POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Towards the automatic refinement of high-level security policies in computer networks

Supervisors:

Prof. Cataldo Basile

Prof. Antonio Lioy

Candidate:

Mattia Bencivenga

Academic Year 2021/2022
Torino

Abstract

In the academic world, there are countless examples of policy-based security management work, but given the ever-increasing difficulty of maintaining and managing a secure network, this topic remains very current. Policy-based management is an approach that allows simpler and more effective management of distributed networks and systems by deploying a set of policies to control network behaviour. A good policy refinement job needs to be carried out to ensure that the policy-based management approach can be truly effective. Policy refinement is the process of transforming a high-level abstract policy specification into low-level, concrete policies that can be enforced on the managed system. This part is certainly the most critical and complex to perform, as it must consider numerous variables.

First of all, there is the human component that could lead to error, and this is because the high-level policies are defined, according to the needs, by expert personnel or by simple users. Once this aspect has been overcome, there are the intrinsic difficulties of the task to be solved. In fact, the policy refinement work is characterized by various phases, each of which has its particular difficulties.

The first phase is called requirements identification and consists in identifying the capabilities necessary to enforce a high-level user policy. This operation is undoubtedly the most complex to carry out. It consists of processing the policy to determine all the security controls necessary to apply the policy and the identification of the PSAs that can enforce the policy. The second phase is called non-enforceability analysis and consists in reporting the inability to apply the defined policy. The last phase is the actual generation of the policy using an abstract vendor-independent and device-independent language.

The goal in policy refinement works is to emulate the behaviour of good network administrators. To do this, we have chosen to use an expert system called CLIPS, a computer program that simulates the abilities of a human expert to make decisions. Rather than using traditional procedural code, expert systems are supposed to handle complicated issues by using forward reasoning through bodies of knowledge, which are mostly expressed as if-then rules.

Thanks to CLIPS, it is possible to carry out the enrichment, i.e., the extrapolation of information from the high-level policy that allows the acquisition of useful knowledge for generating the medium-level policy. In addition to the information extracted thanks to CLIPS, information about the network topology and general prior knowledge is required. That is information that a network administrator may have, which allows better identification of the necessary capabilities. This information is stored in ad-hoc databases.

The developed tool successfully refines filtering and protection policies. It generates medium-level policies that are correct and follow the most up-to-date best practices. Furthermore, the proposed solution has been integrated into a complete framework that has allowed the validation of the refinement of high-level policies up to valid low-level configurations for the end devices.

Acknowledgements

First and foremost, I am extremely grateful to my esteemed supervisor, Prof. Cataldo Basile for his invaluable advice, continuous support and supervision during my master's degree studies. My gratitude extends to the Department of Control and Computer Engineering and to all the Professors I have had the pleasure of meeting during my studies at the Politecnico di Torino.

I would like to thank my family for the support of all these years, for always being close to me and for supporting me in all my choices. Finally, I would like to thank all my friends, fellow students and all those who have crossed their life with mine leaving me something good. Thank you for being my accomplices, each in his own way, in this intense and exciting journey, for better or for worse.

"So, I guess we are who we are for a lot of reasons. And maybe we'll never know most of them. But even if we don't have the power to choose where we come from, we can still choose where we go from there. We can still do things. And we can try to feel okay about them."
Stephen Chbosky, The Perks of Being a Wallflower

Table of Contents

List of Figures x

Acronyms xii

1 Introduction 1

2 Background 6

2.1 UML 6

2.2 Class diagram 7

2.3 XML 8

2.4 XML Schema Definition 9

2.5 Expert System 10

2.6 CLIPS 12

2.6.1 CLIPS Python binding 13

2.7 Flask web server 14

2.8 NSF-Catalogue 15

3 State of the art 17

3.1 Network Function Virtualization 17

3.2 Software-Defined Network 18

3.3 Network Security Function 20

3.4 Policy-based management 21

3.5 Interface to Network Security Functions I2NSF 24

3.5.1 I2NSF Problem definition 25

3.5.2 I2NSF Use cases 27

3.5.3 I2NSF Framework 29

3.5.4 I2NSF Flow security policy structure 31

3.5.5 I2NSF Capability information and data model 32

4 System design 34

4.1 Problem definition 34

4.2 Use cases 35

4.3 Main goal 36

4.4 Soution design 36

4.5 System details 37

4.5.1 Description of files structure 37

5 System implementation 43

5.1 Refinement tool 43

5.1.1 Company database 44

5.1.2 Info database 47
5.1.3 CLIPS rules 47
5.1.4 Network topology generation 49
5.1.5 Device configuration selection 50
5.1.6 Output generation 51
5.2 Converter tool 51
5.3 Orchestrator 52
5.4 Workflow implementation 53

6 System validation 56
6.1 Use cases: IpTables and XFRM 56
6.1.1 Network topology 56
6.1.2 High-level policy definition 58
6.1.3 Refinement output analysis 60
6.1.4 Converter output analysis 64

7 Conclusion and future developments 69

A User manual 71
A.1 Start the NSF-Catalogue 71
A.2 Start the Orchestrator 72
A.3 Upload high-level security policies 72
A.4 Upload the company database 72

A.5 Refinement process 73
A.5.1 Start refinement process with GUI 73
A.5.2 Start refinement process without GUI 74
A.6 Converter process 74
A.6.1 Download each RuleInstance file individually 74
A.6.2 Download all RuleInstance files 75

B Developer manual 76
B.1 Refinement tool 76
B.1.1 Refinement with GUI 77
B.1.2 Refinement without GUI 81
B.2 Converter 81
B.3 Orchestrator 82
B.4 Implement new, or modify existing, functionality 83
B.4.1 Support for new high-level policies 83
B.4.2 Change Refinement default behaviors 87

Bibliography 92

List of Figures

3.1 The IETF/DMTF policy framework 22

3.2 Generic policy management tool 23

3.3 Interaction between Entities 27

3.4 I2NSF framework 30

5.1 Pyvis network topology graph 50

5.2 Framework workflow 54

6.1 Network topology 57

B.1 Refinement process dependencies 77

Acronyms

ACL
Access Control List

AI
Artificial Intelligence

API
Application Programming Interface

BSD
Berkeley Software Distribution

CLIPS
C Language Integrated Production System

COOL
Complete Object Oriented Language

CPU
Central Processing Unit

DMTF
Desktop Management Task Force

DNS
Domain Name System

DTD

Document Type Definition

GUI

Graphical User Interface

HTML

HyperText Markup Language

I/O

Input/Output

I2NSF

Interface to Network Security Functions

IBM

International Business Machines Corporation

ICT

Information and Communications Technologies

IETF

Internet Engineering Task Force

IP

Internet Protocol

ISO

International Organization for Standardization

JSON

JavaScript Object Notation

LISP

List Processing

NAIL

NASA's AI Language

NAT

Network Address Translation

NFV

Network Function Virtualisation

NSF

Network Security Function

NSP

Network Service Provider

P2P

Peer to Peer

PDP

Policy Decision Point

PEP

Policy Enforcement Point

QoS

Quality of Service

REST

Representational State Transfer

SDN

Software Distribution Network

SLA

Service Level Agreement

UML

Unified Modeling Language

URL

Universal Resource Locator

VLAN

Virtual Local Area Network

VM

Virtual Machine

VoIP

Voice over Internet Protocol

VPN

Virtual Private Network

W3C

World Wide Web Consortium

WSGI

Web Server Gateway Interface

XML

Extensible Markup Language

XSD

XML Schema Definition

Chapter 1

Introduction

Virtualization is a technology that allows for more efficient utilization of physical computer hardware resources, it allows you to exploit all the capabilities of a physical machine by distributing the functionality among multiple users or environments. Virtualization uses hypervisor software, a term coined in 1966, to create an abstraction layer over computer hardware, in this way the single hardware elements of a single computer can be divided into multiple virtual computers, commonly called virtual machines (VMs).

The concept of virtualization is generally believed to have its origins in the late 1960s, when IBM invested a lot of time and effort in developing robust time-sharing solutions. The time-sharing concept was born from the idea that if it was possible to allow more than one user to use the computer, efficiency will increase. Originally, the concept was implemented in a simple way: only while performing I/O operations the CPU switched between tasks; while one user was entering data, the computer processed tasks of other users. It will fill pauses and minimize idle time.

With the advance in technology the hardware grew in power, and the tasks created by several users were insufficient to occupy all of its capacity. The CPU was then trained to swap tasks more often. When the CPU worked with a task, it was given a "quantum" of time. If one quant wasn't enough to do the task, the processor moved on to the next one before returning to the previous one at the following quant. The shifts happened so swiftly that

each user would believe he was the only one who used the machine.

In the 2000s the need for virtualization capabilities grew in tandem with the rise of the desktop computer industry, and developers quickly began pushing the technology's boundaries. Virtualization sees its peak of expansion in this period, and it is thanks to this that nowadays virtualization techniques are used in data center to abstract physical hardware and create large, aggregated pools of logical resources, such as CPUs, memory, disks, file storage, applications, and networking, which are then offered to users or customers in the form of agile, scalable, consolidated virtual machines. Even though technology and application cases have evolved, virtualization's primary purpose remains the same: to allow a computer environment to operate numerous independent systems at the same time, this is because the goal to reduce capital and operational costs within the data center is of primary importance.

The numerous advantages of virtualization have greatly increased the demand from the market and this has led to great innovations in cloud computing and Network Function Virtualisation (NFV) technologies. This has made possible the incredible diffusion of this service delivery model, which is particularly useful for providing users with network security services.

Security solutions that are software-based and built to work in a virtualized IT environment are referred to as virtualized security. This differs from traditional hardware-based network security that is, on the other hand, static and operates on devices like traditional firewalls, routers, and switches.

Virtualized security is more flexible and dynamic than hardware-based security. It may be deployed anywhere on the network and is usually cloud-based, rather than being bound to a specific device. This is critical in virtualized networks, where operators dynamically generate workloads and applications; virtualized security allows security services and functions to move around with those dynamically formed workloads.

Furthermore, this cloud-based security service paradigm makes it easier to deploy a variety of security features produced by a variety of vendors. This is well suited to fulfil the rising demands of corporate network systems to incorporate security functions in order to produce more secure systems. Since there is no industry standard for Network Security Functions (NSF)

interfaces, various manufacturers' NSF offer distinct configuration and administration interfaces. This variability adds to the complexity the problem of maintaining numerous suppliers' NSFs, resulting in higher management expenses. As a result, standardization is critical for the proper implementation of NSF offered by multiple manufacturers. To fulfil these demands, certain standardization initiatives, such as the Internet Engineering Task Force Interface to Network Security Functions (IETF I2NSF) working group, have just begun a development process.

Despite the efforts of the I2NSF working group, there are still some issues to be resolved. The thesis fits in the policy-based management area, which allows the administration process to be improved and largely automated. Policies, in fact, are technology-agnostic rules meant to improve the hard-coded functioning of managed devices by incorporating interpreted logic that can be updated dynamically without affecting the underlying implementation. This improves the capability of a device or a network to accept a new set of instructions, that may alter the device, without interfering with the operation of either the managed system or the management system itself. In particular, the work will focus on automated policy-refinement, which consists in the translation of high-level policies into a language of medium abstraction suitable, subsequently, for the translation into low-level policies for the configuration of physical devices. Automated security policy-refinement is a research area that is currently being investigated. In fact, the researcher's attention has recently been drawn to it.

Despite being introduced in several application areas through multiple research initiatives, several standardisation attempts, and significant industry interest, there is still no universal implementation standard for applying policy-based management of large-scale systems. One reason for its reluctance to be used is the difficulty in analysing strategies that ensure configuration stability. Policies may clash, resulting in unpredictable outcomes; also, the number of policies required to regulate medium to large-scale systems may be in the thousands. Furthermore, there is also the human component to consider, which might lead to inaccuracy, because high-level policies are developed, depending on the requirements, by professional employees or by simple users. Once this hurdle has been passed, there are the intrinsic difficulties of the task to be solved. The ideal scenario is to have an automated tool that can select, compare, and configure NSFs based on user requirements.

The key benefit is that user security requirements would be articulated using high-level language that is similar to normal language.

As already mentioned, policy-refinement is a very complex task, which must consider various variables and it is composed of three main phases. The first stage, known as requirements identification, involves determining the capabilities required to enforce a high-level user policy. This is without a doubt the most difficult procedure to carry out. It comprises of analysing the policy to determine all of the security measures required to implement the policy and identifying the devices that can enforce the policy. The second stage is known as non-enforceability analysis, and it consists in revealing the inability to implement the established policy. The final step is to generate the policy using an abstract vendor-independent and device-independent language.

The purpose of policy refinement is to mimic the behaviour of skilled network administrators. To do this, we have chosen to use CLIPS, an expert system that is a computer software that simulates the decision-making abilities of a human expert. Expert systems, rather than standard procedural code, are designed to tackle complex challenges by applying forward reasoning through banks of knowledge, which are mostly expressed as if-then rules. CLIPS enables enrichment, i.e. the extrapolation of information from high-level policy that permits the acquisition of important knowledge for the generation of medium-level policy. In addition to the information retrieved by CLIPS, network topology information and general prior knowledge are required. That is knowledge that a network administrator may have, allowing for a more accurate identification of the required capabilities. This data is kept in ad hoc databases.

In particular, the work aims to create a framework that is able to receive high-level policies, contained in an XML file, and process these policies to extract information as well as would have done a security expert, in this way the framework could be able to associate the necessary security capabilities related to each single high-level policy. Subsequently, the framework must be able to generate a medium-level abstraction language, also contained in an XML file, as defined by the thesis work of Avallone [1], and subsequently continued by the thesis work of Cirella [2], to allow the tools they developed the final translation into low-level rules for the actual configuration of the individual physical devices on the network.

The work will be validated by testing filtering and protection policies, it will be noted how the tool is able to generate mid-level policies that are correct and follow the most up-to-date best practices. Furthermore, the proposed solution was integrated into the general framework that allowed the validation of the refinement starting from the high-level policies up to the low-level configuration of the individual network devices.

Chapter 2

Background

In this chapter the fundamental languages and basic concepts will be presented.

2.1 UML

Unified Modelling Language (UML) is a standardized modelling language that consists of an integrated set of diagrams that was created to assist system and software developers in specifying, visualizing, designing, and documenting software system artifacts, as well as business modelling and other non-software systems. UML was approved as a standard by the Object Management Group (OMG) in 1997. Since then, OMG has been in charge of it. In 2005, the International Organization for Standardization (ISO) accepted UML as a standard. UML has been updated throughout time and is examined on a regular basis [3].

The UML is a collection of best engineering practices for modelling big and complex systems that have been demonstrated to work. The UML is a critical component of object-oriented software development and the software development process. To express the design of software projects, the UML primarily use graphical notations. The UML aids project teams in communicating, exploring new designs, and validating the software's

architectural design.

UML is linked with object-oriented design and analysis. UML makes the use of elements and forms associations between them to form diagrams. Diagrams in UML can be broadly classified as [4]:

1. Structural Diagrams: capture static aspects or structure of a system [4];

2. Behaviour Diagrams: capture dynamic aspects or behaviour of the system [4].

2.2 Class diagram

The class diagram is the most extensively used UML diagram. It serves as the foundation for all object-oriented software systems. Class diagrams are used to describe the static structure of a system by displaying the classes, methods, and properties of the system. Class diagrams also assist us in determining the relationships between various classes or objects. A class in most modelling software is made up of three pieces. Name at the top, attributes in the middle and operations or methods at the bottom. Classes are put together to build class diagrams in a complex system with numerous related classes. Different sorts of arrows represent different types of links between classes. In particular, relationships in class diagrams include different types of logical connections. The following are such types of logical connections that are possible in UML:

- Association: it represents a family of links; in fact, it is a broad term that encompasses just about any logical connection or relationship between classes.

- Aggregation: It is a special form of association. It portrays a part-of relationship. It's a binary connection, thus it can't have more than two classes in it. It is also called as "has-a relationship". It indicates the orientation of an object that is enclosed within another object. A child can exist independently of the parent in aggregation.

- Composition: This relationship is quite similar to the aggregation relationship, with the exception that its primary goal is to emphasize the contained class's reliance on the container class's life cycle. That is, when the container class is destroyed, the enclosed class is destroyed as well. Furthermore, the enclosed class can be included at most in a container and only the container object can create and destroy its parts.

- Generalization: It's a form of connection in which one associated class becomes a child of another by inheriting the parent class's features. To put it another way, the kid class is a subset of the parent class. It connects a generic element to a more specific one.

2.3 XML

XML (Extensible Markup Language) is a markup language that is similar to HTML but does not have any predefined tags. Instead, you create your own tags that are tailored to your individual requirements. This is an effective method of storing data in a format that can be searched, saved, and shared. Most crucially, because the core structure of XML is standardized, the receiver can still understand the data whenever you share or transmit XML across systems or platforms, whether locally or over the internet, thanks to the standardized XML syntax. The main reasons for the importance of the development of XML is:

1. XML languages can be used to develop standards for the interchange of data.

2. There are many XML parsers available which will check an XML document for well-formedness and validity.

3. The structure of XML lends itself easily to a tree representation – showing the derivation tree of the document.

XML allows you to define and control the meaning of the elements contained in a text document, it has some advantages and disadvantages. Let's see first the advantages:

- XML is platform independent and programming language independent, therefore it can be used on any system and supports the technology change when that happens. This also simplifies data sharing between various systems, XML doesn't require any conversion.
- Unicode is supported by XML. Unicode is an international encoding standard for use with many languages and scripts, in which each letter, number, or symbol is given a unique numeric value that may be used across platforms and programs. XML may transport any information written in any human language thanks to this property.
- The data stored and transported using XML can be changed at any point of time without affecting the data presentation.
- XML allows validation using DTD and Schema. This validation ensures that the XML document is free from any syntax error.

