Politecnico di Torino

Master of Science in Cybersecurity

Master Degree Thesis

Docker-based deployment for an optimized security mitigation mechanism

Supervisors
prof. Fulvio Valenza
prof. Riccardo Sisto
prof. Daniele Bringhenti
dott. Francesco Pizzato

Candidate
Paolo Tiecco

Academic Year 2024-2025

Summary

In complex network environments, key roles such as network designers and security managers must collaborate to ensure both functionality and protection. However, miscommunication and human error are common and can result in unintended security vulnerabilities which are potential entry points for cyberattacks, in fact according to the 2022 Data Breach Investigations Report (DBIR), approximately 82% of security breaches involved a human element.

To mitigate these issues, automated systems that leverage formal models and ensure formal correctness of results are essential; furthermore, such systems may also improve the reliability of network configurations while optimizing time and resource utilization. This is the context in which VEREFOO (VErified REFinement and Optimized Orchestrator) operates. The framework is specifically designed to automate the complex task of configuring packet filtering firewalls in virtualized networks by transforming high-level Network Security Requirements (NSRs) into optimized and formally verified firewall configurations. It does so by modeling the configuration as a MaxSMT problem instance, combining constraint solving techniques with formal verification to ensure correctness and efficiency. Additionally, in VEREFOO it is possible to integrate Intrusion Detection Systems (IDSs) to detect malicious activity and dynamically react to cyberattacks. Upon detection, the framework triggers an automatic reconfiguration process that updates only the necessary elements of the network topology to contain the threat, thereby minimizing disruption and avoiding full redeployment which is a time-consuming task.

This thesis focuses on the design and implementation of a demonstrator aimed at highlighting the efficiency and responsiveness of the VEREFOO framework within a realistic network context. To this end, a custom network topology was developed to closely resemble an enterprise environment, consisting of multiple subnets representing, on the one hand, a data center and, on the other, departmental networks of a generic company. On this topology, a Denial-of-Service attack—specifically an ICMP flooding—was simulated in order to evaluate the framework's capacity to promptly react to malicious activities. To enable the demonstrator, several modifications and extensions to the framework were carried out. A custom intrusion detection rule for Snort was designed, configured to generate alerts whenever more than ten ICMP echo requests per second are sent by the same source. This ensured that normal administrative operations such as simple pinging remain unaffected, while preventing their misuse for flooding attacks. Moreover, a graphical interface was developed for the demonstrator to guide the user through each step of the process, displaying explanatory messages and offering insights into both its functionality and customization possibilities.

Finally, an additional supporting Python script was created to compare the virtual topology before and after the attack, thereby illustrating the automatic reconfigurations applied by the framework and assessing their effectiveness.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my academic supervisor Prof. Valenza, Prof. Sisto and particulary Prof. Bringhenti and Dott. Pizzato for their continuous guidance, insightful feedback, and invaluable support throughout the development of this thesis. Their expertise and encouragement have been essential in shaping both the research and my understanding of the field.

I would also like to extend my heartfelt thanks to my family for their unwavering love, patience, and constant encouragement because their support has been a source of strength and motivation during the entire course of my academic journey.

Finally, I wish to thank my friends, who have been by my side through every challenge and success which thanks to their friendship, humor, and understanding, have made this experience not only rewarding but also deeply enjoyable.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Thesis Introduction

Modern networks have become increasingly complex, and the demand for flexibility, security, and adaptability is becoming essential in the current cybersecurity landscape. In this context, Software-Defined Networking (SDN) and Network Function Virtualization (NFV) have revolutionized the way networks are designed, managed, and secured. SDN separates the control plane from the data plane by centralizing the control tasks in a software-based SDN controller which communicates with all the network devices via APIs. Centralization enables the programmability of the network and simplifies the configuration and management of large and dynamic networks. NFV moves traditional network functions like firewalls, load balancers, IDSs, etc., from dedicated hardware into virtual machines or containers running on commodity servers. Together, SDN and NFV work synergistically, with SDN providing centralized control over networks and NFV offering the ability to deploy and run network services more flexibly and efficiently [1]. Although they have been proven to be significant in enhancing and speeding up network management operations such as provisioning of service function chain, at the same time, there are side effects such as increased network size and complexity. Those side effects can lead to vulnerabilities that can be exploited by attackers to deliver different types of attack, such as distributed denial of service (DDoS) attacks which are still trending attacks [2] [3]. This underlines the necessity to have some defensive mechanism that can be suitable for virtual networks, whose output can be used to achieve greater resilience against cyber attacks; therefore, network security is a central aspect in this context.

Intrusion Detection Systems (IDS) are vital components in cybersecurity due to the fact that they can continuously monitor the network traffic and generate alerts when something suspicious or malicious crosses their path. In general, an Intrusion Detection System (IDS), is a software application or device that monitors the system or activities of network for policy violations or malicious activities and generates alerts to the management system. A number of systems may try to prevent an intrusion attempt but this is neither required nor expected of a monitoring system. The main focus of Intrusion detection and prevention systems (IDPS) is to identify the possible incidents, logging information about them and report attempts [4].

In the context of virtual infrastructures, IDSs play a fundamental role by continuously monitoring network traffic, this highlights the necessity for an infrastructure that can react quickly and efficiently to the alerts raised by such systems. The main objective of this research is to integrate IDS-generated alerts into a virtualized network environment based on SDN and NFVs, in a way that supports real-time response and automated adaptation. In particular, this thesis focuses on the enhancement of the Verefoo and React-Verefoo tools. React-Verefoo is currently the only solution found in the literature that is capable of achieving real-time monitoring and automated reconfiguration of NFVs within a virtual network. To this end, the goal is to build a python demonstrator that highlights the potential of React-Verefoo in responding to cyberattacks, by implementing a complete and modular workflow involving multiple algorithms and components. This workflow allows for the detection of intrusions, the generation of new security requirements, and the automatic re-deployment of an updated and secure virtual network configuration in which only the necessary nodes are changed or modified. The demonstrator underlines how the system can efficiently achieve the proposed goals and, most importantly, how it can adapt to different network topologies, demonstrating a high degree of flexibility and applicability across various scenario which could also be realistic scenarios [5] [6] [7].

1.2 Thesis description

The approach is structured as follows:

- **Network Topology:** The first part is the creation of a Network topology. This is done by creating an XML file called DemoTopology.xml which is composed by different nodes that can be found in a real network topology like loadbalancer, traffic monitors, webserver, etc. Along with the network topology, network security requirements (NSRs) are created in terms of Isolation and Reachability requirements to express which traffic is permitted or not.
- **Output Configuration:** This is the firts output that VEREFOO tool creates. It takes the Network Topology along with the NSR to produce an XML file called OutputConfiguration.xml which, thanks to a Z3 solver, it creates a topology which will have the definition of firewalls, along with their configuration, in one of the Allocation Places (AP) defined in the DemoTopology.xml.
- **Virtual Network creation:** VEREFOO will produce the virtual network by using the OutputConfiguration by creating docker containers in which, in each of them, nodes defined in the Topology will run.
- **Attack Simulation:** A DoS attack (ICMP flood) will be used to trigger the reaction of the system.
- **Extraction and merging of NSRs:** The configured IDS will generate an alert after the attack and the Sentinel Policy Extractor (SPE) will produce a

new NSR to isolate the attacker and stop the attack. This NSR will be merged with the previous defined NSR by using the Conflicting Policy Merger (CPM) to have NSR without any conflict.

- **Output Configuration new:** React-VEREFOO will be used to update the virtual network to reflect the changes made by the CPM. A new XML file called OutputConfiguration_new.xml is created in which firewalls will be configured again and maybe new or less firewall will be deployed to react to the attack. Moreover, only nodes affected by the new Isolation or reachability property will be changed.
- **Attack Simulation:** Once again the same attack will be done to check if the update stopped correctly the attack.

Furthermore, in a real case scenario in which this framework is used, a looping process is adopted because after the virtual network creation, the looping part starts from the Attack Simulation in which changes to the topology happens only as a reaction to attacks or changes into the topology or NSRs.

The demonstration was done by using Snort 3 as IDS and Docker to deploy the virtual network. This thesis has some limitation:

- **IDS:** Currently the research has been tested only with Snort 3 as Intrusion Detection system, but the framework suits well for adding new Intrusion Detection Systems.
- **Firewall:** The tests has been done by using only Iptables but once again the tool can be enriched with new type of firewall quite easily
- **Conflicting Policy Merger algorithm:** This algorithm is still in its developmental stages. Its operation might not yet be at peak efficiency, and it requires further validation. Moreover, the algorithm currently makes specific assumptions about the network and the NSRs which may not be universally applicable.

The chapters are organized as follows:

- **Chapter 2: Background and related works** This chapter provides the foundational knowledge necessary to understand the context and technical choices behind this research. It begins with an overview of Intrusion Detection Systems (IDSs), explaining their role in network security, their operational strategies (active vs. passive), and their deployment models (host-based and network-based). The chapter then focuses on the specific IDS used in this research, Snort 3, detailing its architecture, rule-based detection mechanism, and real-time monitoring capabilities and on how Snort rules are structured and how they were tailored in this work to detect ICMP flood attacks effectively. Finally, the chapter highlights how IDS alerts are used within the VEREFOO framework to trigger automatic reconfiguration and enforce network modifications in reaction to detected threats.

- **Chapter 3: Network Security Configuration Automation** This chapter presents the theoretical and practical foundations of VEREFOO, the core framework of this research. It begins by discussing the evolution of network configuration through technologies like NFV and SDN, highlighting the complexity introduced by the need to align network topologies with security requirements (NSRs). The chapter then introduces VEREFOO and explains how VEREFOO works through key components and inputs such as the Service Graph (SG), Allocation Graph (AG), the Z3 MaxSMT solver and how it ensures formal correctness in the computed configurations. Next, the chapter explores modules that extend VEREFOO's capabilities, including:

    - The Virtual Network Translator, which transforms abstract configurations into deployable Docker-based environments.
    - The Sentinel Policy Extractor (SPE) and Conflicting Policy Merger (CPM), which process IDS alerts into new NSRs and resolve policy conflicts.
    - React-VEREFOO, which enables fast and localized updates to firewall rules without redeployment of the whole network.

