discovery of policy anomalies in distributed firewalls," in *IEEE Infocom 2004*, vol. 4, pp. 2605–2616, IEEE, 2004.

[39] H. Hamed and E. Al-Shaer, "Taxonomy of conflicts in network security policies," *IEEE Communications Magazine*, vol. 44, pp. 134–141, March 2006.

[40] 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 (JSAC)*, vol. 23, pp. 2069–2084, October 2005.

[41] A. Hari, S. Suri, and G. Parulkar, "Detecting and resolving packet filter conflicts," in *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3, pp. 1203–1212, IEEE, 2000.

[42] D. Monahan, "Research summary: Network security policy management tools –tying policies to process, visibility, connectivity, and migration," *ENTERPRISE MANAGEMENT ASSOCIATES® (EMA) Research Report*, https://web.tufin.com/network-security-policy-managementtools-ema-research 2018.

[43] C. Cunningham and J. Pollard, "The eight business and security benefits of zero trust," *Forrester Research*, November 2017.

[44] J. L. Dietz and H. B. Mulder, *Enterprise Ontology: A Human-Centric Approach to Understanding the Essence of Organisation.* Springer Nature, 2020.

[45] M. Suchánek and R. Pergl, "Evolvability evaluation of conceputal-level inheritance implemtation patterns," in *The 11th International Conferences on Pervasive Patterns and Applications (EMPAT)*, pp. 1–6, 2019.

[46] G. Haerens, "Using normalized systems to explore the possibility of creating an evolvable firewall rule base," in *The 13th International Conferences on Pervasive Patterns and Applications (PATTERNS 2021)*, pp. 1–10, April 2021.

[47] G. Haerens and H. Mannaert, "Improving firewall evolvability with an iterated local search algorithm," *International Journal on Advances in Security*, vol. 14, no. 1&2, 2021, to be published.

[48] P. M. P. M. Rafael and M. G. C. Resende, *Handbook of Heuristics*. Springer, 2018.

[49] Z. Michalewicz and D. B. Fogel, *How to Solve it: Modern Heuristics*. Springer Science & Business Media, 2013.

[50] E.-G. Talbi, *Metaheuristics: from Design to Implementation*, vol. 74. John Wiley & Sons, 2009.

[51] M. Yoon, S. Chen, and Z. Zhang, "Minimizing the maximum firewall rule set in a network with multiple firewalls," *IEEE Transactions on Computers*, vol. 59, no. 2, pp. 218–230, 2009.

[52] T. D. Cook, D. T. Campbell, and W. Shadish, *Experimental and Quasi-Experimental Designs for Generalized Causal Inference*. Houghton Mifflin Boston, MA, 2002.