XML also have some disadvantages:

- XML syntax is verbose and redundant compared to other text-based data transmission formats such as JSON.
- The redundancy in syntax of XML and the fact that it is very verbose, cause higher storage and transportation cost when the volume of data is large.
- XML document is less readable compared to other text-based data transmission formats such as JSON.
- XML doesn't support array.

2.4 XML Schema Definition

Because the structure of an XML document may vary greatly, there are several methods to communicate the same information. Although this may not be a problem for individuals, it may inhibit appropriate communication between computer systems. As a result, it's a good idea to construct a set of

document rules that specify which elements and attributes are permitted. In this way, both communication parties are aware of what to expect and may adjust their application logic accordingly.

The first modelling language for expressing the tree structure of XML documents was Document Type Definition (DTD). XML may generate a document markup template using DTDs, allowing the placement of elements and their attributes to be examined and validated. Since 2009, more appropriate ways for creating schema definitions have been developed.

XML Schema Definition (XSD) is the best known and most widespread, it is a World Wide Web Consortium (W3C) recommendation that specifies how to formally describe the elements in an Extensible Markup Language (XML) document [5]. This description may be used to ensure that each piece of content in a document matches the description of the element it will be placed in, in this way the positioning of elements and their attributes can be checked and validated.

An XML schema's goal is to define and describe a class of XML documents by utilizing schema components to limit and explain the meaning, usage, and relationships of its constituent parts: data types, elements, attributes, and values. An XML schema defines the basic structure of an XML document as well as the restrictions that apply to the entities included inside it.

2.5 Expert System

Artificial intelligence (AI) is the capacity of a computer or a computer-controlled robot to accomplish activities that would normally be performed by intelligent individuals. The phrase is widely used to refer to a project aimed at creating systems with human-like cognitive abilities, such as the capacity to reason, discern meaning, generalize, and learn from prior experiences.

In the AI field, an expert system is a computer system that simulates the decision-making abilities of a human expert. Expert systems, rather than using traditional procedural code, are supposed to handle complicated issues by reasoning through bodies of knowledge, which are mostly expressed as if–then rules. The first expert systems were developed in the 1970s, and they

became increasingly popular in the 1980s. Expert systems were one of the first kinds of artificial intelligence software to be fully successful and they were the first commercial system to use a knowledge-based architecture on which they are based.

The knowledge base and the inference engine are the two subsystems, and main components, of an expert system.

The knowledge base is a collection of facts and rules about a certain domain. Facts were mostly stated as flat assertions about variables in early expert systems. Rules, on the other hand, are composed of an if portion and a then portion. The first one is a series of patterns which specify the facts (or data) which cause the rules to be applicable. The latter portion is the set of actions to be executed when the rules is applicable. In later expert systems developed with commercial shells, the knowledge base took on more structure and used concepts from object-oriented programming. Classes and subclasses were used to represent the domain, and instances and assertions were substituted with values of object instances. The rules functioned by querying and asserting object values.

The inference engine is an automated reasoning system, its job is to pull relevant data from the knowledge base, analyse it, and come up with a solution that is relevant to the user's problem. To infer new facts, the inference engine uses rules from its knowledge base and applies them to known facts. An inference engine can operate in one of two modes: forward chaining[1] or backward chaining[2]. The distinct methodologies are determined by whether the rule's antecedent (left hand side) or consequent (right hand side) drives the inference engine. In particular, in forward chaining an antecedent fire and asserts the consequent. While backward chaining is a little more complicated, it requires the system looking at alternative outcomes and working backward to check if they are correct.

[1]Forward chaining (or forward reasoning) is one of the two basic techniques of reasoning when implementing an inference engine and is logically defined as the repeated application of *modus ponens*. For expert systems, business rule systems, and production rule systems, forward chaining is a prominent implementation method.

[2]Backward chaining begins with a list of objectives (or a hypothesis) and proceeds backwards from the consequent to the antecedent to check whether any data supports any of these consequents.

In the corporate sector, expert systems have been consistently employed to acquire tactical advantages and foresee market conditions. In this era of globalization, where every business choice is important to success, expert system support is unquestionably necessary and extremely trustworthy for a corporation to flourish.

Expert system has some serious advantages. It can deliver consistent responses for repetitive decisions, processes, and tasks. The findings drawn will remain the same as long as the system's rule base remains the same.It has no human constraints and can work constantly around the clock. Users will be able to utilize it regularly to find solutions. Expert knowledge is a priceless advantage for any business. Unlike humans, who have a hard time adapting to new situations, expert systems are highly adaptable and can quickly satisfy new needs. It may also capture fresh information from an expert and utilize it to solve new issues using inference rules.

Expert system also has some limitations. It lacks the common sense required in certain decision-making because all judgments are based on the system's inference rules. It also can't respond as creatively, and innovatively as human specialists would in odd situations. Domain specialists will not always be able to communicate their logic and reasoning, that will be required during the knowledge engineering process. As a result, codifying our knowledge is a challenging job that might take big effort and a lot of time.

2.6 CLIPS

CLIPS is a free and open-source software tool for creating expert systems. The name CLIPS is an abbreviation for "C Language Integrated Production System". CLIPS was originally created at NASA-Johnson Space Center in 1985 and continued until the mid-1990s, when the development group's duties shifted away from expert system technology. The original name of the project was "NASA's AI Language" (NAIL) [6].

As of 2005, CLIPS was perhaps the most extensively utilized expert system tool. CLIPS is written in C for portability and speed, and its syntax

is similar to that of the Lisp programming language[3]. At the time of its original development, CLIPS was one of the few tools that was written in C and capable of running on a wide variety of conventional platforms. In fact, CLIPS can be ported to any system which has an ANSI compliant C compiler including personal computers, workstations, minicomputers, Mainframes, and supercomputers [7]. CLIPS includes an object-oriented programming language for creating expert systems. COOL mixes procedural, object-oriented, and logical (theorem-proving) programming paradigms.

CLIPS, like other expert system languages, make use of rules and facts. As said before, there are two main modes in which an inference engine can operate: forward chaining and backward chaining. CLIPS's inference engine is based on the Rete algorithm [8] which is an extremely efficient algorithm for pattern matching, it uses forward chaining; various facts can make a rule applicable and when a rule becomes applicable is then fired. This approach was the first, and originally the only, programming paradigm provided by CLIPS, and it is called rule-based programming.

In this methodology, rules are used to represent heuristics, which specify a set of actions to be performed for a given situation. The inference engine selects applicable rules and executes its actions, the process continues until no applicable rules remain.

2.6.1 CLIPS Python binding

Clipspy is a Python CFFI (C Foreign Function Interface) bindings for the 'C' Language Integrated Production System (CLIPS), it is like a "pythonic" thin layer built on top of the CLIPS native C APIs.

The choice to use clipspy is due to various reasons, in particular it allowed me to work with a programming language that I think is easier to understand

[3]Lisp (formerly LISP) is a computer language family with a long record and a distinct, totally parenthesized prefix notation. Although Lisp began as a practical mathematical notation for software applications, it rapidly became the preferred programming language for artificial intelligence (AI) research. LISP is an abbreviation for "LISt Processor", linked lists are one of the most important data structures in Lisp, and lists are used extensively in the language's source code.

and provides numerous advantages such as:

- A simple syntax, both to read and to write;
- Easier implementation of data structures thanks to built-in functions;
- Vast Libraries Support;
- Simple error debugging.

2.7 Flask web server

Flask is a micro-framework in Python which is used for web development. Given its small size and its excellent capabilities, Flask is a great tool for creating your own websites in a dynamic and interactive way.

The term "micro" does not imply that your whole web application must fit inside a single Python file (though it most definitely can), nor does it imply that Flask is limited in capabilities. The term "micro-framework" refers to Flask's goal of keeping the core basic but adaptable. In this way it is possible to use Flask without constraints, such as on the selection of a database but these additions are possible through extensions that give the user full freedom.

Flask is part of the Pallets Projects (previously Pocoo) and is based on numerous others:

- *Werkzeug*: it is a full-featured WSGI web application library. It began as a basic collection of several utilities for WSGI applications and has now evolved into one of the most sophisticated WSGI utility libraries. Werkzeug is capable of creating software objects for request, response, and utility functions. It supports Python 2.7 and 3.5 and beyond and may be used to construct a customizable software framework on top of it. There are no dependencies enforced by Werkzeug. It is the developer's responsibility to select a template engine, database adapter, and even how to handle requests.

- *Jinja*: it is a templating engine that is quick, expressive, and extensible. The template has special placeholders that allow you to write code that is comparable to Python syntax. It manages templates in a sandbox, similar to the Django web framework.
- *MarkupSafe*: it defines a text object that escapes characters so that it can be used safely in HTML and XML. Ones with particular meanings are replaced to seem like the original characters. This prevents injection attacks and allows untrusted user input to be presented on a website safely.
- *ItsDangerous*: it is a BSD-licensed safe data serialisation package for the Python programming language. It is used to store a Flask application's session in a cookie while preventing users from tampering with the session contents.

2.8 NSF-Catalogue

The NSF-Catalogue was developed by Cirella [2] during his thesis work, the BaseX REST API Web Server feature provides a BaseX-based XML Database. The BaseX web server lacks a graphical user interface and must be accessed via `curl` or the Python `requests` library.

The web server is used to give the user functionality to check in advance which NSFs are available, which NSFs share *Security Capabilities*, and which NSFs can implement which *Rule Instances*. The NSF Catalogue database achieves the purpose of allowing users to compare different NSFs and check NSF inner details in terms of which *Security Capabilities* they provide. The NSF-Catalogue supports various queries [2]:

- *Two NSFs comparison*: query that receives two NSF names and checks if their Security Capability Sets are: contained each other, equivalent, or their intersection is empty.
- *Find NSF based on rules*: query that receives the path of a Rule Instance file and returns the NSF that is able to implement the policies stated within the above file.

- *Find NSFs based on Security Capabilities*: query that receives a set of Security Capabilities and returns which NSFs provide those items.
- *Find all the Security Capabilities*: query that returns the list of all the available Security Capabilities arranged by type.

In the context of the thesis in question, as we will see later, will be used only the query *Find NSF based on Security Capabilities* that, given a set of security capabilities, returns the NSFs that support them.

Chapter 3

State of the art

In this chapter some works and technologies that have been fundamental for the thesis in question will be presented.

3.1 Network Function Virtualization

In recent years virtualization has gained popularity in many different IT areas such as: cloud computing, server consolidation and information security. In fact, nowadays most IT environments are over 80% virtualized.

Network Function Virtualization (NFV) is the replacement of network appliance hardware with virtual machines, this is a way to virtualize network services and allow service providers to run their network on standard servers instead of proprietary ones. It exploits the benefits deriving from the virtualization, changing the way in which the network is conceived. NFV defines a virtualized infrastructure for network functions so, in this way, is possible to reach the main objective: decoupling software from hardware. These functions are called virtual network functions (VNFs), thanks to NFV there is no need to have dedicated hardware for each network service (such as routers, firewalls, and load balancers) and both scalability and agility are improved because service providers can deliver new, or change existing, network services and applications on-demand, this is possible because the

network runs on virtual machines that are easily provisioned and managed. This approach enables the instantiation of as many network function instances as possible on whichever physical device is available at the time, resulting in high service agility without the need to purchase and setup additional physical equipment. Software and hardware development may be done concurrently and separately. This also allows for distinct tasks to be assigned to different hardware and software components. In this way, operational costs, response time and administration tasks are reduced. The fact that multiple functions can be executed on a single server implies that less physical hardware is required, which allows for resource consolidation that results in physical space, power, and overall cost reductions.

3.2 Software-Defined Network

Emerging mega-trends in Information and Communication Technologies (ICT) are creating new difficulties to the future Internet, that requires pervasive accessibility, high bandwidth, and dynamic management. Recently, Software-Defined Networking (SDN) has been confirmed as one of the most potential future Internet solutions.

The Open Networking Foundation (ONF) [9] is a non-profit organization committed to the development, standardization, and commercialization of software-defined networking. The ONF has offered the clearest and widely accepted definition of SDN, which is as follows:

Software-Defined Networking (SDN) is a new network architecture that decouples network control from forwarding and is directly programmable [10].

According to this definition, SDN is identified by two features: separating the control plane from the data plane and offering programmability[1] for network application development. As a result, SDN is ready to deliver more efficient configuration, improved performance, and more flexibility to support

[1]The capacity of hardware and software to change; to receive a new set of instructions that alter its behaviour is referred to as programmability. It relates to program logic (business rules) in general, but it also applies to creating the user interface, which includes menus, buttons, and dialogues.

novel network designs.

Conventional methods focused on the manual setup of proprietary devices, on the other hand, are time-consuming and error-prone, and they do not fully leverage the capabilities of physical network infrastructure. Traditional network architecture is static, in fact, protocols are usually defined individually, each solving a particular problem and without the benefits of a basic abstraction. This creates complexity, which is one of the main limitations of today's networks. For example, to add or move devices, IT administrators have to access multiple switches, routers, firewalls, and web authentication portals. They also have to update ACLs, VLANs, Quality of Service (QoS), and other protocol-based mechanisms with a device-level management tool. In addition, you need to consider your network topology, vendor switch model, and software version. Due to this complexity, IT administrators seek to minimize the risk of service interruptions by interpreting today's networks in a relatively static way.

The static structure of networks contrasts sharply with the dynamic nature of today's server environment, where server virtualization has significantly increased the number of hosts requiring network access and fundamentally changed assumptions about host physical location. Before virtualization, programs were housed on a single server and only communicated with a few clients. Applications are now deployed over numerous virtual machines (VMs), which interchange traffic flows. As VMs relocate in order to optimize and rebalance server workloads, the physical endpoints of current flows change over time. Furthermore, as the needs of the data center increase, so must the network. However, with the addition of hundreds or thousands of network devices that must be configured and managed, the network becomes significantly more complicated and difficult to handle.

SDN design isolates the underlying infrastructure from the applications that utilize it by separating the network control and data planes, allowing the network to become as programmable and controllable at scale as the computer infrastructure that it increasingly resembles. An SDN strategy facilitates network virtualization, allowing IT professionals to manage their servers, applications, storage, and networks using a standardized strategy and toolset. SDN adoption, whether in a carrier environment or a corporate data center or campus, may increase network management, scalability, and agility.

The future of networking will depend more and more on software, which will speed up the pace of network innovation in the same way that it has in computing and storage. SDN has the potential to turn today's static networks into flexible, programmable platforms with the intelligence to dynamically assign resources, the scalability to support massive data centers, and the virtualization required to support dynamic, highly automated, and secure cloud environments. With its numerous benefits and astounding industry momentum, SDN is on its path to becoming the new network standard [9].

3.3 Network Security Function

Network Security Function (NSF) is a term defined by the Interface to Network Security Functions (I2NSF) working group as "Software that provides a series of security-related services".

NSFs are defined as functions that enable the execution of some of the most critical security features, such as integrity, confidentiality, and asset availability. Furthermore, NSFs may monitor network traffic and interfere if malicious activity is identified. NSFs are produced and used in a wide range of settings. Users may use network security services enforced by NSFs hosted by one or more providers, which might be their own organization, service providers, or a mix of the two. Similarly, service providers may provide network security services to their clients that are enforced by several security products, functions from many vendors, or open-source technologies. Physical and/or virtualized infrastructure may be used to deliver NSFs. Without common interfaces to regulate and monitor the behaviour of NSFs, it has become nearly difficult for security service providers to automate service offerings that use numerous vendors' security functions. Obviously, NSFs from various suppliers offer a variety of functionality and interfaces. NSFs can also be put in several locations within a network and serve in a variety of ways. Security providers can utilize a variety of NSFs from various suppliers to ensure a wide range of security features.

3.4 Policy-based management

The management of the network infrastructure, and in general of distributed systems, is a task usually complex and difficult to manage. In an era of increasing technical complexity, it is becoming difficult to find qualified personnel able to keep up with the times, and therefore capable of managing the new and ever-changing features that are introduced in the various servers, routers, and switches at ever-increasing rates. In software-defined networks, policy-based network management provides a means by which the administration process can be simplified and largely automated. In fact, policies are technology-independent rules designed to improve the hard-coded functionality of managed devices by introducing interpreted logic that can be dynamically changed without changing the underlying implementation. This allows for a certain degree of programmability without the need to interrupt the operation of either the managed system or of the management system itself [11] [12].

The first steps towards this new management model were made by the Internet Engineering Task Force (IETF) in collaboration with the Distributed Management Task Force (DMTF), who proposed a policy framework that laid the foundations for all future projects. Simplification and automation are the key aspects of this framework which is an excellent example for studying the basic mechanisms of all policy-based management systems. The framework in question, shown in Figure 3.1, is characterized by four elements:

- Policy management tool: It is used by the administrator, or a common user, to define the policies to be enforced within the network;
- Policy repository: It is used to store the policies generated by the management tool;
- Policy Enforcement Point (PEP): It is a device that can apply and execute the different policies by using the policy decision point;
- Policy Decision Point (PDP): It is an intermediary to communicate with the repository, it is responsible for interpreting the policies stored in the repository and communicating them to the PEP.