  Finally, this chapter also outlines the adopted workflow of the demonstrator built in this thesis, where an initial deployment is refined over time in response to cyberattacks.

- **Chapter 4: Thesis Overview** This chapter outlines the main objectives of the thesis and the methodology followed to achieve them. The primary goal was the development of a comprehensive demonstrator to showcase the capabilities of the VEREFOO framework, particularly its reactivity to network attacks. To this end, a realistic network topology and a heterogeneous set of Network Security Requirements (NSRs) were designed to simulate a real-world scenario and, to do so, different modifications were made to the framework to enable seamless processing of inputs and automatic deployment of the virtual network. Furthermore, at the end, a realistic ICMP flood attack scenario was also modeled to trigger the framework's reaction mechanisms. The chapter details the implementation of the demonstrator, which simulates the entire process, from initial setup to network reconfiguration in response to the detected attack. An additional script was developed to visually compare the virtual topology before and after the attack, thus illustrating the effectiveness of the automatic reconfiguration.

- **Chapter 5: Approach** This chapter provides a detailed explanation of the demonstrator developed to validate the framework diving in also in the created topology. It begins by presenting the custom network topology designed to represent a real-world enterprise environment, including client and the different subnets. The simulated cyberattack, an ICMP flood targeting a specific server in the topology, is then introduced. Furthermore, the chapter outlines the full workflow of the demonstrator, from the initial deployment of the virtual network based on the input topology and NSRs, to the detection of the attack using Snort 3, the subsequent extraction and transformation of the alert into a new Network Security Requirement, the automatic merging of the new requirement with existing ones and the reconfiguration of the virtual network through React-VEREFOO. Throughout the chapter, visual evidence such as firewall rule changes and traffic behavior are used to demonstrate how the framework detects, responds to, and mitigates the attack effectively and with minimal human intervention.

- **Chapter 6: Conclusions & future works** This chapter summarizes the key achievements of the thesis, emphasizing the successful development of a demonstrator and the refinement of the VEREFOO framework. It highlights how the demonstrator provides a complete, end-to-end view of the framework's capabilities, showcasing its ability to detect and react to cyber-attacks, specifically an ICMP flood, through dynamic firewall reconfiguration and network adaptation with minimal updates. The chapter also reflects on the technical challenges encountered, such as modifying the latest version of the tool to ensure full automation and building a realistic network topology and attack scenario. Finally, the chapter outlines potential areas for future development, including extending support to other IDSs, integrating alternative firewall technologies, improving protocol support, and testing scalability in larger network environments and also opportunities for further automation using machine learning techniques for alert classification.

Chapter 2

Intrusion Detection Solutions, Docker & Network Security Configuration Automation

2.1 Intrusion Detection System

A series of security confidentiality, and privacy issues have been associated with the use of computer system and the Internet in recent times due to the need to transfer data electronically. Hence, studies have been focusing on the improvement of privacy and secure compute system, but despite the effort, these problems still linger in computer systems. Also, attacks are launched in different forms, with abnormal behavior in the signature database, with attack vectors that can vary in a few seconds during the attack, etc. Therefore, several tools have been used to counter different forms of attack on network systems, and one such tool is Intrusion Detection Systems (IDSs), which was created for the real-time monitoring of network systems for any form of intrusion [8].

Intrusion Detection Systems are systems to identify actors using a computer or a network without authorization but it can also be extendable to identify authorized actors violating their privileges, which is based on the hypothesis that the behavioral pattern of non-authorized users differs from the authorized ones. This monitoring is performed on a rule-based strategy which means that there are rules built upon known patterns, signatures, or behaviors of potential cyber threats. Each rule helps the IDS identify certain types of harmful activities or unusual patterns in network traffic, offering a clear and organized way for the system to monitor the data passing through the network. When incoming traffic matches one of these rules, IDSs can perform various action, in particular, they can generate an alert, indicating a potential security breach or malicious activity which usually is analyzed by a human being to see if that alert is a true or false positive. In general an IDS can base its behaviour on two strategies:

- **Passive IDS:** They try to identify signs of an attack using cryptographic checksum and pattern matching, which is the attack signature, so looking for specific packets which are typical of an attack.

- **Active IDS:** Those types of IDS try to identify unknown attacks or known attacks, but before producing the negative result they first perform a statistical analysis of the system behavior, collect actively statistical information related to traffic and compare that information against statistical parameters and the action is triggered when a threshold is exceeded [9].

Typically and IDS have both active and passive part.

IDS can be also distinguished in two topological features:

- **HIDS (host-based IDS):** They are deployed inside hosts and check for unauthorized changes to files or filesystem, checks log files of OS or application looking for known patterns of succesfull attacks or attempts.
- **NIDS (Network-based IDS):** They are deployed inside a network and they are based on three elements which are:
  - **Sensor:** Check traffic and logs looking for suspicious patterns, generate security events and can interact with the system by doing, for instance, TCP reset or by modifying Access Control Lists.
  - **Director:** It coordinates the work of all sensors and manages a database which can contain statistics, attack signatures, etc.
  - **IDS message system:** Provides an authenticated, integral and reliable communication among the IDS components. Usually, the reliability of the communication is achieved by using a separate physical network or with VLAN or VPN.

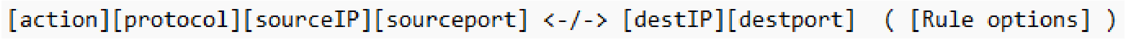
Lastly, there are Intrusion Prevention Systems (IPS) which are IDSs paired with distributed dynamic firewall used to speed-up and automate the reaction to intrusions. The IDS sends alarms related to a certain type of traffic, then the distributed dynamic firewall, when receives an alarm, can block that traffic or isolate a node.

Intrusion Detection Systems are fundamental for this research because their alerts will be used to trigger the reaction of React-Verefoo to update the virtual network in response to attacks. In particular, the alert generated by the IDS will be used to create a new NSR that will be merged with the existing ones to be able to effectively isolate the node that started the attack. The alert is converted into an Isolation Property by the Sentinel Policy Extractor (SPE), which extracts from the alert the IP 5-tuple (IP source, IP destination, source port, destination port and level 4 protocol).

The IDS used in this research is Snort 3 which is an open-source tool that uses different set of rules or custom rules that help define malicious network activity and uses those rules to find packets that match against them and generates alerts for users. Snort can be deployed inline to stop these packets, as well. Snort has three primary uses: As a packet sniffer like tcpdump, as a packet logger, which is useful for network traffic debugging, or it can be used as a full-blown network intrusion prevention system [10].

2.1.1 Snort 3

Originating in the late 1990s, Snort has consistently been at the forefront of intrusion detection and prevention solutions. Over the decades, it has undergone significant development, with Snort 3 being its most advanced version. Designed with versatility in mind, Snort 3 is equipped to handle complex network architectures and traffic patterns. Snort is an open source network intrusion prevention system (IPS) by Cisco. It is capable of performing real-time traffic analysis and packet logging on IP networks. It can perform protocol analysis, content searching and matching, and detect a variety of attacks and probes.



```
[action][protocol][sourceIP][sourceport] <-/-> [destIP][destport]  ( [Rule options] )
```

Figure 2.1. Basic Outline of a Snort Rule

Snort 3 is a rule-driven architecture which means that his central elements are rules. In general Snort's rule are composed by:

- **Action:** The rule action tells Snort what to do then it finds a packet that matches the rule criteria, in this research the action will be alert but they could be also drop or block.
- **Protocol:** Currently Snort analyzes for suspicious behaviour four protocols which are TCP, UDP, ICMP and IP.
- **sourceIP:** It can be a specific IP address or a generic external network which can be specified with *$EXTERNAL_NET*. This variable represent any external network that usually are untrusted.
- **sourceport:** It can be a specific source port or a generic source port identified with *any* or *$HTTP_PORTS*.
- **Direction operator:** This operator indicates the orientation of the traffic to which the rule applies.
- **destIP:** It can be a specific IP address or a generic internal network which can be specified with *$HOME_NET*. This variable represent the trusted network, the one that will be protected.
- **destport:** It can be a specific destination port or a generic source port identified with *any* or *$HTTP_PORTS*.

There are also various rule options available in Snort that significantly enhance its ability to detect specific types of malicious traffic. These options allow for fine-grained control over what constitutes suspicious behavior, making Snort both efficient and highly adaptable. For example, the rule used in this research is configured to generate an alert only when more than 10 ICMP echo requests per second are sent by the same source IP address. This ensures that normal network behavior, such as occasional ping requests, does not trigger alarms, while more aggressive patterns, like ICMP flood attacks, are accurately detected. This demonstrates how Snort's flexible rule structure can be tailored to respond effectively to a wide range of real-world scenarios.

In this research Snort 3 is used as Network Intrusion Detection System (NIDS) which means that in the defined network topology there will be a specific node called TRAFFIC_MONITOR that will be deployed inside a docker container when the network will be virtually deployed.

2.2 Docker

Docker is an open-source platform that enables developers to build, deploy, run, update and manage containers. A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings. Container images become containers at runtime and in the case of Docker containers, images become containers when they run on Docker Engine. Available for both Linux and Windows-based applications, containerized software will always run the same, regardless of the infrastructure. Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging. Docker containers that run on Docker Engine:

- **Standard:** Docker created the industry standard for containers, so they could be portable anywhere.
- **Lightweight:** Containers share the machine's OS system kernel and therefore do not require an OS per application, driving higher server efficiencies and reducing server and licensing costs.
- **Secure:** Applications are safer in containers and Docker provides the strongest default isolation capabilities in the industry [11].

Docker significantly enhances cybersecurity through several key mechanisms. Firstly, by default, Docker containers operate with a restricted set of Linux kernel capabilities, implementing a fine-grained privilege management system that reduces the risk of privilege escalation, even if a process runs as root within the container, its permissions on the host remain limited and secondly, Docker offers Hardened Images (DHI), purposefully slim, secure, and minimal base images that omit unnecessary components like shells, package managers, and compilers to minimize the attack surface, with some images reducing exposure by up to 95%. These images enforce non-root execution and are continuously updated to maintain near-zero known vulnerabilities, substantially improving system resilience.

While Docker offers compelling benefits for virtualization efficiency and rapid deployment, it also introduces specific security challenges that must be carefully managed. One primary aspect is related on the provenance of images because unverified or malicious images that comes from public repositories may include malware or unpatched vulnerabilities that can lead to security breaches. Moreover, the isolation between containers and the host system is not absolute since Contaners and host share the same kernel so, a vulnerability exploited can lead to privilege escalation or compromise the host environment and, misconfiguration related, for instance, on running containers with excessive capabilities further exacerbates the risk [12].

In this research Docker is used to deploy a virtual network in which each node of the network is running in a Docker container. This is done by providing to VEREFOO framework the XML file, that describes the topology of the network composed by different nodes like webservers, webclient, firewall and so on, in order to obtain the necessary files like docker-compose and configuration scripts for deploying the virtual network. This selected approach comes from the fact that Docker is fast, use resources in an efficient way and can be used to introduce more automation in the approach. Further detail will be provided afterwards in this research, but, as an example, this research is composed by two complementary strands of recent work that emphasize automation for both correctness and performance. In industrial control systems, the shift to virtualized networks has shown that treating Virtual Network Function (VNF) embedding and time-sensitive flow scheduling as a single, jointly optimized problem, for instance via Optimization Satisfiability Modulo Theories, can provide formal guarantees of solution correctness while explicitly minimizing end-to-end delay. [13]. In parallel, within multi-domain Kubernetes deployments, the automatic derivation of per-cluster network security policies reduces administrator-induced inconsistencies and enables transparent cross-cluster communication, improving isolation, availability, and scalability; a prototype called *Multi-Cluster Orchestrator* has validated these benefits on realistic use cases [14]. Drawing on these insights, this research uses Docker in the pipeline to enchance performance, to lower configuration error rates, and to scale across heterogeneous administrative domains, providing a robust substrate for VEREFOO's automated, policy-driven reactions to network threats.

2.3 Network Security Configuration Automation

2.3.1 Overview

Network Function Virtualization (NFV) and Software Defined Networking (SDN) made the configuration of networks much more agile, allowing the service function graph to be created more dynamically. One of the most known Service Function Graph is the Service Graph (SG) which is the logic representation of a virtual network which does not depend from the underneath physical infrastructure. The problem lies in the Network Security Requirements (NSR) which must be satisfied in the SG and usually those NSR are handled by the security manager while the network manager manages the network topology and, miscommunication may lead to errors in the implementation of the security controls. Moreover, it is a slow manual approach that does not use resources efficiently and the obtained solution may contain vulnerabilities or may be prone to attacks. This concept is often addressed in the literature, for instance, in [15], the authors highlight how the evolution of modern, large-scale, and heterogeneous networks have made firewall policy management increasingly complex. Traditional manual configurations are prone to errors, inefficiencies, and policy anomalies such as sub-optimal rules and conflicts. Existing anomaly management techniques frequently suffer from excessive complexity and incomplete resolution capabilities. To address these problems, the paper proposes a comprehensive approach for firewall policy anomaly analysis and resolution, leveraging the formal concept of atomic predicates which objective is to streamline anomaly detection, improve efficiency, and ensure complete resolution of configuration conflicts. The feasibility and performance of the approach have been validated through a dedicated framework, demonstrating superior anomaly handling and time efficiency compared to alternative solutions.

Never the less, the Data Breach Investigations Report (DBIR) states in the report of 2022, that the human element continues to drive breaches. This year 82% of the breaches involved the human element, and overall there is a continuous growth of different types of attacks [16].

One of the most used NSF is the firewall that follows three fundamental principles [17]:

- The firewall should be the only point of contact between internal and external network
- Only the authorized traffic can traverse the Firewall
- The firewall must be a highly secure system itself

At this time, thanks to SDN and NFV, firewalls have a much more distributed approach in which multiple instances of firewalls can be allocated. Therefore and for the problems cited above, it is fundamental to find an automatic way to allocate packet filter inside an SG defined a priori by a service designer and being able to create firewall rules that can automatically satisfy the NSR with formal assurance of the solution [7].

2.3.2 VEREFOO

Overview

In this context, VEREFOO emerges as a framework specifically crafted to function within these modern networking paradigms such as NFV, SDN, NSF, Docker, etc. VEREFOO aims to refine high-level NSRs, strategically allocate and configure selected NSFs, and enable the placement of each virtual function of the SG on dedicated servers. Other than that, the solution found by VEREFOO is guaranteed to be optimal and formally correct thanks to the Z3 solver which solves a problem framed internally by VEREFOO as a partial weighted Maximum Satisfiability Modulo Theories (MaxSMT) problem. Moreover, to dive deeper in this concept, this paper proposes the demonstration of a novel security framework based on an optimized approach for the automatic orchestration of virtual distributed firewalls. VEREFOO provides formal guarantees for the firewall configuration correctness and minimizes the size of the firewall allocation scheme and rule set [18]. In the following subsection the VEREFOO components, input and output, will be analyzed.

Service Graph

The Service Graph (SG) is one of the two input provided to VEREFOO and it is crucial because it represents the network topology upon which the NSR will be based. In general, an SG is the logical topology of a virtual network and represents how the service functions and nodes are interconnected to form an end-to-end network service. The SG is built by the service designer without considering the security of the topology, because the objective is merely to create a networking service in which every function is called Network Function (NF). Additionally, low level functions like switches and router are represented in an abstract way.

The Network Functions are different, they can represent a web server or a load balancer and the service designer is the one who chooses which NF will be placed and where it will be placed without considering at the deepest level what the function will do.

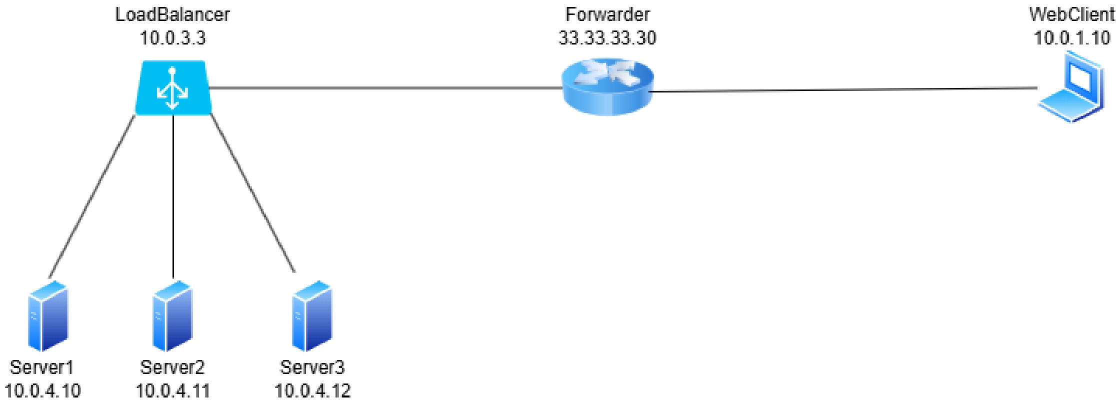


Figure 2.2. Representation of Service Graph

In 2.2, a simple Service Graph is shown, it represents a small modified part of the network topology created for this research. The associated XML file that should define this topology that is accepted by VEREFOO will be the following one:

Listing 2.1. ServiceGraph.xml

```
<node functional_type="WEBCLIENT" name="10.0.1.10">
    <neighbour name="33.33.33.30" />
    <configuration description="e9" name="webclient6">
        <webclient nameWebServer="10.0.4.10" />
    </configuration>
</node>
```

```xml
    <node functional_type="FORWARDER" name="33.33.33.30">
        <neighbour name="10.0.1.10" />
        <neighbour name="1.0.3.3" />
        <configuration name="ForwardConf1">
            <forwarder>
                <name>Forwarder1</name>
            </forwarder>
        </configuration>
    </node>

    <node functional_type="LOADBALANCER" name="10.0.3.3">
        <neighbour name="1.0.0.14" />
        <neighbour name="1.0.0.8" />
        <neighbour name="1.0.0.9" />
        <neighbour name="1.0.0.10" />
        <configuration description="s9" name="loadbalancer">
            <loadbalancer>
                <pool>10.0.4.10</pool>
                <pool>10.0.4.11</pool>
                <pool>10.0.4.12</pool>
            </loadbalancer>
        </configuration>
    </node>

    <node functional_type="WEBSERVER" name="10.0.4.10">
        <neighbour name="1.0.0.8" />
        <configuration description="e2" name="httpserver2">
            <webserver>
                <name>10.0.4.10</name>
            </webserver>
        </configuration>
    </node>

    <node functional_type="WEBSERVER" name="10.0.4.11">
        <neighbour name="1.0.0.9" />
        <configuration description="e3" name="httpserver3">
            <webserver>
                <name>10.0.4.11</name>
            </webserver>
        </configuration>
    </node>

    <node functional_type="WEBSERVER" name="10.0.4.12">
        <neighbour name="1.0.0.10" />
        <configuration description="e4" name="httpserver4">
            <webserver>
                <name>10.0.4.12</name>
            </webserver>
        </configuration>
    </node>
```

The XML file shows the different elements that are common in every file created for this framework. A brief analysis of the file will follow.

- All the XML files of this type start with:

Listing 2.2. Starting definition of SG

```
<graph id="0" serviceGraph="true">
```

This XML tag is composed by an id which is a unique identifier, and a boolean value (serviceGraph) in which a true value means that this XML file will be treated as a Service Graph

- All the network entity are identified with node and a *functional_type* value which indicates the role that the node will cover in the Topology. Different values could be used, in 2.1 the functional types are: WEBCLIENT, WEBSERVER, LOADBALANCER, but there could be more. The unique identifier of each node is name which is the IP address.
- Every node has a neighbour which is represented by the name that once again is the unique attribute of the previous node that indicates the direct connections. These relationships show the interconnections of the network.
- Every node has its own configuration that, for instance, in the case of the LOADBALANCER translates into a pool of server that the load balancer will manage.