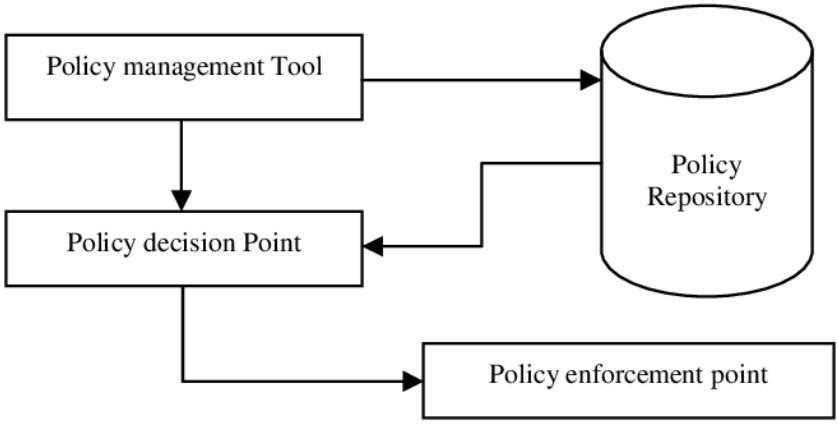

Figure 3.1: The IETF/DMTF policy framework

To fully understand the advantages of this approach, we must focus on the policy management tool and on how it manages, by exploiting the definition of policies, to simplify the provision and configuration of the various devices on the network. Two central aspects that allow this are: *centralization* and *high-level of abstraction*.

Centralization refers to the ability to provide configurations for all devices in one place (policy management tool), instead of having to configure each individual device. This solves two big problems of the old approach: firstly, it allows, as already said, to avoid the configuration of single devices which not only takes more time but can be more complex due to the different platforms and management interfaces of the devices themselves. Secondly, it allows a more uniform configuration of the devices and allows to have the same security policies and the same level of protection on all devices on the network, avoiding problems of non-uniform protection.

High-level of abstraction refers to the ability to provide policies written in a high-level language that is easier to understand for both business managers and simple users and this simplifies the work of administrators who can focus on other aspects.

A policy management tool, as mentioned previously, must support the concepts of high-level policies and technology-level policies. In particular, it must provide a translation mechanism and must ensure the transition from high-level to low-level policies. A generic policy management tool can be

constructed out of four basic components, as shown in Figure 3.2.

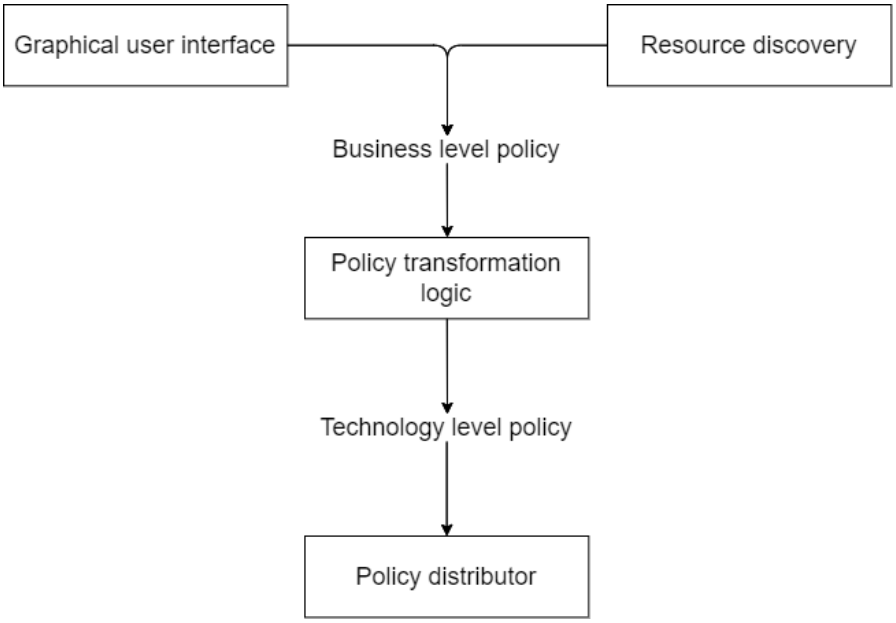

Figure 3.2: Generic policy management tool

The user interface is the means by which administrators or simple users can enter the high-level policies they want to implement. The interface can be done in various ways: simple command line, graphical tools or just the definition of the policies in a file. The resource discovery component establishes the network's topology, as well as the users and applications that are active on the network. The capabilities and topology of the network must be known in order to produce the configuration for the various devices in the network.

The policy transformation logic is responsible for checking that the high-level policies are consistent, correct and above all feasible with the existing capacity and topology of the network. It also deals with the translation of high-level to low-level policies.

The policy distributor takes care of low-level policies being configured on network devices, which can mean writing them to a central repository if supported or configuring single devices individually.

3.5 Interface to Network Security Functions I2NSF

Interface to Network Security Function (I2NSF) is a working group whose purpose is to make security network functions easier to deploy in SDN and NFV scenarios by standardizing the interface that allows the setting of such NSFs, even if they come from different manufacturers or employ different underlying technologies. As a result, network administrators may utilize common interfaces to regulate which rules are applied by which NSF and how each NSF must handle packets.

I2NSF attempted to develop a framework in which a Security Controller will be able to accept security policies from the service provider client, distribute such policies to the available NSFs, and monitor them.

Note that an NSF is defined, as said before, as software that provides a set of security-related services, such as:

- Detecting unwanted activity;
- Blocking or mitigating the effect of such unwanted activity in order to fulfil service requirements;
- Supporting communication stream integrity and confidentiality.

I2NSF will define interfaces at two functional levels for network security control and monitoring. The I2NSF Capability Layer describes how to govern and monitor NSFs at the functional implementation level. The I2NSF Service Layer defines how clients' security policies can be expressed to a security controller by standardizing a set of interfaces through which a security controller can invoke, operate, and monitor NSFs. The controller applies policies based on the different capabilities given by the I2NSF Capability Layer. The I2NSF Service Layer also enables the client to monitor their own rules.

3.5.1 I2NSF Problem definition

The increasing complexity and difficulty of maintaining a secure infrastructure, complying with regulatory requirements, and controlling costs are encouraging companies to use network security functions hosted by service providers. The hosted security solution is particularly appealing to small and medium-sized enterprises (SMEs) that lack security personnel to constantly monitor networks, acquire new skills, and provide quick mitigations to ever-increasing sets of vulnerabilities. But, as said before, because there are no standard interfaces to control and monitor the behaviour of NSFs, it is extremely difficult for providers of security services to automate service offerings that utilize different security functions from multiple vendors [13].

Security service providers can be either inside or external to the organization. For example, an internal IT security group within a big organization might serve as the enterprise's security service provider. An organization, on the other hand, might outsource all security services to an outside security service provider. Any two parties can form a "Customer-Provider" connection. The actors may be from distinct organizations or from different divisions within the same organization.

In such cases, contractual agreements may be necessary to explicitly establish the customer's security needs and the provider's commitments to meet those requirements. Such agreements may include information on protection levels, escalation procedures, alert reporting, and so on. Currently, there is no standard system in place to record such criteria. The activity of the I2NSF aims to solve these issues and obstacles [13].

NSFs come in a number of different forms. Various vendors' NSFs may have different features and interfaces. NSFs can be installed in many places within a network and may play diverse roles. Given the variety of security functions, the settings in which they might be employed, and the ongoing growth of these functions, standardizing all elements of security functions is difficult and unlikely to be practicable.

Fortunately, not all aspects must be standardized. For example, from the standpoint of the I2NSF, there is no requirement to standardize how each firewall's filtering is generated or applied. Some filtering capabilities in a

given vendor's product may be unique to the vendor's product, therefore standardizing these features is not required. A standardized interface is required to regulate and monitor the rule sets used by NSFs to treat packets flowing through these NSFs. Thus, standardizing interfaces will act as an enabler for standardizing existing security functions.

The I2NSF framework must also address the challenges facing customers. When customers utilize hosted security services, a collection of security functions deployed in multiple domains may enforce their security standards. Furthermore, customers may lack the security knowledge required to define adequately explicit expectations or security rules. They may also desire to express guidelines for security management, for example they may want to decide which critical communications to maintain during critical events, or to report network attacks and manage network connectivity.

Customers can use NSFs from several service providers. Customers must indicate to service providers their security requirements, rules, and expectations. In turn, service providers must translate this customer information into customer security rules and related setup procedures for their network's set of security functions. The service provider confronts several obstacles in the absence of a standardized interface that gives a precise technical characterization. First of all, the service provider must process the customer's input without a consistent interface, in fact, there could be extremely disparate interfaces and configuration models for security devices and security policies. Because of that, new innovative security products face a significant barrier to access to the market

There is also the problem of lack of immediate feedback, in fact customers may additionally demand a way for simply updating/modifying their security requirements with instant effect in the underlying NSFs affected. Customers also want to know which policies are actually being implemented, how much bandwidth is covered by those rules, and gather other information so that they may do better security analyses and risk assessments. I2NSF wishes to create simple solutions for customers to get this vital information [13].

3.5.2 I2NSF Use cases

Standard interfaces for monitoring and controlling the behaviour of NSFs are important building blocks for security service providers and companies looking to automate the usage of several NSFs from various manufacturers by their security management entities [13].

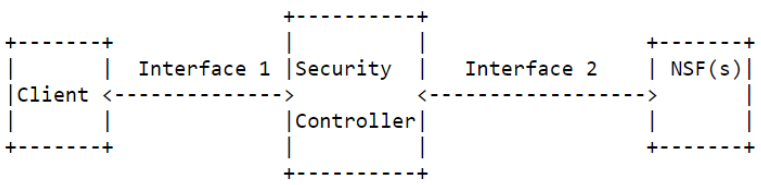

Figure 3.3: Interaction between Entities

As we can see in Figure 3.3, there is a Security Controller that acts as an intermediate between a client that want to upload or update a specific security policy and one or more NSFs that will have to enforce them.

In a *basic scenario*, users request security services through Network Service Providers' (NSPs) specialized clients, and the proper NSP network entity will activate the NSFs in response to the user service request. In this case the NSP network entity is denoted as Security Controller. Interface 1 receives security requirements from clients and converts them into commands that NSFs can recognize and execute. The security controller also returns to the client NSF security reports (e.g., statistics) that the security controller has acquired from NSFs. Interface 2 is used to communicate with NSFs based on the instructions and to obtain NSF status information. When users want to know the state of a security service, they can ask the client for NSF status. The security controller will take NSF data through Interface 2, aggregate it, and provide feedback to the client via Interface 1. This interface may be used to gather not just individual service information, but also aggregated data useful for activities such as infrastructure security evaluation. Customers may also require validating NSF availability, integrity, provenance, and execution. To accomplish this, the security controller may gather security measures and exchange them with an independent and trustworthy third party (through Interface 1) to enable attestation of NSF functions using the third-party additional information. This indicates that Interface 1 may be used by two

sorts of clients: the end user and a trustworthy, independent third party.

The *access networks scenario* offers use cases for users (such as a residential user, a corporate user, a mobile user, and a management system) that request, and control security services located on the NSP infrastructure. Given that NSP customers are basically users of their access networks, the situation is strongly connected to their characteristics as well as the usage of NSFs. In fact, different access clients may have different service requests:

- Residential users usually request services for parental control, content management, and threat management.
- Enterprise users usually request services for enterprise flow security policies and managed security services.
- Service providers usually request services for policies that protect service provider networks against various threats (including DDoS, botnets, and malware).
- Mobile users usually request services from interfaces that monitor and ensure user quality of experience, content management, parental controls, and external threat management

Some access users may be unconcerned with which NSFs are used to provide the services they have requested. In this situation, provider network orchestration systems can choose the NSFs to enforce the security policies specified by the clients. Other access users, particularly business customers, may choose to contract separately for dedicated NSFs for direct control.

In a *cloud data center scenario*, for a variety of reasons, network security devices such as firewalls may need to be dynamically added or deleted. These modifications might be requested expressly by the user or triggered by a pre-agreed-upon demand level in the Service Level Agreement (SLA) between the user and the service provider. For example, the service provider may be required to increase firewall capacity within a certain time frame whenever bandwidth use exceeds a certain level for a specified duration. This capacity increase might result in the addition of new firewall instances on current computers or the creation of a fully new firewall instance on a different machine [13].

Because security services are delivered on-demand and dynamically, it is preferable that network security "devices" are implemented in software or virtual form rather than physical appliance form. This is a provider-side need. Users of the firewall service are unconcerned about whether the firewall service is running on a VM or another form factor. Indeed, they may be unaware that their traffic is being routed through firewalls.

In a *DDoS, Malware, and Botnet attacks scenario* to help enterprises effectively protect their networks against these types of threats, the I2NSF architecture should have a client-side interface that is independent of use case and technology. A technology agnostic system is one that is general, independent of technology, and capable of supporting numerous protocols and data models. An I2NSF interface, for example, might be used to provide security policy configuration information that searches for specific malware signatures. Similarly, botnet assaults might be readily avoided by enforcing security regulations using the I2NSF client-side interface, which blocks access to botnet command and control servers.

3.5.3 I2NSF Framework

This section summarizes the I2NSF framework behaviour. As illustrated in Figure 3.4, an I2NSF User can employ security functions by submitting high-level security policies, which establish the security criteria that the I2NSF user wishes to implement, to the Security Controller via the Consumer-Facing Interface (CFI), which allows various users of an I2NSF system to establish, manage, and monitor security policies for particular flows inside an administrative domain. The consumer of I2NSF policies is unconcerned with the placement and implementation of I2NSF policies [14].

The Security Controller receives and analyses high-level security rules provided by an I2NSF User and determines what sorts of security capabilities are necessary to achieve these high-level security policies.

The Security Controller then finds NSFs with the necessary security capabilities and develops low-level security rules for each of the NSFs, ensuring that the high-level security policies are eventually implemented by those NSFs. Finally, the Security Controller communicates the low-level security

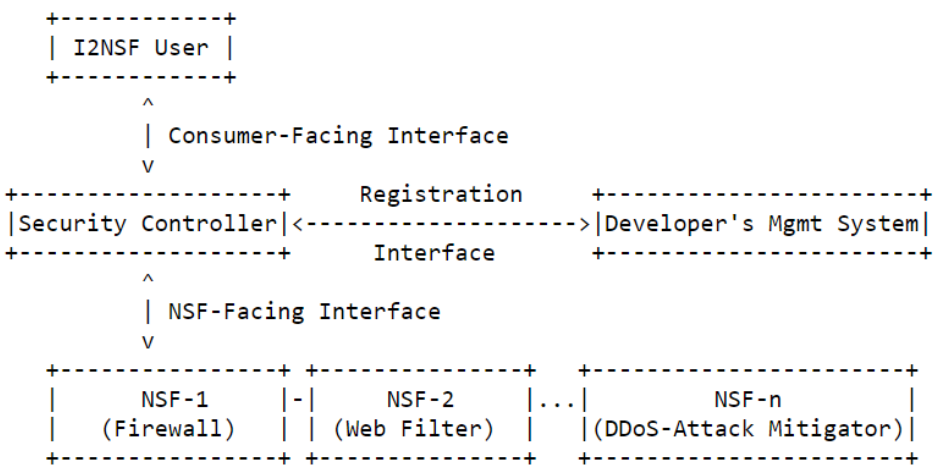

Figure 3.4: I2NSF framework

rules developed to the NSFs via the NSF-Facing Interface (NFI) [15], which is used to specify and monitor flow-based security policies implemented by one or more NSFs. It should be noted that the I2NSF Management System is not required to use all capabilities of a particular NSF, nor is it required to use all accessible NSFs. As a result of this abstraction, NSF features may be considered as building blocks by an NSF system, allowing developers to employ the security functions described by NSFs regardless of vendor or technology [14].

Developers (or vendors) use a Developer's Management System (DMS) to notify the Security Controller about the capabilities of the NSFs via the Registration Interface (RI), which is used to register (or deregister) the associated NSFs [15].

While providing much greater flexibility and, in many circumstances, being an indispensable solution given the deployment requirements, the use of externally provided NSFs introduces numerous additional security risks. The most serious risks linked with this type of security platform are as follows:

- An unknown/unauthorized user may attempt to impersonate another user who is authorized to access external NSF services. This attack may result in unauthorized access to the targeted user's I2NSF Policy Rules

and services, as well as the generation of network traffic outside of the security functions using a forged identity.

- A user may attempt to install flawed components (e.g., I2NSF Policy Rules or configuration files) in order to gain direct control of an NSF or the whole provider platform. For example, a user may attempt to intercept or manipulate the traffic of other users on the same provider platform by exploiting a vulnerability in one of the functionalities.
- A fraudulent provider can change the software's functionality to affect one or more NSFs. Because the provider has the greatest level of privileges managing the software's functioning, this incident has a significant impact on all users accessing NSFs.
- A consumer with physical access to the provider platform can change the behaviour of hardware/software components or the components themselves.

It is suggested that all users and applications employ authentication, authorization, accounting, and audit techniques while accessing the I2NSF environment. This may be improved further by forcing authorized parties to employ attestation to identify changes to the I2NSF environment. The features of these processes will establish the I2NSF environment's level of assurance [14].

3.5.4 I2NSF Flow security policy structure

An I2NSF Policy Rule consists of three Boolean clauses: an Event clause, a Condition clause, and an Action clause. This is also known as an ECA (Event-Condition-Action) Policy Rule. A Boolean clause is a logical statement that can only be TRUE or FALSE. It can include one or more terms; if more than one term is present, each term in the Boolean clause is connected using logical connectives (i.e., AND, OR, and NOT) [16].

Let's see these three terms in detail.

- An *Event* is defined as any significant event throughout the course of a change in the system being managed and/or the environment in which

the system is being managed. It is used in the context of I2NSF Policy Rules to decide whether or not the Condition clause of the I2NSF Policy Rule may be examined.

- A *Condition* is described as a collection of characteristics, features, and/or values that must be compared to a known set of attributes, features, and/or values in order to decide whether or not the set of Actions in that I2NSF Policy Rule can be implemented.
- An *Action* is used to control and monitor aspects of flow-based NSFs when the event and condition clauses are satisfied. NSFs provide security functions by carrying out numerous Actions.

Some clients may lack the necessary security skills to describe security requirements or policies specific enough to be implemented in an NSF. Instead, these clients may articulate expectations (e.g., objectives or purposes) about the functionality needed by their security rules. Customers may also express preferences, such as which sorts of destinations are, or are not, permitted for certain users. As a result, multiple layers of content and abstractions may be employed in Service Layer policies [14].

It is possible that a single policy flow will need the implementation of more than one NSF. It is not necessary for a user policy rule to be identical to the NSF low level language; I2NSF will begin by focusing on user policies that are as near to the flow security policies used by individual NSFs as feasible. An I2NSF user flow policy should have a structure similar to an I2NSF Policy Rule, but with more user-oriented expressions for packet content, context, and other aspects of an ECA policy rule. This allows the user to create an I2NSF Policy Rule without knowing the specific syntax of the required content (e.g., real tags or addresses) to match in the packets [14].

3.5.5 I2NSF Capability information and data model

Security Capabilities specify the tasks that Network Security Functions (NSFs) can perform in order to implement security policies. Security capabilities exist independently of the security control mechanisms that will be used

to implement them. I2NSF describes these Security Capabilities by means of a Data Model and an Information Model.

Every NSF should be characterized in terms of the capabilities it provides. Security Capabilities allow security functions to be specified in a vendor-independent way. That is, while constructing the network, it is not necessary to refer to a specific product or technology; rather, the functions described by their capabilities are taken into account. Security Capabilities are a market facilitator since they allow for the definition of tailored security protection by explicitly stating the security features provided by a certain NSF [16].

A Capability Information Model (CapIM) formalizes the capabilities offered by an NSF. This allows for the explicit specification of what an NSF may accomplish in terms of security policy enforcement, so that computer-based operations can refer to, utilize, configure, and manage NSFs without ambiguity. Network security specialists can refer to security control categories and understand each other. For example, network security specialists can all agree on what the words "NAT," "filtering", and "VPN concentrator" imply. For another example, network security specialists unambiguously define "packet filters" as devices that allow or reject packet forwarding depending on a variety of circumstances. In the case of other devices, such as stateful firewalls or application layer filters, further information is necessary. These devices filter packets or communications, however there are distinctions in the packets and communications they can classify as well as the states they retain. Network engineers handle these variations by asking further questions to establish the device's unique category and functioning. Machines may use a similar method, which is known as question-answering. In this context, the CapIM and the resulting data model can be useful by providing a rich source of information [16].

Chapter 4

System design

In this chapter the problem that the thesis tries to solve will be described and the design and details of the proposed framework will be presented.

4.1 Problem definition

The thesis work in question arises from the need to create a tool for the refinement of high-level policies into medium-level policies. This is one of the most critical phases that constitutes the work initiated by I2NSF, in fact the translation from a high-level security policy to the corresponding low-level security policy will be able to easily elevate I2NSF in real-world deployment.

The underlying problem is that NSFs cannot understand high-level policies directly, there is a need to translate them into a language closer to the low-level configuration language of the devices. Solving this problem would allow to maintain a high degree of generalization that does not bind the framework to the specific languages of each individual device and gives the opportunity to create a tool that can be easily updated and that is able to work with a higher level of automation than the current ones.

A rule in a high-level policy might include many abstract concepts that must be associated with concrete concepts, this work of associations which

for a human being is extremely simple becomes particularly complex for a machine, plus a security policy translator must handle this mapping in a flexible manner while understanding the general purpose of a policy statement.

4.2 Use cases

A policy translation tool for the proposed framework, as mentioned before, is a core problem whose resolution would lead to an enormous step towards the realization of effective systems for the enforcement of security policies in an automatic and intelligent way. In fact, the realization of this tool would allow the adoption of frameworks of this kind in various scenarios.

First of all, the cybersecurity market will have great benefits, in fact it is characterized by NSFs supplied by different vendors, who develop their products with custom interfaces and languages that are always different from each other, often making the integration of these functions into distributed systems very complicated. The main purpose of the proposed framework is to allow the integration of different NSFs produced by different vendors, so, in this way, there could be great improvements in this regard.

Another great advantage in creating a tool like this is that companies would no longer be tied to a single manufacturer, in fact it often happens that companies find themselves choosing the same vendor for fear of having to rebuild their entire service infrastructure from scratch if they should choose to change supplier. Thanks to the framework in question, it would be possible to integrate more NSFs from various suppliers because it would not be necessary to worry about the different low-level languages of the devices. Furthermore, all this would make the market more competitive with lower prices for companies that would not only have the possibility of integrating services from various vendors but would also have lower running costs.

4.3 Main goal

The main objective of this thesis is to demonstrate that it is possible to create a policy refinement tool that can automatically extract information from a high-level policy and partially replace the knowledge and competence of an expert network administrator. The work aims to produce a tool that works automatically and is extensible, therefore capable of being updated with new features and above all with the possibility of supporting more high-level policies.

It is important to note that the tool in question integrates perfectly with the work carried out by Cirella [2] who deal with the management of languages of lower abstraction. In fact, we were able to create a complete framework that starting from the high-level policies generates a language of medium abstraction with the tools object of this thesis and subsequently through the work of my colleague is able to directly generate the low-level languages of the individual devices.

4.4 Soution design

The extraction of information from high-level policies was performed through the use of an expert system, which is a computer system that simulates the decision-making abilities of a human expert. In particular, CLIPS was chosen, it is based on the Rete algorithm, as mentioned in Section 2.6, which is an extremely efficient algorithm for pattern matching, it uses forward chaining; various facts can make a rule applicable and when a rule becomes applicable is then fired.

CLIPS also provides a binding in python called CLIPSPY. I chose to use python due to various reasons, in particular, it allowed me to work with a programming language that I think is easier to understand and provides numerous advantages such as: simple syntax, vast libraries support and simple error debugging.

Thanks to CLIPSPY it is possible to make the assertion of facts which in turn are conditions of rules that once satisfied make the rule applicable. When

a rule is applicable it is immediately launched. A rule can in turn make the assertion of other facts or launch the execution of specific python functions. There is also the possibility of passing parameters to python functions directly from the knowledge base created in CLIPS, this last integration between the CLIPS inference engine and the functions written in python was very useful and effective as it allowed to exploit the mechanisms of reasoning of the CLIPS inference engine and the flexibility, ease of writing and immediacy of the python language which also has extensive library support.

4.5 System details

The proposed framework is composed of the main tools:

- Refinement tool: It deals with the enrichment phase; it takes an input XML file that contains the high-level policies and generates thanks to CLIPS expert system an intermediate file which contains all the necessary *Security Capability* for each rule that have to be defined for that specific policy.

- Converter tool: It deals with the conversion from the intermediate file generated by the Refinement to the *RuleInstance* XML file. It generates a *RuleInstance* file for each NSF configured in the intermediate file.

- Orchestrator: It is a python Flask web server that handles everything, it provides some endpoints in order to receive the input file containing the high-level policies and also the company database with all the information required. It executes the Refinement and the Converter tool. In this way it generates the *RuleInstance* files.

4.5.1 Description of files structure

First of all, it must be explained how the input and output files of the various tools are made.

High-level policy XML file

```
<hspl-list xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://fishy-project.eu/hspl"
          xsi:schemaLocation="http://fishy-project.eu/hspl hspl.xsd">
        <hspl id="hspl1">
            <subject>Alice</subject>
            <action>is authorized to access</action>
            <object>Internet traffic</object>
            <optionalField>time period</optionalField>
            <optionalValue>18:30 20:00</optionalValue>
        </hspl>
        <hspl id="hspl2">
            <subject>Alice</subject>
            <action>protect confidentiality</action>
            <object>Internet traffic</object>
        </hspl>
</hspl-list>
```

Listing 4.1: Example of HSPL.xml file

The file containing the high-level policies is an XML file and it is structured as shown in Listing 4.1, as can be seen within a single file it is possible to define multiple policies and each policy is characterized by some predefined fields:

- subject: is the user who needs to access or perform some operation on an object (e.g., employee, family member) and may be omitted if the policy is applied to the user that defines the policy.
- action: is the operation performed on the object (e.g., protect, authorize access, enable).
- object: is the entity (i.e., a resource such as e-mail scanning, Internet traffic, P2P traffic) target of the action.
- (Optional Field, Value Field): is an optional condition that add specific constraints to the action (e.g., time, content type, traffic type). The value part is a string with specific format depending on the field type.

Intemediate txt file

The intermediate file, as the name implies, is an output characterized by all the useful information extracted by the refinement engine and it is in a format that allows the subsequent conversion to *RuleInstance* files in a simple and immediate way.

```
1  -------------------------
2  (hspl (subject "Alice") (action "is authorized to access") (
       object "Internet traffic"))
3  (MatchActionCapability time)
4  (TimeStartConditionCapability 18:30)
5  (TimeStopConditionCapability 20:00)
6  (AcceptActionCapability)
7  (NSFs IpTables)
8  (IpDestinationAddressConditionCapability 192.168.0.0/30)
9  (IpSourceAddressConditionCapability 10.3.3.24)
10 (MatchActionCapability conntrack)
11 (ConnTrackStateConditionCapability NEW,ESTABLISHED)
12 (AppendRuleActionCapability FORWARD)
13 (Configure firewall-1 stateful from 10.3.3.24 to
       192.168.0.0/30)
14 -------------------------
15 (hspl (subject "Alice") (action "is authorized to access") (
       object "Internet traffic"))
16 (MatchActionCapability time)
17 (TimeStartConditionCapability 18:30)
18 (TimeStopConditionCapability 20:00)
19 (AcceptActionCapability)
20 (NSFs IpTables)
21 (AppendRuleActionCapability FORWARD)
22 (MatchActionCapability conntrack)
23 (ConnTrackStateConditionCapability ESTABLISHED,RELATED)
24 (Configure firewall-1 stateful from 192.168.0.0/30 to
       10.3.3.24)
25 (IpDestinationAddressConditionCapability 10.3.3.24)
26 (IpSourceAddressConditionCapability 192.168.0.0/30)
27 -------------------------
28 (hspl (subject "Alice") (action "protect confidentiality") (
       object "Internet traffic"))
29 (PolicySpiConditionCapability 0x23ec5997)
30 (IpProtocolTypeConditionCapability esp)
31 (NSFs XFRM)
32 (Configure vpn-gateway from 192.168.0.0/30 to 10.3.3.24)
```

```
(IpDestinationAddressConditionCapability 10.3.3.24)
(IpSourceAddressConditionCapability 192.168.0.0/30)
(PacketEncapsulationActionCapability transport)
(PolicyDirConditionCapability fwd)
(TemplateConditionCapability)
(IpSecRuleTypeActionCapability SecurityPolicy)
-------------------------
(hspl (subject "Alice") (action "protect confidentiality") (
   object "Internet traffic"))
(PolicySpiConditionCapability 0x23ec5997)
(IpProtocolTypeConditionCapability esp)
(NSFs XFRM)
(PacketEncapsulationActionCapability transport)
(TemplateConditionCapability)
(PolicyDirConditionCapability out)
(IpSecRuleTypeActionCapability SecurityPolicy)
(Configure vpn-gateway from 10.3.3.24 to 192.168.0.0/30)
(IpDestinationAddressConditionCapability 192.168.0.0/30)
(IpSourceAddressConditionCapability 10.3.3.24)
-------------------------
```

Listing 4.2: Extract of Intermediate.txt file

As you can see in Listing 4.2, it contains all the capabilities associated with each single policy and contains other textual information useful for understanding and managing the configurations of the various NSFs.

RuleInstance file

The last output generated by the tool is the RuleInstance for each single NSF that needs to be configured, the file name is constructed like this: [NSFname]_RuleInstance. The framework is able to generate a file for each NSF configured within the intermediate output generated by the Refinement. Obviously, every single NSF has its own specific language with its peculiarities even if the basic structure is shared.

```
<policy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="language_IpTables.xml"
   nsfName="IpTables">
    <rule id="0">
        <option operator="exactMatch">
```

```
4             <capabilityValue>
5                 <exactMatch>time</exactMatch>
6             </capabilityValue>
7         </option>
8         <acceptActionCapability />
9         <ipDestinationAddressConditionCapability operator="
    rangeCIDR">
10            <capabilityIpValue>
11                <rangeCIDR>
12                    <address>192.168.0.0</address>
13                    <maskCIDR>30</maskCIDR>
14                </rangeCIDR>
15            </capabilityIpValue>
16        </ipDestinationAddressConditionCapability>
17        ...
18        ...
19        <appendRuleActionCapability operator="exactMatch">
20            <capabilityValue>
21                <exactMatch>FORWARD</exactMatch>
22            </capabilityValue>
23        </appendRuleActionCapability>
24     </rule>
25     <rule id="1">
26        ...
27        ...
28        <ipDestinationAddressConditionCapability operator="
    exactMatch">
29            <capabilityIpValue>
30                <exactMatch>10.3.3.24</exactMatch>
31            </capabilityIpValue>
32        </ipDestinationAddressConditionCapability>
33        <ipSourceAddressConditionCapability operator="
    rangeCIDR">
34            <capabilityIpValue>
35                <rangeCIDR>
36                    <address>192.168.0.0</address>
37                    <maskCIDR>30</maskCIDR>
38                </rangeCIDR>
39            </capabilityIpValue>
40        </ipSourceAddressConditionCapability>
41     </rule>
42 </policy>
```

Listing 4.3: Extract of IpTables_RuleInstance.xml file

This is also an XML file, and you can see its structure in Listing 4.3, this file contains every single rule of the corresponding NSF. As previously mentioned, the framework generates a file for each NSF configured in the intermediate output generated by the Refinement and it is the task of the Converter to translate the intermediate file into these RuleInstance which can then be used directly as input to the tool resulting from the work of Cirella [2] to allow the subsequent translation into low-level language of the single device.

Chapter 5

System implementation

This chapter will describe all the implementation details of the proposed framework.

5.1 Refinement tool

The refinement work constitutes the core of this thesis work. An expert system, as previously mentioned, was used for the extrapolation of data from high-level policies. In particular, we used CLIPSPY a python binding for CLIPS.

The IT high-level security policy is sent into the refinement process, which generates abstract configurations for the security controls present in the landscape. The policy refinement tool is a modular framework that guides the administrator through all steps of the refinement process, allowing him or her to make intelligent choices and provide necessary information. In fact, the Policy Refinement application allows for the automatic selection of a refinement strategy, or a manual selection of the devices to configure for each IT security policy. In particular, if there is only one device configurable, it is selected automatically. If more than one device is configurable then the choice is left to the user. In this context there is space for further improvement, there are endless possibility to work on an automatic choice of the device to

configure when there are more than one configurable.

The refinement tool needs a series of information to function properly. First of all, as mentioned previously, it needs to have information about the landscape, i.e. the network topology and the devices present in it. The basic idea to make this possible was to use two distinct databases which, as we will see, will provide numerous useful and necessary information for the functioning of the refinement engine.

5.1.1 Company database

The first database is the one relating to the company; therefore, it contains all the information that only the internal technical staff of the company can have.

```
from ipaddress import IPv4Address, IPv4Network
import os

NetworkBackbone = {
    'Firewall1': ['Subnet1.1', 'SubnetDMZ', 'SubnetDecoyDMZ',
    'Firewall2', 'Internet'],
    'Internet': ['SubnetAway'],
    'Firewall2': ['Subnet2.1', 'Subnet2.2', 'FirewallHP', '
   VPNgateway'],
    'FirewallHP': ['SubnetHP'],
    'VPNgateway': ['Firewall3'],
    'Firewall3': ['Subnet3.1', 'Subnet3.2', 'Subnet3.3']
}

FirewallAndSubnet = {
    'Firewall1': 'firewall-1',
    'Subnet1.1': IPv4Network('10.1.1.0/24'),
    ...
}

DevicesNSF = {
    'firewall-1': ['XFRM', 'IpTables'],
    ...
    'vpn-gateway': ['XFRM', 'StrongSwan'],
    'firewall-HP': ['IpTables'],
    '10.3.3.24': ['XFRM', 'StrongSwan', 'IpTables'],
```

```
25 }
26
27 subnetIP = {
28     'Subnet1.1': IPv4Network('10.1.1.0/24'),
29     'SubnetDMZ': IPv4Network('10.1.2.0/24'),
30     'SubnetDecoyDMZ': IPv4Network('10.1.3.0/24'),
31     ...
32 }
33
34 sub_obj_IP = {
35     'Alice': IPv4Address('10.3.3.24'),
36     'Bob': IPv4Address('10.1.1.12'),
37     'Subnet1.1': IPv4Network('10.1.1.0/24'),
38     'SubnetDMZ': IPv4Network('10.1.2.0/24'),
39     ...
40     'Internet traffic': IPv4Network('192.168.0.0/30'),
41     ...
42 }
43
44 sub_obj_URL = {
45     'Charlie': 'www.charlie-domain.com'
46 }
```

Listing 5.1: Extract of company_database.py file

As we can see in Listing 5.1, this database contains information about the network topology, in particular the network backbone is represented which shows all the main connections of the network between the various subnets and the main network devices, allowing the tool to have a clear idea of the various existing connections.