Allocation Graph

The Service Graph is then provided to VEREFOO that will transform it into an internal representation called Allocation Graph (AG). The AG is characterized, between each node, by placeholders called Allocation Places (AP) which are the places where a firewall could be allocated. Those AP may be used by a service designer to allocate in advance some firewall that may or not may be changed afterwards by the tool itself. This manual contribution may lead to an increased flexibility and reduction of computation time, but could also lead to an unoptimized solution.

Following, there is a visual representation of an Allocation Graph that, once again, is a small modified part of the demonstration created in this research.

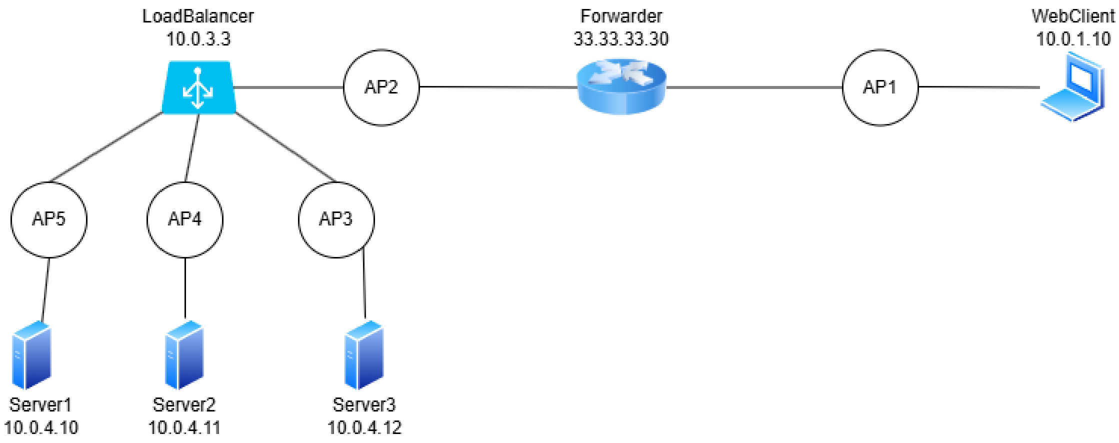


Figure 2.3. Representation of an Allocation Graph

Visually, the only difference between 2.2 and 2.3 is the presence of the allocation places cited above. The XML associated to figure 2.3 is the following one that will be explained afterwards.

Listing 2.3. Allocation Graph.xml

```
<node functional_type="WEBCLIENT" name="10.0.1.10">
    <neighbour name="33.33.33.30" />
    <configuration description="e9" name="webclient6">
        <webclient nameWebServer="10.0.4.10" />
    </configuration>
</node>

<!-- AP1 -->
<node name="1.0.0.1">
    <neighbour name="33.33.33.30" \>
    <neighbour name="10.0.1.10" \>
</node>

<node functional_type="FORWARDER" name="33.33.33.30">
    <neighbour name="10.0.1.10" />
    <neighbour name="1.0.3.3" />
    <configuration name="ForwardConf1">
        <forwarder>
            <name>Forwarder1</name>
        </forwarder>
    </configuration>
</node>

<!-- AP2 -->
<node name="1.0.0.2">
    <neighbour name="33.33.33.30" \>
    <neighbour name="10.0.3.3" \>
</node>

<node functional_type="LOADBALANCER" name="10.0.3.3">
    <neighbour name="1.0.0.14" />
    <neighbour name="1.0.0.8" />
    <neighbour name="1.0.0.9" />
    <neighbour name="1.0.0.10" />
    <configuration description="s9" name="loadbalancer">
        <loadbalancer>
            <pool>10.0.4.10</pool>
            <pool>10.0.4.11</pool>
            <pool>10.0.4.12</pool>
        </loadbalancer>
    </configuration>
</node>

<!-- AP3 -->
<node name="1.0.0.3">
    <neighbour name="10.0.3.3" \>
    <neighbour name="10.0.4.10" \>
</node>

<!-- AP4 -->
<node name="1.0.0.2">
    <neighbour name="10.0.3.3" \>
```

```
        <neighbour name="10.0.4.11" \>
    </node>

    <!-- AP5 -->
    <node name="1.0.0.2">
        <neighbour name="10.0.3.3" \>
        <neighbour name="10.0.4.12" \>
    </node>

    <node functional_type="WEBSERVER" name="10.0.4.10">
        <neighbour name="1.0.0.8" />
        <configuration description="e2" name="httpserver2">
            <webserver>
                <name>10.0.4.10</name>
            </webserver>
        </configuration>
    </node>

    <node functional_type="WEBSERVER" name="10.0.4.11">
        <neighbour name="1.0.0.9" />
        <configuration description="e3" name="httpserver3">
            <webserver>
                <name>10.0.4.11</name>
            </webserver>
        </configuration>
    </node>

    <node functional_type="WEBSERVER" name="10.0.4.12">
        <neighbour name="1.0.0.10" />
        <configuration description="e4" name="httpserver4">
            <webserver>
                <name>10.0.4.12</name>
            </webserver>
        </configuration>
    </node>
```

The XML file of the Allocation Graph is very similar to the Service Graph in 2.1. The only difference is about those Allocation Places that are defined with *node*, their unique identifier is the name which is still an IP address in the 1.0.0.0/24 network and once again, neighbour elements are specified in order to map out the network's interconnections.

Network Security Requirements

The network security requirements are the other input provided to VEREFOO along with the AG. These are properties that express connectivity between endpoints and there are four approaches defined other than the default behaviour.

- **Whitelisting:** All that is not explicitly permitted, is forbidden. This means that the default behaviour is to block the traffic unless there is a reachabilty requirement expressed for that traffic. Usually whitelisting approach have higher security but they are more difficult to manage since permitted traffic must be known in advance.

- **Blacklisting:** All that is not explicitly forbidden, is permitted. This means that the default behaviour is to permit the traffic unless there is an isolation requirement expressed for that traffic. This lead to lower security due to the fact that a traffic is permitted until it becomes a malicious traffic, but they are easier to manage.
- **Rule-oriented specific:** The aim is to minimize the number of rules defined, there is no default behaviour specified, it is possible to express both reachability and isolation requirements.
- **Security-oriented specific:** Only the communications necessary to satisfy user requirements are permitted. Also in this approach no default behaviour is specified because it is possible to express both reachability and isolation requirements.

In the last two approaches, NSRs are assumed to be without any type of anomalies, it is not a restriction because anomalies can be solved easily thanks to known and used techniques. For instance, in this paper [19], the anomalies in firewall policies are solved introducing a formal model which is based on first-order logic rules that analyses the network topology and the security controls at each node to identify the detected anomalies and suggest the strategies to solve them with excellent performance and good scalability.

In every four approaches, the NSRs are defined using a medium-level language in which the IP's 5-tuple is specified [7]. This translates into NSR with the following attributes:

- **ruleType:** The NSR could be an Isolation Requirement or a Reachability Requirement meaning that the traffic will be Isolated or Permitted.
- **IPSrc:** It defines the source IP address, the value -1 can be used to express an entire subnetwork.
- **IPDst:** It defines the destination IP address and the value -1 can be used to express an entire subnetwork.
- **portSrc:** It defines the source port, also here the value -1 can be used to express all the ports. If no portSrc is defined it will be considered as -1 value.
- **portDst:** It defines the destination port, also here the value -1 can be used to express all the ports. If no portSrc is defined it will be considered as -1 value.
- **transportProto:** Here the transport protocol is specified. The values that can be expressed are:
  - **TCP**
  - **UDP**
  - **OTHER:** Any transport protocol excluding TCP and UDP
  - **ANY:** Any transport protocol including TCP and UDP.

In the following figure it is possible to see examples of Isolation and Reachability Requirements. These are also a small part of the NSR defined in the demonstration created in this research.

Listing 2.4. A part of the NSR used in this research

```
<Property graph="0" name="ReachabilityProperty" src="10.0.1.11"
    dst="10.0.4.11" lv4proto="TCP"/>
<Property graph="0" name="ReachabilityProperty" src="10.0.4.11"
    dst="10.0.1.11" lv4proto="TCP"/>
<Property graph="0" name="ReachabilityProperty" src="10.0.1.11"
    dst="10.0.1.10" lv4proto="TCP"/>
<Property graph="0" name="ReachabilityProperty" src="10.0.1.10"
    dst="10.0.1.11" lv4proto="TCP"/>
<Property graph="0" name="ReachabilityProperty" src="10.0.1.11"
    dst="10.0.5.10" lv4proto="OTHER"/>
<Property graph="0" name="ReachabilityProperty" src="10.0.5.10"
    dst="10.0.1.11" lv4proto="TCP"/>

<Property graph="0" name="IsolationProperty" src="10.0.1.11" dst="10.0.4.10"
    lv4proto="TCP"/>
<Property graph="0" name="IsolationProperty" src="10.0.4.10" dst="10.0.1.11"
    lv4proto="TCP"/>
<Property graph="0" name="IsolationProperty" src="10.0.1.11" dst="10.0.4.12"
    lv4proto="TCP"/>
<Property graph="0" name="IsolationProperty" src="10.0.4.12" dst="10.0.1.11"
    lv4proto="TCP"/>
<Property graph="0" name="IsolationProperty" src="10.0.1.11" dst="10.0.5.20"
    lv4proto="TCP"/>
<Property graph="0" name="IsolationProperty" src="10.0.5.20" dst="10.0.1.11"
    lv4proto="TCP"/>
```

As it is possible to see from Figure 2.4, the followed approach is **Rule-oriented specific**, these are Isolation and Reachability Properties defined for a web client in the topology. This client can communicate with TCP and other protocols only with three other nodes in the topology in both direction, for other nodes in the topology there is no permitted communication with TCP and other protocols in both direction. There is also no specification for source and destination ports, this means that they will be treated as the value -1 which indicates that those requirements will be applied for any source or destination ports.

Z3 solver and MaxSMT problem

An SMT solver is a tool for deciding the satisfiability (or dually the validity) of formulas in these theories. SMT solvers enable applications such as extended static checking, predicate abstraction, test case generation, and bounded model checking over infinite domains, to mention a few. Z3 is an SMT solver from Microsoft Research, it is used to solve problems that arise in software verification and software analysis. Consequently, it also integrates support for a variety of theories. A prototype of Z3 participated in SMT-COMP'07, where it won 4 first places, and 7 second places. Z3 uses novel algorithms for quantifier instantiation and theory combination. The first external release of Z3 was in September 2007 [20].

The used approach is having a methodology that permits to compute automatically the reconfiguration of a distributed firewall whenever a user specifies new network security requirements. Furthermore, automation is combined with optimization and formal verification and for this reason formal models are employed. The formal model represents the network, the network function configuration and the exchanged traffic while the resolution of the MaxSMT problem permits to generate an allocation graph and rules used for the configuration of the firewall. MaxSMT problems are different from classic SMT problems because they are characterized by hard constraint that are used to find the solution to the problem along with soft constraints that are optional and employed to reach a greater optimization of the solution. The solution employed will be always the one that satisfies all the hard constraints and that maximize the sum of the soft constraints' weights. In particular, when the algorithm finds the firewall instances that require a reconfiguration these information are used along with the formal models as input for the MaxSMT problem formulation. The preferred solution is always the optimal reconfiguration which is the one that do not require any update in the network because it would cause less delays. For this reasons, soft constraints are employed in order to prefer always updating existing firewalls and at the same time updating existing rules instead of creating new firewalls or creating new rules.

The formal model used is characterized by:

- Allocation Graph.
- All the packets that cross the Allocation Graph are grouped into classes defined by the headers' values. They are represented as predicates characterized by sub-predicates which are values of the IP 5-tuple.
- all the rules generated for the firewalls are modeled with the same predicate of the packets (IP 5-tuple).
- Two behaviour, related on how the AG manages every packet, are considered which are forwarding and transformation. The forwarding function is a function that maps the ingress traffic to true only if the node from which the traffic passes blocks the packets, while the transformation function maps the ingress traffic to the correspondent output traffic considering Identity transformation or transformation that actually changes the IP 5-tuple.
- Network Security Requirements are also considered and modeled as a combination of two elements.
  - Set of Network Security Requirements in which every NSR is formally modeled with the tuple (a,C) in which a is the action while C is the condition predicate
  - A generic behaviour which is don't care that permits to define isolation or reachability requirements without any type of restrictions

All those elements will be modeled in the MaxSMT problem as hard constraints making their satisfaction compulsory [5].

Virtual Network Translator Module

The virtual network translator is a module specifically developed for VEREFOO to convert an Allocation Graph into a functional virtual network. This module takes as input the Allocation Graph and produces the required configuration files to convert the Allocation Graph into a real network topology with containers. To achieve that, the algorithm must be able to extract information from the XML file of the Allocation Graph and translate them into configuration files that are used to start up the entire environment. Since the XML file is an high-level abstraction of the topology, it has not so many information regarding the real topology. For this reason, the developed module finds the crucial information from the XML file and takes them to build the container in which every node will be deployed to set-up a fully working virtual network. By analyzing the previous listing 2.3, all crucial information can be extracted. Each node has an IP address that is defined with the *name* attribute, with the *neighbour* attributes it is possible to map the links between every node defined, the *functional type* attribute is used to identify the functionality that the node must support in the real topology and the *configuration* attribute is important only for some specific nodes like load balancers, NAT, traffic monitors, etc. Furthermore, before creating the virtual network files, the module eliminates any superfluous Allocation Places ensuring that only the necessary network nodes are retained. This creates an optimized environment for the subsequent translation process.

The responsibility of converting the refined output configuration, the Allocation graph with firewalls deployed which is called OutputConfiguration, into actionable files, like Docker Compose configurations, assorted configuration files, and Dockerfiles for various container types, falls to the VnetworkTranslator and IptablesVnetwork classes. Presently, the module is equipped to only generate iptables configuration files, but it can be expanded easily, thanks to his modularity to support additional firewall software options, including EBPF and OpenvSwitch, aiming to increase its versatility and extend its applicability [21] [22].

Sentinel Policy Extractor (SPE) & Conflicting Policy Merger (CPM)

As cited before, one of the node that can be specified in the Topology can be an Intrusion Detection System(IDS) that inspects traffic flowing through their monitored interfaces to search for potential attacks. When a malicious packet is detected, the rule specified in the IDS configuration will be triggered and an allert will be generated. The alert cannot be used in his raw form, for this reason the Sentinel Policy Extractor (SPE), which has read permission on the log files written by the IDS, extracts relevant information and produces a set of NSRs that should correctly stop the attack.

The SPE firstly, whenever a log entry about an attack detection is created, applies an abstract model of the alarm to that specific entry, so as to retrieve the information representing the formal model of the traffic that causes the alert. The extracted information is composed by the IP 5-tuple fields, if no information are specified the wildcard * is used. Secondly, the SPE finds all the traffic flows where the malicious packets appears creating a subset of potentially malicious flows and lastly, for each flow, the SPE extracts a policy in which the source IP address and port value are taken from the initial flow packet class, while the destination fields are taken from the final flow packet class because that flow could cross nodes like NAT that could change some of those values.

It is important to underline that surely IDS could create false positives which is an intrinsic behaviour, but administrator can remedy the problem by monitoring the process, detecting false positives and applying appropriate correction.

Now, policies are extracted from the alert and they should be merged with the original NSR. This is not a simple operation due to the fact that a new policy may have conflict with the existing ones, for example a traffic flow that had to reach its destination according to a reachability property must now be blocked because the newly extracted isolation policy blocks that communication. In this setting the Conflicting Policy Merger (CPM) takes his place and for each policy, it checks if it conflicts with any of the new ones and determine if and how to include them [6]. For instance, in the demonstrator created in this research, there was a Reachability Property that permitted ICMP traffic between the node 198.51.100.11 and the server 10.0.4.12 and another Reachability Property for the opposite direction.

Listing 2.5. NSR Properties before CPM

```
<Property name="ReachabilityProperty" graph="0" src="198.51.100.11"
    dst="10.0.4.12" lv4proto="OTHER" isSat="true"/>
<Property name="ReachabilityProperty" graph="0" src="10.0.4.12"
    dst="198.51.100.11" lv4proto="OTHER" isSat="true"/>
```

The extracted requirement from the IDS is an Isolation Property that stops ICMP communication between the two nodes. This creates a conflict between the existing property and the newly extracted one, for this reason the CPM modifies the original property to block the communication between the two nodes.

Listing 2.6. NSR Properties after CPM

```
<Property name="ReachabilityProperty" graph="0" src="10.0.4.12"
    dst="198.51.100.11" lv4proto="OTHER" isSat="true"/>
<Property name="IsolationProperty" graph="0" src="198.51.100.11"
    dst="10.0.4.12" lv4proto="OTHER" isSat="true"/>
```

As another example cited here [22] the extracted policy blocks TCP traffic on port 7597 while the original NSR permits all TCP traffic, for this reason the CPM splits the reachability property into two which are a TCP traffic permitted with a port range from 0 to 7596 and another reachability property with a port range from 7598 and 65535 in order to have the Isolation Property that blocks communication with port 7597 without any conflict.

The merging process involves two distinct scenarios:

- **Exact match:** If an extracted isolation requirement exactly matches a current reachability requirement, the reachability requirement is removed from the final list, and the isolation requirement is added. This is the case of the first example cited above, the one related to the ICMP traffic.

- **Loose match:** If an extracted isolation requirement loosely matches a current reachability requirement (i.e., the isolation requirement is a subset of the reachability requirement), the reachability requirement is replaced by other reachability requirements that allow the same traffic as the original, except for the portion blocked by the isolation requirement. Subsequently, the isolation requirement itself is added to the final list. This is the case of the second example cited above, the one related to TCP traffic on port 7597.

The *DeriveReachabilityProperties()* function is responsible for generating these new reachability requirements. The algorithm requires more comprehensive testing to ensure reliability in various scenarios. At this stage, although it includes small optimizations and pre-processing steps to enhance efficiency, there is ample scope for advancement. Enhancements are needed to ensure the script can handle more complex cases efficiently and effectively, which is essential for its broader applicability and robustness in different network settings[22].

2.3.3 React-VEREFOO

React-VEREFOO is an advanced version of VEREFOO due to the fact that VEREFOO needs a recomputation of the firewall configuration and a redeployment of the virtual network whenever a change is made to the set of NSR. For this reason, in order to improve the computation time and enhancing the use of resources, React-VEREFOO is the best solution. The approach is to have a metodology that permits to compute automatically the reconfiguration of distributed firewalls everytime a new NSR is specified. This is combined along with automation in order to have optimization and formal verification to ensure that NSRs are satisfied, in a short computation time. React-VEREFOO takes as input the logic topology of the virtual network with the distributed firewall already specified and a pair of NSR's set which are the initial ones, and the Target NSR which are the ones to be implemented. The output produced by React-VEREFOO will be an Allocation Graph with firewalls' filtering rules updated [5][23].

The approach is based on different steps:

- **Formal models:** A formal model is defined to represent the network, the configuration of network functions and the exchanged traffic.
- **Algorithm for detection of network area to be reconfigured:** An algorithm is employed to detect which network's area needs a reconfiguration and it is based on the intersection of the two set of NSR.
- **MaxSMT problem:** The resolution of the MaxSMT problem permits to generate an allocation graph and configuration rules of the firewalls.