All the IP addresses of the various subnets and of the possible subjects and objects, which may appear in the high-level policies, are also indicated. Subjects and objects can be indicated either by their IP addresses or by their URLs. If only the information on the URL is present, the tool is programmed to perform an automatic search to allow the association between URL and IP address, so as to allow in any case the identification of subjects or objects within the network topology.

This database also contains information about the NSFs configured on the various network devices, this information is necessary as only the company's network administrator can know how the various devices have been configured

and which NSFs are available on each of them.

This database, as we can see in the Listing 5.2, also contains information about the preferred protection algorithms to be used, indicating for each level of protection required (encryption, authentication, or both) the preferred algorithm, the preferred encryption or authentication mode and it also generates a random key of the correct length.

```
protectionALGO = {
    'Encryption': {'mode': 'cbc',
                   'algoEnc': 'aes128',
                   'key': os.urandom(16)},
    'Authentication': {'mode': 'hmac',
                       'algoHash': 'sha256',
                       'key': os.urandom(20)},
    'ConfAuth': {'mode': 'gcm16',
                 'algoAEAD': 'aes128',
                 'key': os.urandom(16)}
}
```

Listing 5.2: Extract of company_database.py file

The last information contained in this database is connected to the objects of the high-level policies, in fact it is possible to indicate general information regarding concepts such as DNS traffic or regarding objects belonging to the company, as in the case in the Listing 5.3, where there is an application called Web App for which information on the port and the IP protocol used are indicated with its associated capabilities. This object information is checked before the information contained in the second database, in this way the information entered by the company has a greater importance and overwrites the information in the second database.

```
objectsINFO = {
    'DNS traffic': [
        'DestinationType DNS',
        'IpProtocolTypeConditionCapability udp',
        'DestinationPortConditionCapability 53'
    ],
    'Web App': [
        'IpProtocolTypeConditionCapability tcp,udp',
        'DestinationPortConditionCapability 9999'
    ]
```

```
}
```

Listing 5.3: Extract of company_database.py file

5.1.2 Info database

This second database contains general information that industry experts usually have. As can be seen in Listing 5.4, it contains information on the high-level policy objects and associates the corresponding capability to each information.

```
INFO_obj = {
    'VoIP traffic': [
        'DestinationType VoIP',
        'IpProtocolTypeConditionCapability udp',
        'DestinationPortConditionCapability 5060'
    ],
    'DNS traffic': [
        'DestinationType DNS',
        'IpProtocolTypeConditionCapability tcp,udp',
        'DestinationPortConditionCapability 53'
    ],
    'Intranet traffic': [
        'DestinationType Intranet'
    ],
    'All traffic': [
        'DestinationType All'
    ]
}
```

Listing 5.4: Extract of info_database.py file

For example, it associates VoIP traffic with information such as the use of port 5060 and the use of UDP as the type of IP protocol.

5.1.3 CLIPS rules

Another central and fundamental aspect of the refinement engine are the rules that define the behaviour of the CLIPS inference engine. This set of

rules allow the tool to extrapolate information from the high-level language through forward chaining mechanisms, in fact it is possible to define rules that are launched when all their conditions are satisfied.

The conditions are formed by the facts contained in the knowledge base which, as mentioned in the Section 2.5, constitutes the set of information you have on that particular domain. Conditions can be linked via logical operators and there is also the possibility to use variables that can be used when the matched rule is executed. When a rule is executed, it can carry out the assertion of other facts and/or launch certain functions in the python environment, furthermore, these functions can be launched with parameters taken directly from the knowledge base.

```
RULES = [
    ...
    """
    (defrule object-association
        (not (hspl (object "")))
        (or (hspl (action "is authorized to access"))
            (hspl (action "is not authorized to access"))
            (hspl (action "protect confidentiality"))
            (hspl (action "protect integrity"))
            (hspl (action "protect confidentiality integrity
   "))
        )
        =>
            (object-filter-protection-analysis)
    )
    """,
    """
    (defrule filtering
        (or (hspl (action "is authorized to access"))
            (hspl (action "is not authorized to access"))
        )
        =>
            (assert (Case filtering))
    )
    """,
    """
    (defrule protection
        (or (hspl (action "protect confidentiality"))
            (hspl (action "protect integrity"))
```

```
29              (hspl (action "protect confidentiality integrity
    "))
30          )
31          =>
32              (assert (Case protection))
33      )
34      """,
35      """
36      (defrule filtering-association0
37          (hspl (action "is authorized to access"))
38          =>
39              (assert (AcceptActionCapability))
40      )
41      """
42      ...
```

Listing 5.5: Extract of rules.py file

As we can observe in Listing 5.5, each rule is linked to a particular function of the tool, usually the name is self-explanatory, and the basic concept is that it is possible to extend the functionality of the tool by adding rules to cover all future cases that will be introduced.

5.1.4 Network topology generation

The refinement tool needs the network topology in order to work properly. The information about the network backbone, as said before, are present in the company database. Every time the tool is executed it generates a network graph from this information. In particular, the tool uses two different python libraries: `NetworkX` and `Pyvis`.

The first one is used within the python environment, and it has no aesthetic purposes. It is needed to understand the location of subjects and objects within the network topology.

Pyvis, on the other hand, as we can see in Figure 5.1, provides a web interface that allows the visualization and, if enabled, the modification of the graphical representation of the topology. Pyvis provides the possibility to integrate NetworkX, this was helpful since it was possible to take advantage of this integration and re-use the NetworkX graph to build a Pyvis topology

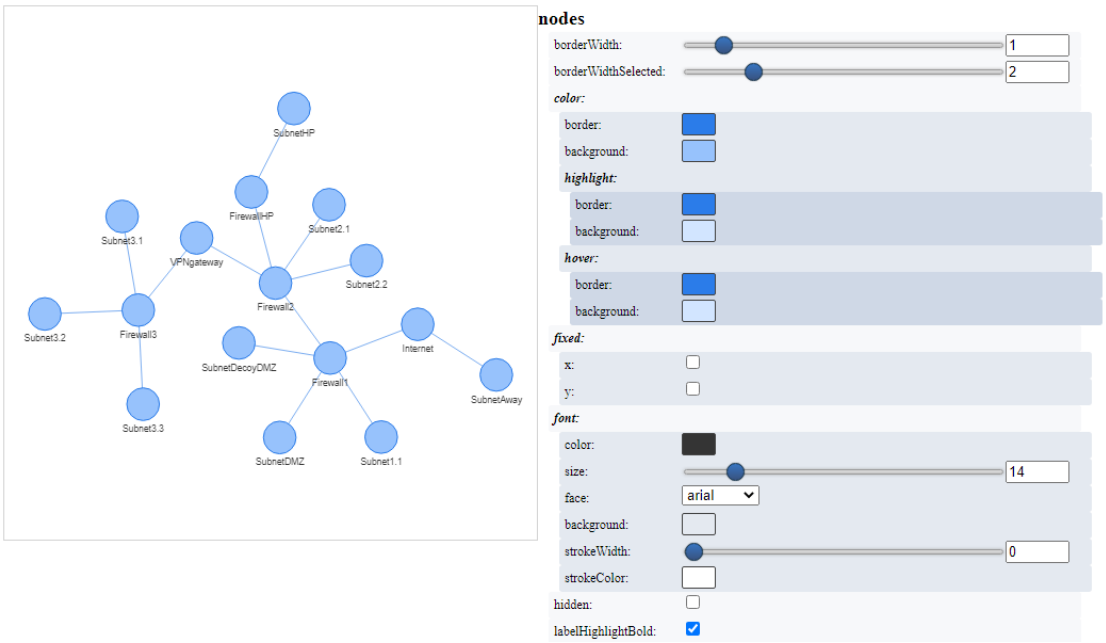

Figure 5.1: Pyvis network topology graph

that is more complete from a graphical point of view.

5.1.5 Device configuration selection

The device configuration phase is another critical aspect of the proposed tool, it takes place when, thanks to the CLIPS engine and the two databases, all the information has been extracted and when a path from the subject to the object of the high-level policy has been found.

So, when all the required capabilities for that specific policy are collected it is possible to query the NSF-catalogue, presented in Section 2.8, through the BaseX server made by Cirella [2] in order to obtain the list of NSFs that supports these capabilities.

The queries are made through the requests library in python and, once a response has been obtained, the company database is checked in order to understand if the devices on the path support at least one of the NSF received in the response.

As discussed before in Section 5.1, when the information on the configurable devices is obtained if only one device on the path have at least one NSF of the required ones it is automatically configured, if there are more than one device configurable a PysimpleGUI interface is shown, and the user have the possibility to choose which device configure.

5.1.6 Output generation

The output of the refinement tool is an intermediate file, which, as mentioned before, is a text file that contains all the information and all the details extracted from the high-level policies. The output is generated after the device configuration phase terminates through specific printing functions. The code, through recursive calls, manages the printing of the configurations for both outbound and inbound traffic. Specifically, it generates two configurations for the filtering rules and four for the protection ones, which must consider both the configuration of the state and the configuration of the policy itself.

As we discussed in the Section 4.5, each configuration is separated through a series of "-" characters, in this way the task of the Converter tool is made a lot easier. In fact, in this way it can identify in an immediate way a single policy that it has to add to the specific RuleInstance file.

5.2 Converter tool

The purpose of the Converter is to take the intermediate file generated by the refinement and generate the corresponding RuleInstance file for each NSF present among the various configurations. Specifically, the converter first deletes all the RuleInstance files present in the homonym folder and then begins the reading and analysis of the intermediate file, in particular, it separates the file into blocks according to the separator character "-" and analyses each single block that corresponds to a single policy of a specific NSF to be translated.

The first thing the Converter performs is the search for the "NSFs" line which contains all the configurable NSFs for that device.

As mentioned previously, the refinement tool does not make the choice on the NSFs if there are more than one NSF that supports the necessary capabilities on the device. This choice is made by the Converter, which randomly chooses an NSF among those that can be configured and creates the corresponding RuleInstance file if it has not been previously created. If the file already exists, the Converter will append the current policy to the file.

Once the creation of the XML RuleInstance file is done, the Converter begins line-by-line parsing of the block. Analyse the line to identify the capability and the corresponding value so as to be able to set all nodes of the XML file correctly.

When the parsing of the block is completed, the corresponding policy is written in the RuleInstance file associated to that NSF.

5.3 Orchestrator

The Orchestrator is the one who takes care of the management of the Refinement and the Converter tool. It deploys a Flask server that provides various APIs necessary for the execution of the various components of the framework.

Let's see all the endpoints present in the Flask server:

- Upload file: It allows to upload the file that contains the high-level security policies on the server.
- Upload database: It allows to upload the file containing information about the network topology but also information on the subjects and objects of the high-level policies and finally information on the preferred protection algorithms.
- Refinement: Endpoint that allows the user to start the refinement engine execution. In particular, it must be performed after uploading the file containing the high-level policies and the database.

- Refinement without GUI: Provides, as before, the possibility to perform the refinement starting from the high-level policies contained in the XML file but it performs the refinement without the generation of GUI. In particular, the graph with the network topology and the windows for selecting the devices to be configured for each individual policy are not opened.
- Converter: Enable the execution of the Converter, which takes the intermediate file generated by the Refinement and generates the RuleInstance for each configured NSFs. Returns the names of the RuleInstance files generated in JSON format.
- Download of RuleInstance: It provides the possibility to download a single RuleInstance file.

It is important to note that there is also a bash script that allows the automation of the last step, in fact if when you execute the penultimate step you redirect the output to a file and pass this `.json` file to the script, the latter executes the download of all the RuleInstance files generated by the Converter.

5.4 Workflow implementation

A general workflow of how the framework in question should be used is shown in Figure 5.2. It should be noted that the creation of high-level policies and the upload of the database, which contains information about the network topology, preferred protection algorithm and devices present on the network, are responsibilities of the user, he is responsible for creating them appropriately by editing, with an editor of his choice, the XML file that must contain the high-level policies and the python file containing the company information. The user must follow the structures presented in Section 4.5. The rest of the workflow proceeds autonomously, always according to the user's request, and checks for any possible errors by the consumer of the service.

1. Start the NSF-Catalogue: the first thing to do is to start the NSF-Catalogue, or make sure it is already running. In fact, the framework in

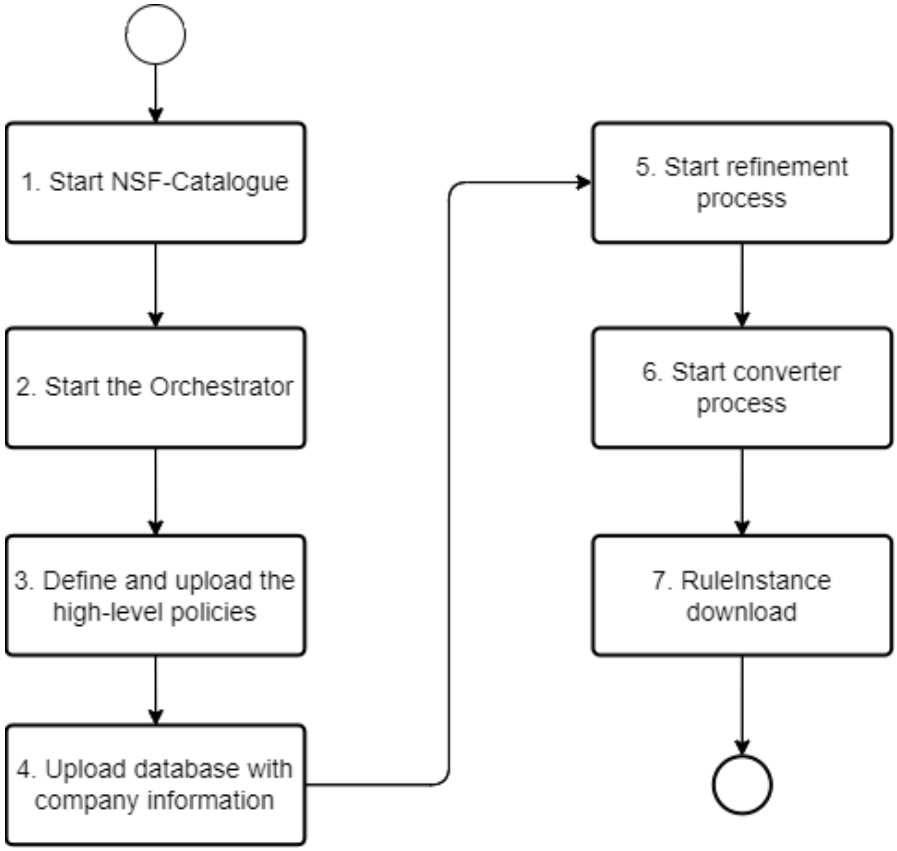

Figure 5.2: Framework workflow

question needs the BaseX server implemented by Cirella [2] to be active and ready to provide information about the NSFs and the supported Security Capabilities.

2. Start the Orchestrator: once the NSF-Catalogue has been set up, the Orchestrator that implements the Flask server must be started, which will receive all subsequent user requests and will take care of the upload of the files and of the execution of the Refinement and the Converter.

3. Define and upload of the high-level policies: the user has the task of creating the XML file that contains the high-level policies, which reflect the needs in terms of security required of the particular environment in which they will be inserted. He is also in charge of uploading the newly created file to the Orchestrator.

4. Upload database with company information: the user has the task of

uploading the file containing information about the company (or, in general, about the deployment environment) which includes information on the network topology, on the preferred protection algorithms and on the configurable devices in the network with related supported NSFs.

5. Start refinement process: once the necessary files have been uploaded, the refinement process can be carried out. The Orchestrator can run the Refinement tool, which takes the XML file containing the high-level policies and generates the intermediate file.

6. Start converter process: once the refinement process is finished, the Converter tool can be run. It takes the intermediate file generated by the Refinement and creates, in turn, the RuleInstance files according to the configured NSFs.

7. RuleInstance download: once the RuleInstance have been generated, they can be downloaded individually via the corresponding API. There is also the possibility to download them all together via a provided bash script which calls the corresponding API for each RuleInstance generated by the Converter.

Chapter 6

System validation

In this chapter the workflow presented in Section 5.4 will be followed and the framework in question will be applied to a real application scenario. This will allow to validate the thesis work carried out and will offer food for thought for possible future developments.

6.1 Use cases: IpTables and XFRM

The following paragraphs provide examples of use of the main functionalities of the proposed framework. *IpTables and XFRM NFSs* are taken into consideration in order to effectively show the functioning of the tool. Furthermore, the various output files will be explained in detail.

6.1.1 Network topology

The first thing to consider is the application scenario created. As can be seen in the Figure 6.1, the network topology created is quite complex, this has allowed us to carry out many tests of high-level policy implementations of various kind. In fact, as you can see, the network offers several configurable devices, each with its own characteristics.