Since React-VEREFOO is used at the end of the process, it represents the actual reaction to cyberattacks, because it requires a set of Target Network Security Requirements (NSRs) to initiate the reconfiguration. For this reason, React-VEREFOO must be positioned at the final stage of the workflow because these

Target NSRs are generated by the Security Policy Extractor (SPE) and the Conflict Policy Merger (CPM). Once these components produce the final Target NSR set, React-VEREFOO, together with other required inputs, updates the virtual topology and firewall rules in order to implement the necessary changes and mitigate the detected attack. In line with this approach, [24] emphasizes that traditional manual firewall configuration methods are no longer suitable for next-generation virtualized networks due to their growing complexity. While some automated strategies have been explored, they often remain partially dependent on human intervention and this cited work proposes a fully autonomous firewall reconfiguration framework based on React-VEREFOO, automating all steps from the extraction of security requirements, derived from IDS logs, to the deployment of computed configurations. Moreover, experimental validation confirms the approach's efficiency and its capability to respond effectively to evolving cyber threats.

Importantly, this reconfiguration process done by React-VEREFOO, modifies only the nodes that need to be updated in order to apply the new security rules, thus avoiding a full redeployment of the network. This is in contrast to the standard VEREFOO workflow, which would regenerate the entire deployment from scratch, therefore it reduces the computation time and enhances the resource usage.

This chapter represents also the followed approach for this research. In few words, the first time a topology along with NSR are created and provided to VEREFOO to produce the virtual network with firewalls and nodes deployed and running. After that a looping process starts because, with the virtual network fully working, every time an alert is created due to some rules of the IDS triggered, the reconfiguration will start thanks to SPE, CPM and React-VEREFOO. By correctly employing this approach it is possible to achieve a flexible and adaptive network which is able to react quickly to changes and cyberattacks.

Chapter 3

Thesis Objective

As highlighted in the introduction, within the field of network security even seemingly minor configuration errors or a lack of coordination between security designers and service designer can result in severe vulnerabilities. Such weaknesses may be exploited by adversaries to achieve a variety of critical and potentially catastrophic outcomes, including but not limited to service disruption, data breaches, or substantial financial losses. These risks emphasize the necessity of approaches that reduce the dependency on manual interventions and in this regard, automation emerges as a powerful strategy to minimize the likelihood of human error, increase consistency in network configurations, and ensure a more resilient security posture.

The primary objective of this thesis was the development of a demonstrator capable of showcasing the distinctive features and strengths of VEREFOO framework. This demonstrator is intended not only to serve as a proof-of-concept but also as an educational and explanatory tool, highlighting how the framework can be effectively applied in realistic scenarios.

In order to achieve this primary goal, a set of secondary objectives was first identified. As an initial step, a custom network topology was carefully designed with the intention of resembling a realistic enterprise-like network environment. Alongside this topology, a set of Network Security Requirements (NSRs) was formally defined, with the purpose of specifying and regulating the communication flows between different nodes. These NSRs were deliberately crafted to encompass a heterogeneous mix of reachability and isolation properties, in line with the security-driven methodology discussed in chapter 2.3.2. This ensured that the network specification reflected both functional needs and security requirements, thereby providing a solid foundation for subsequent experiments.

Subsequently, modifications were carried out on the master branch of the framework in order to obtain a fully operational system. The enhanced version of the framework was designed to automatically process the defined topology and NSRs, both of which are provided in a single input file, namely *DemoTopology.xml*. From these two inputs alone, the framework is capable of executing all the necessary steps to generate a fully functional virtual network. This degree of automation illustrates the efficiency and robustness of the framework, particularly in scenarios where scalability and adaptability are critical.

After these preparatory steps, in order to evaluate and demonstrate the reaction capabilities of the framework, a cyberattack scenario was introduced. Specifically, a Denial of Service (DoS) attack was simulated, in the form of an ICMP flood, where one of the servers defined in the topology was targeted by a malicious node acting as the attacker. To enhance realism, the attack was not only technically implemented but also contextualized through a narrative, thereby replicating as closely as possible a plausible real-world threat scenario. This step was essential to evaluate the responsiveness of the framework in dynamic conditions and to highlight its capacity for mitigating ongoing attacks.

Finally, the demonstrator itself was developed. Implemented as a Python script, the demonstrator provides a detailed, step-by-step execution of the network setup and subsequently illustrates the ability of the framework to promptly and efficiently react to the simulated attack. In addition to showcasing the sequence of actions performed, the demonstrator also integrates explanatory outputs, enabling users to follow the process in detail and, if necessary, adapt or extend the framework to different use cases.

Complementary to this, a secondary Python script was created with the specific purpose of highlighting the differences between the virtual topology before and after the attack. This comparison allows a clear visualization of the reconfiguration performed automatically by the framework, thereby evidencing both the nature and the effectiveness of the applied countermeasures. Together, these two demonstrators provide a comprehensive view of the workflow, making the reconfiguration process more transparent and reinforcing the role of automation as a cornerstone in modern network security management.

The final contribution of this research was the design and implementation of a graphical interface for the demonstrator, structured as a fully functional web application composed of a server and a client. Specifically, the server component was developed as a Python script responsible for handling requests and managing the interaction logic with the framework, while the client was implemented using React to provide a modern, intuitive, and user-friendly graphical interface.

This architecture ensures that the demonstrator retains the same core functionalities as the original Python-based implementation, while presenting them in a more accessible and organized way. Through the graphical interface, users are able to follow the different stages of the demonstrator more clearly, receive real-time feedback about the operations being executed, and interact with the system in a cleaner and more structured manner. This approach enhances usability and provides a more engaging representation of the VEREFOO framework's capabilities, making the demonstrator a practical and interactive showcase of its potential.

Chapter 4

Approach

4.1 Approach and metodology

4.1.1 Setup & Workflow

This section offers an overview of the approach, encompassing the setup and execution of the framework and demonstrator. The framework can be easily installed but it requires some additional package that can be installed by linux terminal and from github.

- Ubuntu version 22
- Docker Engine & and Docker compose
- openjdk-1.8
- z3-4.8.15-x64-glibc-2.31.

These are the requirements to be able to run the framework.

As discussed in previous chapters of this research the workflow is composed by different steps 2. The initial phase involves creating a network topology and NSR to make VEREFOO be able to create the Output configuration where firewalls will be deployed in the allocation places specified in the XML file 4.1.

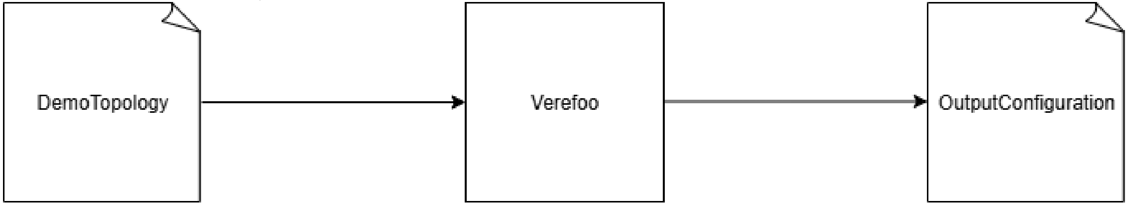


Figure 4.1. Workflow's first phase

s

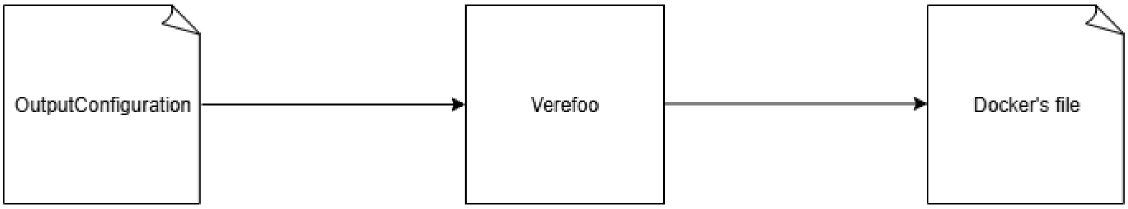


Figure 4.2. Workflow's second phase

After the first phase the resulting OutputConfiguration.xml which contains the allocated firewall, will be provided to VEREFOO once again to obtain all the docker's file to prepare and launch the virtual network 4.2.

Now the virtual network is deployed and fully working. After those two steps the attack simulation will be done and a reaction will be triggered. In particular, the IDS will create a log file that will be read and transformed by SPE in a new property that will be merged without any conflict by the CPM to have an updated topology 4.3.

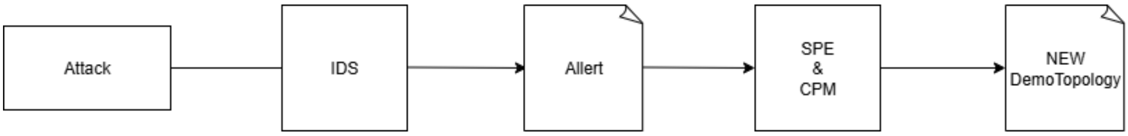


Figure 4.3. Workflow's third phase

At this stage, the updated topology is an XML file that contains the firewalls that have been deployed at the predefined allocation places, as well as two distinct groups of Network Security Requirements (NSRs). The first group, defined within the InitialProperties XML element, represents the set of previously established NSRs, while the second group, specified within the PropertyDefinition element, includes both the pre-existing NSRs and the newly extracted property derived from the IDS's alert. This distinction is essential, as it allows React-VEREFOO to be triggered appropriately because, thanks to the differences between InitialProperties and PropertyDefinition, React-VEREFOO identifies and updates only the set of nodes that are traversed by traffic related to the newly introduced property. This selective update mechanism ensures that only the relevant portions of the virtual topology are reconfigured, avoiding a complete redeployment of the network 4.4.

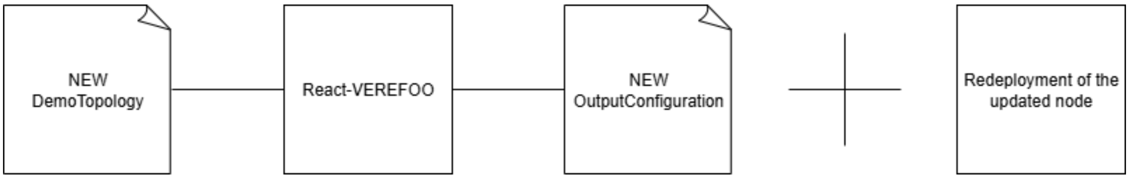


Figure 4.4. Workflow's fourth phase

Now the virtual network is updated, fully working and blocks the attack.

4.1.2 DemoTopology.xml

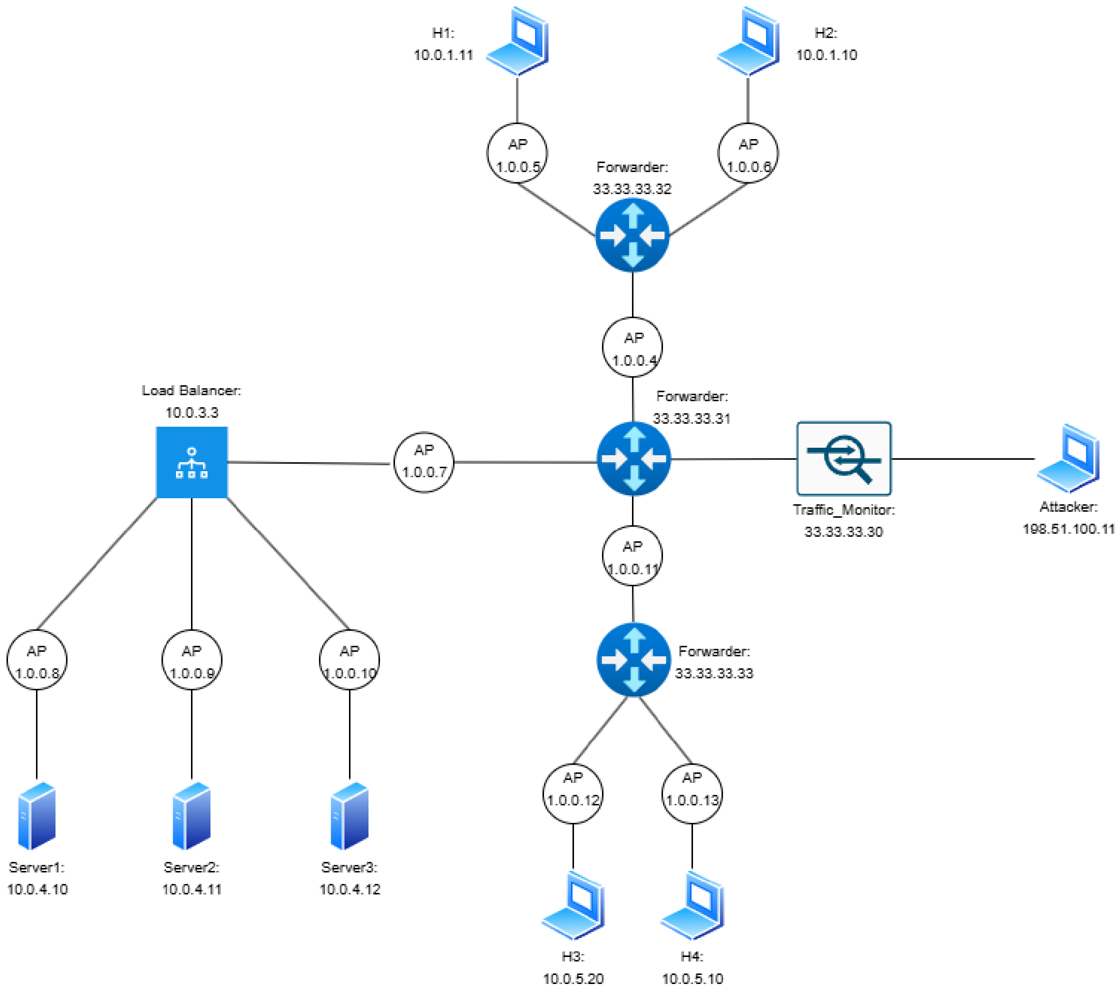


Figure 4.5. Network topology created for the demonstrator

The network topology designed for this research 4.5 was carefully constructed with the aim of closely approximating a realistic enterprise network environment. Although it represents a conceptual abstraction, it was developed with the intention of emulating a segment of a larger and more complex infrastructure. The design choices were guided by practical considerations that are typically encountered in real-world networks, such as subnet segmentation, the placement of critical services, and traffic monitoring points.

Within the topology, three main logical zones can be identified: the first two zones consist of standard hosts and are intended to represent distinct departments or functional units within an organization, located in the 10.0.1.0/24 and

10.0.5.0/24 subnets, respectively. Each of these segments could correspond, for instance, to a business department, human resources or research and development department, in a larger network. The third main zone, associated with the 10.0.4.0/24 subnet, is structured to resemble a data center, here is where more centralized and critical services are presumed to be hosted, such as web servers, or storage systems that could be the main point of attack.

In addition to these three zones, the rightmost section of the topology symbolizes a gateway or interconnection point between the simulated network and an external or extended part of the infrastructure. This area includes two key components: a Traffic Monitor (33.33.33.30) and an host (198.51.100.11), in particular, the traffic monitor plays a pivotal role in this architecture, as it inspects both incoming and outgoing traffic that permits to detect potential threats or anomalies and to trigger alerts accordingly. The host, on the other hand, represents an external entity that could be, for instance, another part of the wider network interacting with internal services.

Following, the XML file will be listed:

Listing 4.1. DemoTopology.xml

```
<graphs>
    <graph id="0">
        <node functional_type="WEBCLIENT" name="198.51.100.11">
            <neighbour name="33.33.33.30" />
            <configuration description="e1" name="attacker1">
                <webclient nameWebServer="10.0.4.10" />
            </configuration>
        </node>

        <node functional_type="TRAFFIC_MONITOR" name="33.33.33.30">
            <neighbour name="198.51.100.11" />
            <neighbour name="33.33.33.31" />
            <configuration name="trafficmonitor1">
                <traffic_monitor>
                    <name>snort3</name>
                </traffic_monitor>
            </configuration>
        </node>

        <node functional_type="FORWARDER" name="33.33.33.31">
            <neighbour name="33.33.33.30" />
            <neighbour name="1.0.0.4" />
            <neighbour name="1.0.0.7" />
            <neighbour name="1.0.0.11" />
            <configuration name="ForwardConf1">
                <forwarder>
                    <name>Forwarder1</name>
                </forwarder>
            </configuration>
        </node>

        <node name="1.0.0.7">
            <neighbour name="33.33.33.31" />
            <neighbour name="33.33.33.34" />
        </node>
        <node functional_type="FORWARDER" name="33.33.33.34">
            <neighbour name="1.0.0.7" />
            <neighbour name="1.0.0.15" />
            <neighbour name="1.0.0.14" />
            <configuration name="ForwardConf4">
                <forwarder>
                    <name>Forwarder4</name>
                </forwarder>
            </configuration>
        </node>

        <node name="1.0.0.15">
            <neighbour name="33.33.33.34" />
            <neighbour name="33.33.33.35" />
        </node>

        <node functional_type="FORWARDER" name="33.33.33.35">
            <neighbour name="1.0.0.15" />
            <neighbour name="1.0.0.16" />
            <neighbour name="1.0.0.17" />
            <configuration name="ForwardConf5">
                <forwarder>
                    <name>Forwarder5</name>
                </forwarder>
            </configuration>
        </node>

        <node name="1.0.0.16">
            <neighbour name="33.33.33.35" />
            <neighbour name="10.0.2.2" />
        </node>

        <node functional_type="WEBCLIENT" name="10.0.2.2">
            <neighbour name="1.0.0.16" />
            <configuration description="e8" name="webclient5">
                <webclient nameWebServer="10.0.4.10" />
            </configuration>
        </node>

        <node name="1.0.0.17">
            <neighbour name="33.33.33.35" />
            <neighbour name="10.0.2.1" />
        </node>

        <node functional_type="WEBCLIENT" name="10.0.2.1">
            <neighbour name="1.0.0.17" />
            <configuration description="e9" name="webclient6">
                <webclient nameWebServer="10.0.4.10" />
            </configuration>
        </node>

        <node name="1.0.0.14">
            <neighbour name="33.33.33.34" />
            <neighbour name="10.0.3.3" />
        </node>

        <node functional_type="LOADBALANCER" name="10.0.3.3">
```

```
            <neighbour name="1.0.0.14" />
            <neighbour name="1.0.0.8" />
            <neighbour name="1.0.0.9" />
            <neighbour name="1.0.0.10" />
            <configuration description="s9" name="loadbalancer">
                <loadbalancer>
                    <pool>10.0.4.10</pool>
                    <pool>10.0.4.11</pool>
                    <pool>10.0.4.12</pool>
                </loadbalancer>
            </configuration>
        </node>

        <node name="1.0.0.8">
            <neighbour name="10.0.3.3" />
            <neighbour name="10.0.4.10" />
        </node>

        <node name="1.0.0.9">
            <neighbour name="10.0.3.3" />
            <neighbour name="10.0.4.11" />
        </node>

        <node name="1.0.0.10">
            <neighbour name="10.0.3.3" />
            <neighbour name="10.0.4.12" />
        </node>

        <node functional_type="WEBSERVER" name="10.0.4.10">
            <neighbour name="1.0.0.8" />
            <configuration description="e2" name="httpserver2">
                <webserver>
                    <name>10.0.4.10</name>
                </webserver>
            </configuration>
        </node>

        <node functional_type="WEBSERVER" name="10.0.4.11">
            <neighbour name="1.0.0.9" />
            <configuration description="e3" name="httpserver3">
                <webserver>
                    <name>10.0.4.11</name>
                </webserver>
            </configuration>
        </node>

        <node functional_type="WEBSERVER" name="10.0.4.12">
            <neighbour name="1.0.0.10" />
            <configuration description="e4" name="httpserver4">
                <webserver>
                    <name>10.0.4.12</name>
                </webserver>
            </configuration>
        </node>

        <node name="1.0.0.4">
            <neighbour name="33.33.33.32" />
            <neighbour name="33.33.33.31" />
        </node>

        <node functional_type="FORWARDER" name="33.33.33.32">
            <neighbour name="1.0.0.4" />
            <neighbour name="1.0.0.5" />
            <neighbour name="1.0.0.6" />
            <configuration name="ForwardConf2">
                <forwarder>
                    <name>Forwarder2</name>
                </forwarder>
            </configuration>
        </node>

        <node name="1.0.0.5">
            <neighbour name="33.33.33.32" />
            <neighbour name="10.0.1.11" />
        </node>

        <node functional_type="WEBCLIENT" name="10.0.1.11">
            <neighbour name="1.0.0.5" />
            <configuration description="e5" name="webclient1">
                <webclient nameWebServer="10.0.4.11" />
            </configuration>
        </node>

        <node name="1.0.0.6">
            <neighbour name="33.33.33.32" />
            <neighbour name="10.0.1.10" />
        </node>

        <node functional_type="WEBCLIENT" name="10.0.1.10">
            <neighbour name="1.0.0.6" />
            <configuration description="e6" name="webclient2">
                <webclient nameWebServer="10.0.4.12" />
            </configuration>
        </node>

        <node name="1.0.0.11">
            <neighbour name="33.33.33.31" />
            <neighbour name="33.33.33.33" />
        </node>

        <node functional_type="FORWARDER" name="33.33.33.33">
            <neighbour name="1.0.0.11" />
            <neighbour name="1.0.0.12" />
            <neighbour name="1.0.0.13" />
            <configuration name="ForwardConf3">
                <forwarder>
                    <name>Forwarder3</name>
                </forwarder>
            </configuration>
        </node>

        <node name="1.0.0.12">
            <neighbour name="33.33.33.33" />
            <neighbour name="10.0.5.20" />
        </node>

        <node functional_type="WEBCLIENT" name="10.0.5.20">
            <neighbour name="1.0.0.12" />
            <configuration description="e7" name="webclient4">
                <webclient nameWebServer="10.0.4.11" />
            </configuration>
        </node>

        <node name="1.0.0.13">
            <neighbour name="33.33.33.33" />
            <neighbour name="10.0.5.10" />
        </node>

        <node functional_type="WEBCLIENT" name="10.0.5.10">
            <neighbour name="1.0.0.13" />
            <configuration description="e8" name="webclient3">
                <webclient nameWebServer="10.0.4.12" />
            </configuration>
        </node>
    </graph>
</graphs>
```