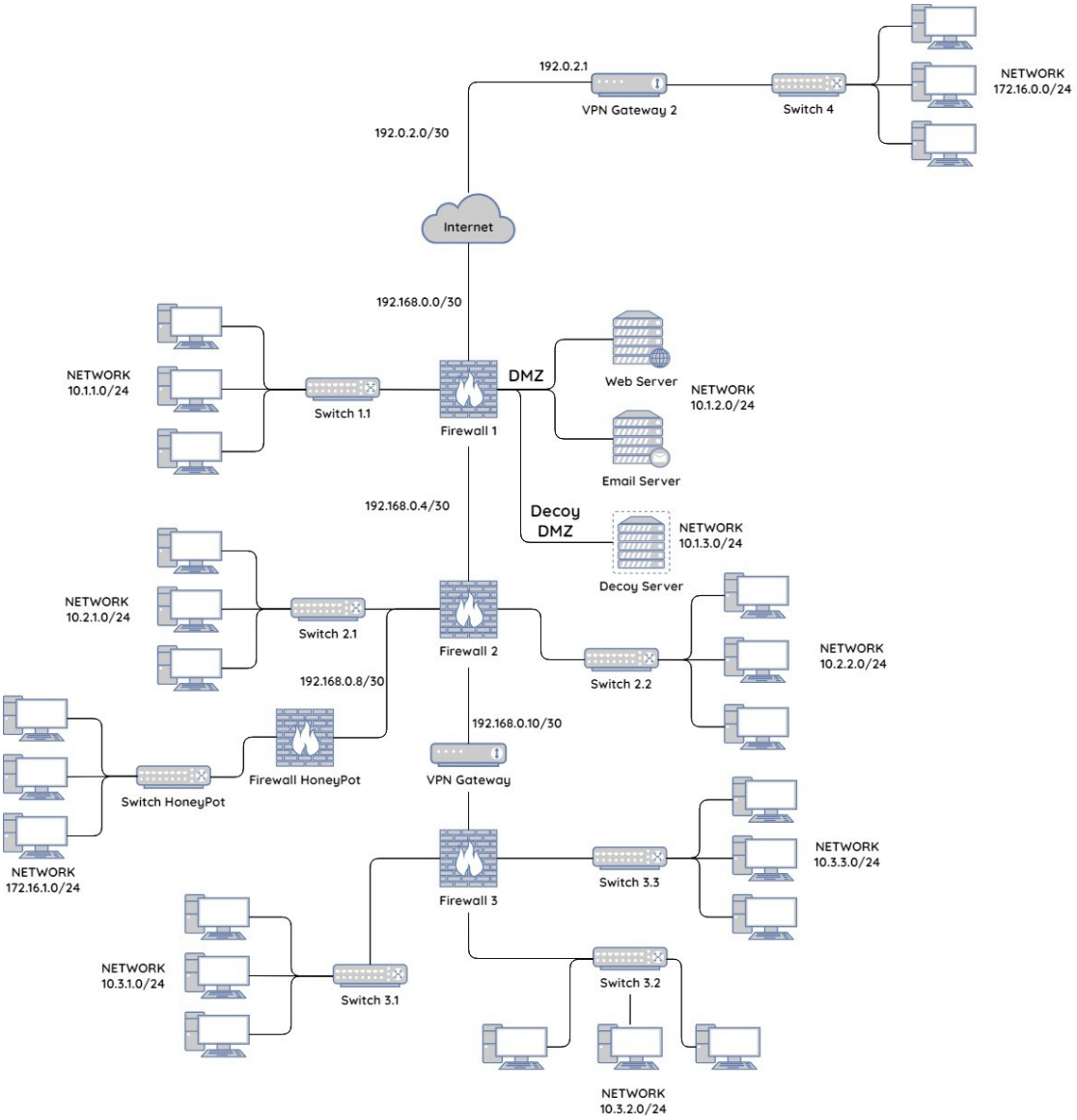

Figure 6.1: Network topology

The information about the topology of the network and of the devices that can be configured in it are contained within the company database, to analyse the structure of the database and understand how this information is contained within it, please refer to the Section 5.1.1 which shows extracts of the database which represents the network topology in question here.

It is important to note that the creation of the company database is

responsibility of the user who, as discussed in the Section 5.3, has the possibility to load it into the Orchestrator Flask server through the corresponding API.

6.1.2 High-level policy definition

The definition and creation of the file containing the high-level policies is another critical aspect that is up to the user. As for the company database, the user has the possibility to upload the high-level policies through the corresponding API provided by the Orchestrator.

The structure and details of the file containing the policies are shown in the Section 4.5, where you can see the various fields that make up a high-level policy. In this chapter we will see an example case using the policies shown in Listing 6.1.

```
<hspl-list xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance" xmlns="http://fishy-project.eu/hspl"
            xsi:schemaLocation="http://fishy-project.eu/hspl
    hspl.xsd">
        <hspl id="hspl1">
            <subject>Bob</subject>
            <action>is authorized to access</action>
            <object>Internet traffic</object>
            <optionalField>time period</optionalField>
            <optionalValue>18:30 20:00</optionalValue>
        </hspl>
        <hspl id="hspl2">
            <subject>Alice</subject>
            <action>protect integrity</action>
            <object>DNS traffic</object>
        </hspl>
</hspl-list>
```

Listing 6.1: HSPL.xml file

The use of these policies made it possible to test numerous main functions of the proposed framework, already described in the Chapter 5, including:

- Semantic analysis of the high-level policy fields: the framework was able

to recognize, thanks to a look-up in the company database, words like `Bob` and `Internet traffic`, associating them with the corresponding IP addresses.

- Path analysis: starting from the company database, the framework was able to find and reconstruct a path capable of connecting the `subject` of the high-level policy with the `object`.
- Capability recognition: once the path has been built, the framework, using the CLIPS inference rules, enriches its knowledge base and inserts all the capabilities required for the implementation of the high-level policy.
- Device recognition: once the required capabilities and the path have been found, the framework is able to identify which devices on the path support these capabilities and, depending on the mode of use selected, chooses, or let choose the devices to configure.
- Output generation: once the devices have been selected, the framework generates the intermediate output containing the CLIPS knowledge base already structured to be sent as input to the Converter, which will take care of the generation of the final RuleInstance.
- RuleInstance generation: once the Refinement has completed its process, the Converter can generate the RuleInstance of the corresponding NSF configured in the refinement process.

The first policy shows an example of traffic filtering, with the addition of an optional field about the time window in which the policy is active. In particular, the purpose of this security policy is to authorize Internet access to the user identified as `Bob` in the time slot 18:00-20:00. This policy, in addition to the basic mechanisms previously mentioned, will allow us to test the functioning of *IpTables NSF*.

The second policy, on the other hand, shows an example of protection that will allow us to test the operation of *XFRM NSF*. In particular, this security policy aims to protect the integrity of the `DNS traffic` of the user identified as `Alice`.

It is important to note that these, of course, were not the only tests carried out and above all the proposed framework independently selects the NSFs to

be configured according to the needs. In particular, regarding the filtering rules, there is also the possibility to configure stateless type policies. This happens if the configurable devices do not support, for example, *IpTables*. In fact, thanks to the work done by Cirella [2], there is an NSF called *genericPacketFilter* which allows the configuration in stateless mode of the devices that support it.

6.1.3 Refinement output analysis

Once the high-level policies have been defined, let's analyse the output of the refinement process in order to determine the correspondence of the generated output with the policies entered at the input.

```
1  -------------------------
2  (hspl (subject "Bob") (action "is authorized to access") (
       object "Internet traffic"))
3  (MatchActionCapability time)
4  (TimeStartConditionCapability 18:30)
5  (TimeStopConditionCapability 20:00)
6  (AcceptActionCapability)
7  (NSFs IpTables)
8  (IpDestinationAddressConditionCapability 192.168.0.0/30)
9  (IpSourceAddressConditionCapability 10.1.1.12)
10 (MatchActionCapability conntrack)
11 (ConnTrackStateConditionCapability NEW,ESTABLISHED)
12 (AppendRuleActionCapability FORWARD)
13 (Configure firewall-1 stateful from 10.1.1.12 to
       192.168.0.0/30)
14 -------------------------
15 (hspl (subject "Bob") (action "is authorized to access") (
       object "Internet traffic"))
16 (MatchActionCapability time)
17 (TimeStartConditionCapability 18:30)
18 (TimeStopConditionCapability 20:00)
19 (AcceptActionCapability)
20 (NSFs IpTables)
21 (AppendRuleActionCapability FORWARD)
22 (MatchActionCapability conntrack)
23 (ConnTrackStateConditionCapability ESTABLISHED,RELATED)
24 (Configure firewall-1 stateful from 192.168.0.0/30 to
       10.1.1.12)
```

```
25 (IpDestinationAddressConditionCapability 10.1.1.12)
26 (IpSourceAddressConditionCapability 192.168.0.0/30)
27 -------------------------
28 ...
```

Listing 6.2: First part of Intermediate.txt file

The first thing that can be noticed is the number of rules generated starting from the two high-level policies loaded in input. In fact, as discussed in Section 5.1.6, by observing Listing 6.2, we can notice how two rules were generated for the first high-level rule (which was a filtering policy) while, as we can see in Listing 6.3 four rules were generated for the second high-level policy (which was a protection policy).

This is due to the fact that the high-level filtering policies, in this case configured through the *NSF IpTables*, require two low-level rules for the configuration of the individual devices. In particular, two rules are necessary because both outgoing and ingoing traffic must be taken into consideration.

As we can see, therefore, in the two rules generated for the first high-level policy, `IpDestinationAddressConditionCapability` and `IpSourceAddressConditionCapability` are inverted, and the `ConnTrackStateConditionCapability` has different values, in fact it considers `NEW` connections for outbound traffic, while for inbound traffic it only considers `ESTABLISHED` or `RELATED` connections.

The rest of the Capabilities do not change. This is because they are not associated with the direction of the traffic.

It is important to note that the option regarding the time window is also managed correctly, as it associates the correct values with the corresponding Capabilities.

```
1 ...
2 -------------------------
3 (hspl (subject "Alice") (action "protect integrity") (object
     "DNS traffic"))
4 (IpProtocolTypeConditionCapability udp)
5 (DestinationPortConditionCapability 53)
6 (DataAuthenticationActionCapability authAlgoMode mode hmac
     algoHash sha256)
```

```
(DataAuthenticationActionCapability key 1323
   a5839d1ac38793bbf97e727ff45b0224e32d)
(IpSecRuleTypeActionCapability SecurityAssociation)
(PolicySpiConditionCapability 0x3a58f254)
(IpProtocolTypeConditionCapability ah)
(NSFs XFRM)
(IpDestinationAddressConditionCapability 0.0.0.0/0)
(IpSourceAddressConditionCapability 10.3.3.24)
(Configure 10.3.3.24 from 10.3.3.24 to DNS)
-------------------------
(hspl (subject "Alice") (action "protect integrity") (object
   "DNS traffic"))
(IpProtocolTypeConditionCapability udp)
(DataAuthenticationActionCapability authAlgoMode mode hmac
   algoHash sha256)
(DataAuthenticationActionCapability key 1323
   a5839d1ac38793bbf97e727ff45b0224e32d)
(IpSecRuleTypeActionCapability SecurityAssociation)
(PolicySpiConditionCapability 0x3a58f254)
(IpProtocolTypeConditionCapability ah)
(NSFs XFRM)
(Configure 10.3.3.24 from DNS to 10.3.3.24)
(SourcePortConditionCapability 53)
(IpDestinationAddressConditionCapability 10.3.3.24)
(IpSourceAddressConditionCapability 0.0.0.0/0)
-------------------------
(hspl (subject "Alice") (action "protect integrity") (object
   "DNS traffic"))
(IpProtocolTypeConditionCapability udp)
(PolicySpiConditionCapability 0x3a58f254)
(IpProtocolTypeConditionCapability ah)
(NSFs XFRM)
(Configure 10.3.3.24 from DNS to 10.3.3.24)
(SourcePortConditionCapability 53)
(IpDestinationAddressConditionCapability 10.3.3.24)
(IpSourceAddressConditionCapability 0.0.0.0/0)
(PacketEncapsulationActionCapability transport)
(PolicyDirConditionCapability in)
(TemplateConditionCapability)
(IpSecRuleTypeActionCapability SecurityPolicy)
-------------------------
(hspl (subject "Alice") (action "protect integrity") (object
   "DNS traffic"))
(IpProtocolTypeConditionCapability udp)
(PolicySpiConditionCapability 0x3a58f254)
```

```
46 (IpProtocolTypeConditionCapability ah)
47 (NSFs XFRM)
48 (PacketEncapsulationActionCapability transport)
49 (TemplateConditionCapability)
50 (PolicyDirConditionCapability out)
51 (IpSecRuleTypeActionCapability SecurityPolicy)
52 (Configure 10.3.3.24 from 10.3.3.24 to DNS)
53 (DestinationPortConditionCapability 53)
54 (IpDestinationAddressConditionCapability 0.0.0.0/0)
55 (IpSourceAddressConditionCapability 10.3.3.24)
56 -------------------------
```

Listing 6.3: Second part of Intermediate.txt file

As for the four rules generated for the second high-level policy (the protection one), the first thing to note is the `IpSecRuleTypeActionCapability` that allows us to distinguish the two rules for configuring the state, from the two rules for configuring the policy.

In fact, considering that we are in the case of using *XFRM NSF*, two rules are required for configuring low-level devices before applying the actual policy. These rules are called Security Association[1]; as you can see, these rules contain Capabilities that concern the configuration, as the name suggests, of the state. For example, the `DataAuthenticationActionCapability` contains information about the key, the methods and protection algorithms to be used. Remember that this information is obtained from the company database.

The two rules for configuring the actual policy are called Security Policy[2] and, as you can see, contain Capabilities such as `PacketEncapsulationActionCapability` and `PolicyDirConditionCapability` that concern the way in which the policy will be applied.

[1]The Security Association (SA) is a term used to describe the encryption techniques that are employed on IPsec packets. The IPsec protocols employ a security association, in which communication parties agree on common security properties such as algorithms and keys.

[2]Security Policy (SP) is a method that will put SA to use. The SP protocol specifies the IPsec security method for a connection. SP can specify a host or a port connection. When a policy is assigned to a connection, the kernel determines which security associations should be applied to that connection.

Furthermore, as for the filtering rules, it can be seen that, also in this case, the inversion of the `IpDestinationAddressConditionCapability` and of the `DestinationPortConditionCapability` between the case of outgoing traffic and incoming traffic is correctly managed.

6.1.4 Converter output analysis

Once the Intermediate file has been produced and analysed, it is possible to proceed with the execution of the Converter, which takes care of the generation of the final RuleInstance files.

Given the size of these files, only extracts will be presented, and we will analyse only the part observed in the previous section that is the most relevant. It will be important to verify the correspondence between the generated Intermediate file and the final RuleInstance files and their correctness from the syntactic point of view.

First of all, it is important to notice that two RuleInstance files have been generated. In fact, as mentioned previously, the cases under analysis concern *IpTables NSF* and *XFRM NSF*, which were the NSFs configured during the refinement process. In particular, it can be noted that an IpTables_RuleInstance.xml file has been generated for the first high-level policy and an XFRM_RuleInstance.xml file has been generated for the second high-level policy.

```
...
    <rule id="0">
        ...
        ...
        <ipDestinationAddressConditionCapability operator="
   rangeCIDR">
            <capabilityIpValue>
                <rangeCIDR>
                    <address>192.168.0.0</address>
                    <maskCIDR>30</maskCIDR>
                </rangeCIDR>
            </capabilityIpValue>
        </ipDestinationAddressConditionCapability>
        ...
```

```
14          ...
15          <connTrackStateConditionCapability operator="union">
16              <capabilityValue>
17                  <union>
18                      <elementValue>NEW</elementValue>
19                      <elementValue>ESTABLISHED</elementValue>
20                  </union>
21              </capabilityValue>
22          </connTrackStateConditionCapability>
23          <appendRuleActionCapability operator="exactMatch">
24              <capabilityValue>
25                  <exactMatch>FORWARD</exactMatch>
26              </capabilityValue>
27          </appendRuleActionCapability>
28      </rule>
29      <rule id="1">
30          ...
31          ...
32          <ipDestinationAddressConditionCapability operator="
    exactMatch">
33              <capabilityIpValue>
34                  <exactMatch>10.1.1.12</exactMatch>
35              </capabilityIpValue>
36          </ipDestinationAddressConditionCapability>
37          <ipSourceAddressConditionCapability operator="
    rangeCIDR">
38              <capabilityIpValue>
39                  <rangeCIDR>
40                      <address>192.168.0.0</address>
41                      <maskCIDR>30</maskCIDR>
42                  </rangeCIDR>
43              </capabilityIpValue>
44          </ipSourceAddressConditionCapability>
45      </rule>
46  ...
```

Listing 6.4: Extract of IpTables_RuleInstance.xml file

As before we will start analysing the output for the first high-level policy, which is contained in the IpTables_RuleInstance.xml file. As can be seen in Listing 6.4, two rules have been generated. In particular, it can be noted that the first rule (`rule id = "0"`) is the one regarding outgoing traffic, while the second rule (`rule id = "1"`) is the one regarding inbound traffic.

In the first rule, we can notice how the nodes for the definition of the Capabilities are constructed in a correct way. For example, it can be noted that the `ipDestinationAddressConditionCapability` has the correct attribute `operator` and is made up of the corresponding sub-nodes `capabilityIpValue` and `rangeCIDR`. It is also interesting to note how the `connTrackStateConditionCapability` has been managed correctly despite the presence of multiple values through the `operator union` and how, also in this case, the sub-nodes have been inserted appropriately.

Regarding the second rule, we can see how the `ipDestinationAddressConditionCapability` has been correctly reversed, together with the `ipSourceAddressConditionCapability`, with respect to the previous rule for the management of return traffic. It is essential to highlight the fact that, also in this second rule, as well as the first, the entire file follows the structure defined in the works of Avallone [1] and Cirella [2], who dealt with the definition of the Capabilities and the translation of the RuleInstance files into low-level rules for device configuration.

The dissertation remains largely unchanged with regards to the second high-level policy, which generated the XFRM_RuleInstance.xml file. It can be noted that together with the `rule id` there is also the `ruleType` attribute which allows to distinguish the Security Association rules from the Security Policy rules. Regarding the former, we can see how the `dataAuthenticationActionCapability` has been added correctly with all the corresponding sub-nodes. While for the latter it can be noted that the corresponding Capabilities for the Security Policy rules have been inserted, such as: `packetEncapsulationActionCapability` and `policyDirConditionCapability`.

```
...
    <rule id="0" ruleType="SecurityAssociation">
        <ipProtocolTypeConditionCapability operator="
    exactMatch">
            <capabilityValue>
                <exactMatch>udp</exactMatch>
            </capabilityValue>
        </ipProtocolTypeConditionCapability>
        <destinationPortConditionCapability operator="
    exactMatch">
```

```
            <capabilityValue>
                <exactMatch>53</exactMatch>
            </capabilityValue>
        </destinationPortConditionCapability>
        <dataAuthenticationActionCapability>
            <authAlgoMode>
                <mode>hmac</mode>
                <algoHash>sha256</algoHash>
            </authAlgoMode>
            <key>1323a5839d1ac38793bbf97e727ff45b0224e32d</
    key>
        </dataAuthenticationActionCapability>
        <policySpiConditionCapability operator="exactMatch">
            <capabilityValue>
                <exactMatch>0x3a58f254</exactMatch>
            </capabilityValue>
        </policySpiConditionCapability>
        ...
        ...
     </rule>
     ...
     ...
     <rule id="2" ruleType="SecurityPolicy">
        <ipProtocolTypeConditionCapability operator="
    exactMatch">
            <capabilityValue>
                <exactMatch>udp</exactMatch>
            </capabilityValue>
        </ipProtocolTypeConditionCapability>
        ...
        ...
        <packetEncapsulationActionCapability operator="
    exactMatch">
            <capabilityValue>
                <exactMatch>transport</exactMatch>
            </capabilityValue>
        </packetEncapsulationActionCapability>
        <policyDirConditionCapability operator="exactMatch">
            <capabilityValue>
                <exactMatch>in</exactMatch>
            </capabilityValue>
        </policyDirConditionCapability>
     </rule>
     ...
     ...
```

Listing 6.5: Extract of XFRM_RuleInstance.xml file

Obviously, the analysis made previously is also valid for this second file, as the entire file follows the structure defined by Avallone [1] and Cirella [2] for the generation of a file that conforms to the one defined in their work and it is ready to be processed by the framework proposed by them for the generation of low-level rules for the configuration of physical devices.

Chapter 7

Conclusion and future developments

The study in question, which is part of the policy-based management systems field, started from an analysis of the work done by the Interface To Network Security Functions (I2NSF) working group and the work done by Avallone [1] in his thesis and it has the goal to create a framework for the security policy refinement able to integrate with the work carried out by Avallone [1] himself and by Cirella [2].

The initial phase of the work was characterized by an analysis of the problem and, in general, by the study of previous works and the state of the art in the academic field and beyond. This made it possible to evaluate multiple options and to find the path that best suited the problem in question.

The main problem that was solved with the thesis in question was the creation of a framework capable of translating high-level policies into a medium-level language defined by Avallone [1] and updated by Cirella [2] that could allow policy-based management of a complex network system. In particular, the framework is able to receive in input a series of high-level policies defined in XML and, for each of them, it is able to associate the necessary security capabilities to apply the policy. The framework integrates perfectly with the work done by Cirella [2] and, in fact, exploits the NSF-Catalogue created by him to obtain the list of NSFs that support

the capabilities found by the framework in question. Furthermore, it is able to recognize the network topology through the use of a database and is able to independently select, or leave the choice to the user, the devices to be configured in the network according to the NSFs supported by the latter.

Possible future developments certainly concern the support of various high-level policies, in fact the framework currently only supports filtering policies, through the use of IpTables NSF and genericPacketFilter NSF, and protection policies, through the use of XFRM NSF. Another thing that leaves room for further future developments is better management of automation regarding the selection of devices to be configured within the network. In fact, the framework currently leaves the user the possibility to choose or randomly select a configurable device that supports the required capabilities. What could be done is to implement a system that can monitor the volume of rules configured on each individual device and compare them with the capacity of the device itself. In this way it could be possible to choose to distribute the configuration of the devices in a uniform and efficient way.

Appendix A

User manual

This section shows how a user can use the proposed framework. A fundamental prerequisite for using the tool in question is the use of a python version not lower than 3.9, since some functions and some libraries are only supported from this version onwards.

Another thing to keep in mind is that the proposed framework uses the *NSF-Catalogue* developed by Cirella[2] in his thesis work, for any details about its operation and the necessary prerequisites, please refer to his thesis.

A.1 Start the NSF-Catalogue

As previously mentioned, the framework developed during this thesis work uses the NSF-Catalogue. The first thing to do is to start this server, which will be used by exploiting its API which returns the NSFs that support the Security Capabilities sent to it.

In order to build the docker containing the NSF-Catalogue:

```
docker build -t register_and_planner .
```

In order to start the NSF-Catalogue:

```
docker run -p 8984:8984 register_and_planner
```

A.2 Start the Orchestrator

The Orchestrator is the one who takes care of the management of the Refinement and the Converter tool. It deploys a Flask server that provides various APIs necessary for the execution of the various components of the framework.

In order to start the server:

```
python orchestrator.py
```

A.3 Upload high-level security policies

Once the server is ready to be used, you can define the high level policies, following the indications provided in the Section 4.5.1, and upload the XML file to the server by running the following command:

```
curl -X POST -F file=@path/to/HSPL.xml http://localhost:5000/
   upload
```

A.4 Upload the company database

The user can upload the company database, which, as mentioned in Section 5.1.1, is a file containing information about the company (or, in general, about

the deployment environment) which includes information on the network topology, on the preferred protection algorithms and on the configurable devices in the network with related supported NSFs.

To upload this file to the server the user should follows this syntax:

```
curl -X POST -F file=@path/to/company_database.py http://
   localhost:5000/upload_database
```

A.5 Refinement process

Once the necessary files have been uploaded, the refinement process can be carried out. As mentioned in Section 5.1, the refinement process is the core of the proposed framework, it deals with the extrapolation of information from the high-level policies and generates an intermediate output which will then be used by the converter for the generation of the final RuleInstance files. The refinement process can be done in two distinct ways: with or without the appearance of a GUI.

A.5.1 Start refinement process with GUI

The first way in which the user can perform the refinement process is with the appearance of a GUI that allows the user to select the devices to be configured, when obviously more than one configurable device is available.

In order to execute this modality:

```
curl -X GET http://localhost:5000/refinement
```

A.5.2 Start refinement process without GUI

The second way in which the user can perform the refinement process is without the appearance of a GUI, in this case the user is not given the possibility to select a device to configure, which is chosen automatically by the framework itself.

In order to execute this modality:

```
curl -X GET http://localhost:5000/refinement_no_gui
```

A.6 Converter process

Once the intermediate file has been generated by the Refinement, it is possible to execute the Converter, whose purpose is to take the Intermediate file and generate the corresponding RuleInstance file for each NSF present among the various configurations. The Converter returns the list of RuleInstance files generated starting from the NSFs present in the Intermediate file produced by the Refinement. This list is shown on the terminal, or you can redirect the output to a file so you can use a script provided within the framework that allows you to download all the RuleInstance generated together. This allows the user to perform the last two steps in two different ways.

A.6.1 Download each RuleInstance file individually

First of all you have to run the Converter which will return the list of generated RuleInstance files.

To execute the Converter:

```
curl -X GET http://localhost:5000/converter
```

Following the user can download each RuleInstance file individually like this:

```
curl -X GET -O http://localhost:5000/download/[
   RuleInstance_name].xml
```

A.6.2 Download all RuleInstance files

To download all the RuleInstance files altogether, first of all, as before, the Converter should be run. This time, however, the output will not be printed on the terminal, but will be redirected to a file.

To do this the user can use this command:

```
curl -X GET http://localhost:5000/converter > configured_NSFs
   .json
```

Once this command has been executed, it is possible to use a script proposed within the framework that will download all the files generated by the Converter. To do this the user can run the following command:

```
./RuleInstance_downloader.sh configured_NSFs.json
```

Appendix B

Developer manual

This section contains the specifications and more detailed information on the functions of the proposed framework, this allows a greater understanding of the various tools.

In particular, the many components of the framework will be explained in detail and the methods for adding new methods and functionalities or modifying existing ones will also be indicated. The main used language is Python.

B.1 Refinement tool

The refinement process constitutes the core of this thesis work. An expert system was used for the extrapolation of data from high-level policies. In particular, we used CLIPSPY a python binding for CLIPS.

As mentioned in Section 5.1, the refinement process is governed in its functioning by the inference rules of the CLIPS expert system and uses the information of two distinct databases for various purposes. These dependencies are shown in Figure B.1.

The refinement, as mentioned previously, can be done in two different ways: with or without a GUI. The details of the version that supports a GUI

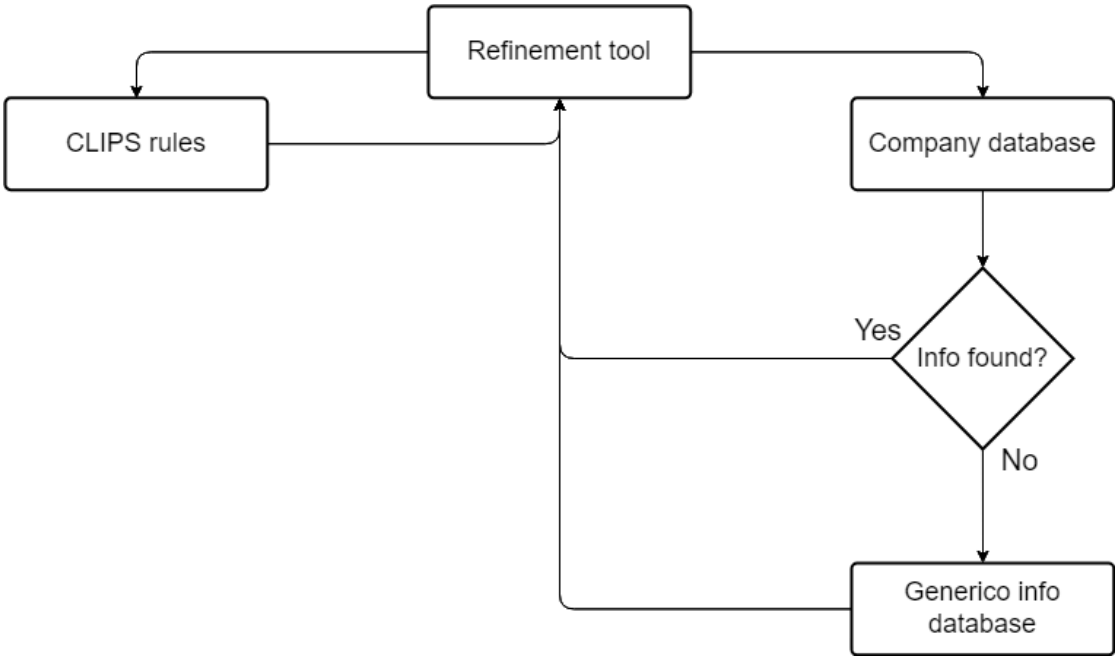

Figure B.1: Refinement process dependencies

will be presented first and then the differences in the version without a GUI will be listed.

B.1.1 Refinement with GUI

This section will review the main steps, presented in Section 5.4, of the execution of the refinement tool and will explain its main functions and all that is relevant from a development point of view, to allow, in the future, to insert new functionalities or modify existing ones.

To be as effective as possible, the main functions will be presented according to the execution order, which however is not fully deterministic as it depends on the CLIPS inference engine, in fact its *rules* play an important role in the execution order of the functions of the tool.

The main functions present within the refinement tool are:

- `main()`: It deals with the generation of the network topology graph

through the auxiliary function `build_graph()`, and it takes care of the initialization of the CLIPS environment; the definition of all the functions that will be called by the CLIPS inference engine and the `build` of the CLIPS rules. It then takes care of parsing the file containing the high-level rules. In particular, it searches all the high-level policies contained in the file, and each time it finds one it identifies the various fields (`subject`, `object`, `action`, etc.) and enters this information in the CLIPS knowledge base. Once inserted, it runs the environement (`env.run()`). When the execution of the environment terminates, it is reset (`env.reset()`) and the process start all over again for the subsequent high-level policies contained in the XML file.

  - `build_graph()`: Using the information contained in `company_database.py`, it reconstructs the network topology graph using the `NetworkX` library that will be used within the framework for all reasoning on the position of the various network devices. Subsequently, it generates another graph via the `Pyvis` library, which is more graphically complete, for display purposes to the user.

- `subject_analysis()`: It is usually one of the first to be performed by the tool and it is performed when a `subject` is inserted in the CLIPS knowledge base, it allows the identification of the `subject` of the high-level policy. In particular, it first performs a text analysis that allows the recognition of the logical operators "*and*" and "*exclude*" and then manages all the possible combinations between the supported values of `subject` with these logical operators. At the end a support function is used, called `database_subject_search(sub, exclude)`.

  - `database_subject_search(sub, exclude)`: It is in charge of searching the database for the `subject` of the high-level policy and inserting this information into the CLIPS knowledge base. At the beginning it does a search in `company_database.py` among the IPs to associate its correspondent to the `subject`, in case it does not find a match, it also searches among the information of the URLs. The information can be of inclusion or exclusion of the determined `subject` according to the parameters passed in input. Also, if `subject` is not found in the database, a check is made to verify that subject is an IP address itself. If it is an IP address, it is

entered in the CLIPS knowledge base, otherwise Internet is entered as source by default.

- `object_filter_protection_analysis()`: This function is performed when there is an `object` in the CLIPS knowledge base together with an `action` that can be associated with filtering and protection policies. This is due to the fact that, only with this kind of action, it is necessary to consider the object as a destination (i.e., an element to be reached). This function first searches the company database to get an IP or an URL associated with the object and then performs a search first in the `company_database.py` and then in the `info_database.py` to obtain the prior knowledge information that cannot be deduced from the expert system, such as the port number for a given type of traffic or the protocol used.

- `start_path_search()`: It is performed when all the information on the sources and destinations of the rule has been entered in the CLIPS knowledge base. The function therefore, first of all, searches the knowledge base for this information and, subsequently, for each combination between source and destination, launches an auxiliary function called `path_search(source, destination)`. Please note that the function is also able to distinguish the cases (called `near_configuration`) where there is no real physical destination, such as the case of filtering a certain type of traffic (for example, VoIP), in this case it skips the auxiliary function `path_search(source, destination)`.

  - `path_search(source, destination)`: This function that receives a source and a destination, using the `shortest_path` function of the `NetworkX` library, finds the shortest path between the source and the destination and then calls the function: `device_configuration_selection(path, source, destination)`.

- `device_configuration_selection(path, source, destination)`: It deals with the identification and selection of the devices on the path that connects the source to the destination. Firstly, it identifies all the *Security Capabilities* within the CLIPS knowledge base and subsequently, for each device on the path, it first search in the company database all the *NSFs* supported by that device and, subsequently,

makes a request to the NSF-Catalogue to discover the *NSFs* that have those certain capabilities. The function also deals with the generation of a GUI to allow the user to select the device if there are more than one configurable on the path. Once the selection has been made, the function carries out an analysis to understand if the devices to be configured support a `stateless` or `stateful` configuration for the filtering policies so as to be able to add the corresponding *Security Capabilities* in the knowledge base. It also takes care, for protection policies, of selecting the best transport mode: `tunnel` if the connection is via the Internet, otherwise `transport`. This function also uses an auxiliary function called `device_position_identification(device, path)` to enrich the knowledge base of information that requires comprehension of the position of the device to be configured on the path.

  - `device_position_identification(device, path)`: It receives the device and the path as input and, depending on its position and the type of policy in question, updates the CLIPS knowledge base with appropriate *Capabilities*.

- `protection_algorithm_sel(action)`: It takes care of going to retrieve the preferred security information in the company database, according to the `action` passed as input, and insert it into the knowledge base. It also deals with the generation of a random `SPI` for the protection policies implemented with *XFRM NSF*.

- `check_destination_device(obj, capability, value, action)`: It receives an `object`, a `capability` with the corresponding `value` and an `action` as input. In this way it takes care of checking if the `object` supports that particular `capability` and generates the necessary output in the *Intermediate.txt* file.

- `multiple_device_facts_printer()`: It takes care, through the use of an auxiliary function called `facts_printer()`, to add the information about the extracted *Security Capabilities* to the `Intermediate.txt` file. In particular, it is also responsible for managing the generation of the rules needed to handle return traffic. In fact, it eliminates the information to be changed or redundant and, through a recursive call, it takes care of adding the information for the return traffic with

the appropriate modifications. In detail, the recursion is interrupted by checking the `global position` variable, which helps to correctly configure the devices in this phase.

  - `facts_printer()`: It filters the *facts* in the CLIPS knowledge base that are not relevant but were used for secondary purposes and adds the others to the `Intermediate.txt` file.

B.1.2 Refinement without GUI

Refinement without a GUI differs from what was described in the previous section only in two functions:

- `device_configuration_selection(path, source, destination)`: It performs the same things as the previous case, the only difference is that no GUI is generated for selecting the devices to be configured on the path, but this selection is done automatically by the tool. So far, the selection takes place in random mode if there is more than one configurable device on the path, but there are countless possibilities for implementation in this regard.

- `build_graph()`: It only generates the graph through the `NetworkX` library, it doesn't generate and show to the user the graph of the network topology through the `Pyvis` library.

B.2 Converter

The Converter's objective is to take the intermediate file produced by the refinement and generate the equivalent RuleInstance XML file for each NSF present in the various configurations. Specifically, the converter deletes all RuleInstance files in the corresponding folder before beginning the reading and processing of the intermediate file.

In order to do that, the Converter only uses two functions:

- `main()`: It takes care of removing the RuleInstance files already existing in the corresponding folder and separating the content of the intermediate file generated by the Refinement according to the separator "-". Once the file is separated into blocks, it is analyzed piece by piece by the `block_analysis(block)` function.

- `block_analysis(block)`: It performs two cycles on the block analyzing the contents line by line. During the first iteration it searches the line containing the various NSFs configurable on that device selected during the Refinement phase and, if more than one is present, it randomly selects one. If in the folder containing the RuleInstance files there is already a file to contain the mid-level policy of that NSF, the function identifies the last configured rule and calculates the `rule_id` for the rule corresponding to the block being analyzed. If, on the other hand, there is no RuleInstance file for the selected NSF in the folder then create the file and set `rule_id` to 0. During this first cycle, the lines with irrelevant information are also deleted. In the second iteration, always on the same block, the analysis always takes place line by line. This time the tool analyzes the *Security Capability* relating to that line through a series of conditions and inserts the corresponding nodes and sub-nodes in the XML file.

B.3 Orchestrator

The Orchestrator is in charge of the administration of the Refinement and the Converter tool. It sets up a Flask server, which exposes numerous APIs required for the framework's various components to run.

The main functions of the Orchestrator that correspond to the APIs provided to the user, as discussed in Section 5.3, are the following:

- `upload_file()`: It takes care of loading the file containing the high-level policies, checks that the `POST` request fields are correctly inserted and that the file has an extension among those allowed through the support function `allowed_file()`. Then check that the filename is safe using the `secure_filename()` function of the `werkzeug.utils`

library. Then save the file in the destination folder configured by `app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER`.

- `upload_database()`: It performs exactly the same operations as `upload_file()`. Then save the file in the destination folder configured by `app.config['UPLOAD_FOLDER_SRC'] = UPLOAD_FOLDER_SRC`.
- `refinement()`: First check that there is at least one file containing high-level policies inside the corresponding folder. It then prints the modification date and time of the last file added or modified to the folder for both the file containing the high-level policies and the `company_database.py` file. These checks are carried out using the `time`, `platform` and `glob` libraries and the `modification_time()` auxiliary function. Once these checks have been made, it launches the execution of `refinemt.py`.
- `refinement_no_gui()`: It performs the exact same operations of the `refinement()` function, but execute `refinement_no_gui.py` instead of `refinement.py`.
- `converter()`: Execute `converter.py` and return the list of files created in the RuleInstance folder in `json` format.
- `download(filename)`: It receives in input the name of a file contained in the RuleInstance folder and sends the corresponding file to the user who has run this API.

B.4 Implement new, or modify existing, functionality

This section will present indications on how to add or modify particular features in the proposed framework.

B.4.1 Support for new high-level policies

The management of high-level policies is done by the Refinement tool. In particular, the components to be modified to implement new high-level

policies, or modify existing ones, are all those indicated in Figure B.1. The changes will be indicated for the case of refinement with GUI, since it is easy to deduce from this the differences to be made for the case without GUI.

The main files to be modified to support new policies are: `rules.py`, `company_database.py` and, secondly, also `info_database.py`.

The `refinement.py` file that regulates the operation of everything, however, does not require many changes as long as the policy to be added is not of a completely different type from those already supported. For example, as can be seen in Section B.1 and in Chapter 7, Refinement supports filtering and protection policies with the distinction between the case called `near_configuration`, where there is no path between a subject and an object, and those where there is a real path.

But let's proceed in order and show the modifications component by component.

CLIPS rules

The first thing to modify to introduce or modify support for new high-level policies is the `rules.py` file which contains all the rules that govern the operation of the tool and the inference engine.

In fact, this file contains all the rules that allow the inference engine to recognize the keywords of the high-level rules, such as "`is not autorized to access`", "`protec confidentiality`", etc. This means that if the specifications of the high-level policies changed, the modification to recognize the appropriate added or modified keyword (that could be the `subject`, `object`, `action`, etc.) must be done within this file, as we can see in Listing B.1.

```
"""
(defrule filtering-association0
    (hspl (action "is authorized to access"))
    =>
        (assert (AcceptActionCapability))
)
"""
```

Listing B.1: Extract of rules.py

If it is necessary to manage the new type of high-level policy with changes also to refinement.py, by adding a new function, it is necessary to create and add a new rule in rules.py that allows the execution of the function to recognize the new keywords, as in the case in Listing B.2.

```
    """
    (defrule object-association
        (not (hspl (object "")))
        (or (hspl (action "is authorized to access"))
            (hspl (action "is not authorized to access"))
            (hspl (action "protect confidentiality"))
            (hspl (action "protect integrity"))
            (hspl (action "protect confidentiality integrity
   "))
        )
        =>
            (object-filter-protection-analysis)
    )
    """
```

Listing B.2: Extract of rules.py

As you can guess, this rule executes the `object-filter-protection-analysis` function, which is mapped (this convention is maintained throughout the file) with the equivalent in `refinement.py` called `object_filter_protection_analysis()`, described in Section A.5, and launches its execution. This function is the one to modify for the management of high-level policy objects.

Company database

For the implementation of new high-level policies it is also very important to modify the `company_database.py` file which contains all the information about the network topology and the devices it contains.

In detail, it contains a series of dictionary with associated the keyword,

that indicates the `subject` and the `object` of the high-level policy, with the corresponding URL or IP. In particular, to add new policies, the `sub_obj_IP`, `subnetIP` and `objectsINFO` dictionaries, shown in Listing B.3, must be changed.

```
sub_obj_IP = {
    'Alice': IPv4Address('10.3.3.24'),
    'Bob': IPv4Address('10.1.1.12'),
    'Subnet1.1': IPv4Network('10.1.1.0/24'),
    'SubnetDMZ': IPv4Network('10.1.2.0/24'),
    'SubnetDecoyDMZ': IPv4Network('10.1.3.0/24'),
    ...
    ...
    'Internet traffic': IPv4Network('192.168.0.0/30'),
    'Web App': IPv4Address('10.3.1.1'),
    'SubnetAway': IPv4Network('172.16.0.0/24')
},
subnetIP = {
    'Subnet1.1': IPv4Network('10.1.1.0/24'),
    'SubnetDMZ': IPv4Network('10.1.2.0/24'),
    'SubnetDecoyDMZ': IPv4Network('10.1.3.0/24'),
    ...
},
objectsINFO = {
    'DNS traffic': ['DestinationType DNS',
    'IpProtocolTypeConditionCapability udp',
    'DestinationPortConditionCapability 53'],
}
```

Listing B.3: Extract of company_database.py

Note that there is the corresponding dictionary containing URL information for each of the existing ones which contains IP information.

Info database

This database is consulted when in the company database no information about the `object` of the high-level policy is found in the `objectsINFO` dictionary. It contains information on previous knowledge that the inference engine could not autonomously extrapolate, such as the port associated with a relative service, or the protocol used.

```
INFO_obj = {
    'VoIP traffic': ['DestinationType VoIP',
    'IpProtocolTypeConditionCapability udp',
    'DestinationPortConditionCapability 5060'],

    'DNS traffic': ['DestinationType DNS',
    'IpProtocolTypeConditionCapability tcp,udp',
    'DestinationPortConditionCapability 53'],

    'Intranet traffic': ['DestinationType Intranet'],
    'All traffic': ['DestinationType All']
}
```

Listing B.4: Extract of info_database.py

Also in this case, as in the case of the company database, the keywords present in the dictionary, shown in Listing B.4, must be updated.

B.4.2 Change Refinement default behaviors

In this section the procedures to modify the default behaviors of the Refinement tool will be presented, possible ideas for future implementation solutions will also be indicated.

The default behaviors of the Refinement process mainly concern choices made during the configuration of the devices, or during the search for a path between a source and a destination, or when an element is not found in the database. The functions in which these choices are present and the ways in which these choices are made will be presented and, subsequently, ideas will be provided for the modification or implementation of new functionalities. Note that all the functions that will be discussed are contained in the `refinement.py` and `refinement_no_gui.py` files.

The first important thing to note is that there is a global string (`IP_address_NSF_catalogue = "localhost"`) to change the address where the NSF-Catalogue is located, remember that this service is necessary to obtain information about the *Security Capabilities* supported by the various *NSFs*. The value is set to `localhost` since the tests during the thesis have always been carried out by starting the NSF-Catalogue on the same machine

as the proposed framework. But it is assumed, in a real development environment, that the NSF-Catalogue could be provided by a different machine than the one that provides the Orchestrator Flask server.

Device configuration

The selection of the devices to be configured along the way, as mentioned several times, changes depending on whether the Refinement execution with GUI support has been launched or if the Refinement has been performed without GUI and it is performed in the `device_configuration_selection(path, source, destination)` function. In the first case the choice is left to the user, in the second case the choice is made by the tool. In particular, the tool performs the choice randomly as you can see in Listing B.5.

```
if len(devices) > 1:
    ...
    device = random.choice(devices)
else:
    device = devices[0]
```

Listing B.5: Extract of refinement_no_gui.py

There are countless ways to change this behavior when there is more than one configurable device on the path between the source and the destination. A possibility is to create a system capable of memorizing the number of policies configured on each device so as to be able to choose from time to time the device with the fewest rules configured on it among those available for that specific path. Obviously there are various variants of this solution, the simplest of all is to consider only the number of policies configured on each device but other various factors could be considered in the creation of such a system, such as, for example, the capacity of the device itself, or even the complexity of the policies configured on it.

Path search

During the selection of the path, and therefore in the identification of the `subject` and the `object` of the high-level policy, there are some default

behaviors. In particular, three main ones can be identified and the first is found within the `database_subject_search(sub, exclude)` function.

```
if sub in sub_obj_IP:
    env.assert_string('(IPSource '+str(sub_obj_IP[sub])+')')
elif sub in sub_obj_URL:
    env.assert_string('(URLSource '+sub_obj_URL[sub]+')')
else:
    try:
        ipaddress.ip_address(sub)
    except ValueError:
        env.assert_string('(IPSource 192.168.0.0/30)')
    else:
        env.assert_string('(IPSource '+sub+')')
```

Listing B.6: Extract of refinement.py

As you can see in the Listing B.6, in the event that the information on the subject is not found either among the information on the IPs or among the information on the URLs, the tool first tries to understand if the entered subject is an IP address using the command `ipaddress.ip_address(sub)` so as to be able to insert it directly in the knowledge base. Then, if this command fails, it inserts Internet as source into the CLIPS knowledge base.

Other default behaviors can occur when searching for the path between the source and destination. In particular we can find them in the `start_path_search()` and `path_search(source, destination)` functions.

As for the first, the behavior is similar to what we have seen above. As we can see in the Figure B.7, the function searches the CLIPS knowledge base for information on sources and destinations and if it does not find any source or destination, it adds the default route to the Internet.

```
for fact in env.facts():
    if "IPSource" == fact.template.name:
        sources.append(fact[0])
        continue
    elif "DestinationType" == fact.template.name:
        near_configuration = True
        destinations.append(fact[0])
    elif "IPDestination" == fact.template.name:
```

```
        destinations.append(fact[0])
        continue
if not destinations:
    # Default route is towards Internet
    destinations.append('192.168.0.0/30')
if not sources:
    # Default route is towards Internet
    sources.append('192.168.0.0/30')
```

Listing B.7: Extract of refinement.py

The `path_search (source, destination)` function, on the other hand, as we can see in Listing B.8, tries to find the shortest path that connects source and destination. At this point in the execution of the framework, as this function receives a `source` and a `destination` as input, it means that the information has been obtained from the database or, as seen in the case above, entered by default. If a path cannot be found with the `nx.shortest_path(graph, source_ip, destination_ip)` function, it can be assumed that the source IP address was not found in the network topology graph.

```
path = []
try:
    path = nx.shortest_path(graph, source_ip, destination_ip)
except nx.NodeNotFound:
    for key, value in sub_obj_IP.items():
        if "Internet" in key:
            source_ip = value
            try:
                path = nx.shortest_path(graph, source_ip,
                destination_ip)
            except nx.NodeNotFound:
                print('Destination is not present in the
                network topology graph.')
                env.assert_string('(Error)')
                return 1
            break

path = path[1:-1]
path.insert(0, source)
path.append(destination)
```

Listing B.8: Extract of refinement.py

The tool then finds the IP address to which the Internet is configured in the database and replaces it at the source. In this way, the shortest path between from Internet to the destination is found and, subsequently, the source is replaced with the original one. In this way, the medium-level configurations generated by the tool will take into consideration the IP address entered and contained in the database as a value to be added, but the path will be the one to the Internet as it is the only plausible one, given that the IP address has not been found in the network topology graph.

Note that the case in which the destination is not present in the network topology graph is also checked, but this case cannot be remedied since if not present in the topology graph it is only possible to assume an error in the data inserted in the high-level policy or in the database.

Bibliography

[1] Andrea Avallone. *A formal model of security controls implementing the IPsec and IKE protocols*. URL: https://webthesis.biblio.polito.it/20399/.

[2] Aurelio Cirella. *An abstract model of NSF capabilities for the automated security management in Software Networks*.

[3] ISO/IEC JTC 1. *Information technology — Open Distributed Processing — Unified Modeling Language (UML)*. en. Standard ISO/IEC 19501:2005. Geneva, CH: International Organization for Standardization, 2005. URL: https://www.iso.org/standard/32620.html.

[4] GeeksforGeeks. *Unified Modeling Language (UML) | An Introduction*. 2019. URL: https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction.

[5] World Wide Web Consortium. *W3C XML Schema Definition Language (XSD) 1.1*. 2012. URL: https://www.w3.org/TR/xmlschema11-1/.

[6] Wikipedia. *CLIPS*. 2022. URL: https://en.wikipedia.org/wiki/CLIPS.

[7] United States. National Aeronautics and Space Administration. *Technology 2001: Conference Proceedings : the Second National Technology Transfer Conference and Exposition, December 3-5, 1991, San Jose Convention Center, San Jose, CA*. NASA conference publication. National Aeronautics and Space Administration, 1991. URL: https://books.google.it/books?id=bcFULyzDoHAC.

[8] Charles L. Forgy. «Rete: A fast algorithm for the many pattern/many object pattern match problem». In: *Artificial Intelligence* 19.1 (1982), pp. 17–37. ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(82)90020-0. URL: https://www.sciencedirect.com/science/article/pii/0004370282900200.

[9] Open Networking Foundation. 2022. URL: https://opennetworking.org/.

[10] Gunjan P Tank, Anmol Dixit, Alekhya Vellanki, and D. Annapurna. «Software-Defined Networking: The New Norm for Networks». In: Apr. 2012.

[11] Raouf Boutaba and Issam Aib. «Policy-based Management: A Historical Perspective». In: *Journal of Network and Systems Management* 15 (2007), pp. 447–480.

[12] Dinesh C. Verma. «Simplifying network administration using policy-based management». In: *IEEE Netw.* 16 (2002), pp. 20–26.

[13] Susan Hares, Diego Lopez, Myo Zarny, Christian Jacquenet, Rakesh Kumar, and Jaehoon (Paul) Jeong. *Interface to Network Security Functions (I2NSF): Problem Statement and Use Cases*. RFC 8192. July 2017. DOI: 10.17487/RFC8192. URL: https://www.rfc-editor.org/info/rfc8192.

[14] Diego Lopez, Edward Lopez, Linda Dunbar, John Strassner, and Rakesh Kumar. *Framework for Interface to Network Security Functions*. RFC 8329. Feb. 2018. DOI: 10.17487/RFC8329. URL: https://www.rfc-editor.org/info/rfc8329.

[15] Jaehoon (Paul) Jeong, Sangwon Hyun, Tae-Jin Ahn, Susan Hares, and Diego Lopez. *Applicability of Interfaces to Network Security Functions to Network-Based Security Services*. Internet-Draft draft-ietf-i2nsf-applicability-18. Work in Progress. Internet Engineering Task Force, Sept. 2019. 29 pp. URL: https://datatracker.ietf.org/doc/html/draft-ietf-i2nsf-applicability-18.

[16] Liang Xia, John Strassner, Cataldo Basile, and Diego Lopez. *Information Model of NSFs Capabilities*. Internet-Draft draft-ietf-i2nsf-capability-05. Work in Progress. Internet Engineering Task Force, Apr. 2019. 26 pp. URL: https://datatracker.ietf.org/doc/html/draft-ietf-i2nsf-capability-05.