As it is possible to see both from the image and from the XML file, this is defined as an Allocation Graph because inside, the file Allocation Places are already defined. This is done with a design choice in mind, because, by placing by hand the allocation places in specific point, VEREFOO will allocate firewalls in specific part of the network. Firewalls, due to the design choices, will not be placed to the right side of the forwarder with address 33.33.33.31, because conceptually, other firewalls will be placed in the other part of the network that is not represented in this topology.

Listing 4.2. Focus on Traffic Monitor and Forwarder

```
<node functional_type="TRAFFIC_MONITOR" name="33.33.33.30">
    <neighbour name="198.51.100.11" />
    <neighbour name="33.33.33.31" />
    <configuration name="trafficmonitor1">
        <traffic_monitor>
            <name>snort3</name>
        </traffic_monitor>
    </configuration>
</node>

<node functional_type="FORWARDER" name="33.33.33.31">
    <neighbour name="33.33.33.30" />
    <neighbour name="1.0.0.4" />
    <neighbour name="1.0.0.7" />
    <neighbour name="1.0.0.11" />
    <configuration name="ForwardConf1">
        <forwarder>
            <name>Forwarder1</name>
        </forwarder>
    </configuration>
</node>
```

Inside the XML file there two nodes must be discussed. Firstly, there is a functional_type named FORWARDER, this is an element that can be inserted manually by the user or VEREFOO can insert them onto unused allocation places. Forwarders do not have a specific role inside the topology because their task is simply to forward traffic; they are abstraction of network elements like switches or router.

The other key element defined in the XML file is the node with functional type TRAFFIC_MONITOR and address 33.33.33.30. As shown in the figure, this node is configured to operate as a Network Intrusion Detection System (NIDS) using Snort 3 and through the configuration tag, VEREFOO is instructed to deploy this node inside a Docker container and configure it as an IDS instance running Snort 3, where both custom and standard detection rules can be specified. In VEREFOO, traffic monitors are structurally similar to forwarders since they inherit the same basic elements. However, the presence of the *functional_type="TRAFFIC_MONITOR"* attribute and the specific configuration block, allows VEREFOO to recognize and treat the node as a dedicated traffic monitor.

As said before, VEREFOO requires an additional input which are the network security requirements that are properties that express connectivity between endpoints. The NSR used in this thesis are the following:

Listing 4.3. NSR specified in DemoTopology.xml

```
<PropertyDefinition>
        <!-- WebClient1 -->
        <Property graph="0" name="ReachabilityProperty" src="10.0.1.11"
            dst="10.0.4.11" lv4proto="TCP"/>
        <Property graph="0" name="ReachabilityProperty" src="10.0.4.11"
            dst="10.0.1.11" lv4proto="TCP"/>
        <Property graph="0" name="ReachabilityProperty" src="10.0.1.11"
            dst="10.0.1.10" lv4proto="TCP"/>
        <Property graph="0" name="ReachabilityProperty" src="10.0.1.10"
            dst="10.0.1.11" lv4proto="TCP"/>
        <Property graph="0" name="ReachabilityProperty" src="10.0.1.11"
            dst="10.0.5.10" lv4proto="OTHER"/>
        <Property graph="0" name="ReachabilityProperty" src="10.0.5.10"
            dst="10.0.1.11" lv4proto="OTHER"/>

        <Property graph="0" name="IsolationProperty" src="10.0.1.11"
            dst="10.0.4.10" lv4proto="TCP"/>
        <Property graph="0" name="IsolationProperty" src="10.0.4.10"
            dst="10.0.1.11" lv4proto="TCP"/>
        <Property graph="0" name="IsolationProperty" src="10.0.1.11"
            dst="10.0.4.12" lv4proto="TCP"/>
        <Property graph="0" name="IsolationProperty" src="10.0.4.12"
            dst="10.0.1.11" lv4proto="TCP"/>
        <Property graph="0" name="IsolationProperty" src="10.0.1.11"
            dst="10.0.5.20" lv4proto="OTHER"/>
        <Property graph="0" name="IsolationProperty" src="10.0.5.20"
            dst="10.0.1.11" lv4proto="OTHER"/>

        <!-- WebClient2 -->
        <Property graph="0" name="ReachabilityProperty" src="10.0.1.10"
            dst="10.0.4.11" lv4proto="TCP"/>
        <Property graph="0" name="ReachabilityProperty" src="10.0.4.11"
            dst="10.0.1.10" lv4proto="TCP"/>
        <Property graph="0" name="ReachabilityProperty" src="10.0.1.10"
            dst="10.0.5.10" lv4proto="OTHER"/>
        <Property graph="0" name="ReachabilityProperty" src="10.0.5.10"
            dst="10.0.1.10" lv4proto="OTHER"/>

        <Property graph="0" name="IsolationProperty" src="10.0.1.10"
            dst="10.0.4.10" lv4proto="TCP"/>
        <Property graph="0" name="IsolationProperty" src="10.0.4.10"
            dst="10.0.1.10" lv4proto="TCP"/>
        <Property graph="0" name="IsolationProperty" src="10.0.1.10"
            dst="10.0.4.12" lv4proto="TCP"/>
        <Property graph="0" name="IsolationProperty" src="10.0.4.12"
            dst="10.0.1.10" lv4proto="TCP"/>
        <Property graph="0" name="IsolationProperty" src="10.0.1.10"
            dst="10.0.5.20" lv4proto="OTHER"/>
        <Property graph="0" name="IsolationProperty" src="10.0.5.20"
            dst="10.0.1.10" lv4proto="OTHER"/>

        <!-- WebClient3 -->
        <Property graph="0" name="ReachabilityProperty" src="10.0.5.20"
            dst="10.0.4.10" lv4proto="TCP"/>
        <Property graph="0" name="ReachabilityProperty" src="10.0.4.10"
            dst="10.0.5.20" lv4proto="TCP"/>
        <Property graph="0" name="ReachabilityProperty" src="10.0.5.20"
            dst="10.0.4.12" lv4proto="TCP"/>
        <Property graph="0" name="ReachabilityProperty" src="10.0.4.12"
            dst="10.0.5.20" lv4proto="TCP"/>

        <Property graph="0" name="IsolationProperty" src="10.0.5.20"
            dst="10.0.4.11" lv4proto="TCP"/>
        <Property graph="0" name="IsolationProperty" src="10.0.4.11"
            dst="10.0.5.20" lv4proto="TCP"/>

        <!-- WebClient4 -->
        <Property graph="0" name="ReachabilityProperty" src="10.0.5.10"
            dst="10.0.4.10" lv4proto="TCP"/>
        <Property graph="0" name="ReachabilityProperty" src="10.0.4.10"
            dst="10.0.5.10" lv4proto="TCP"/>
        <Property graph="0" name="ReachabilityProperty" src="10.0.5.10"
            dst="10.0.4.12" lv4proto="TCP"/>
        <Property graph="0" name="ReachabilityProperty" src="10.0.4.12"
            dst="10.0.5.10" lv4proto="TCP"/>

        <Property graph="0" name="IsolationProperty" src="10.0.5.10"
            dst="10.0.4.11" lv4proto="TCP"/>
        <Property graph="0" name="IsolationProperty" src="10.0.4.11"
            dst="10.0.5.10" lv4proto="TCP"/>

        <!-- WebClientTraff_Monitor -->
        <Property graph="0" name="ReachabilityProperty" src="198.51.100.11"
            dst="10.0.4.12" lv4proto="OTHER"/>
        <Property graph="0" name="ReachabilityProperty" src="10.0.4.12"
            dst="198.51.100.11" lv4proto="OTHER"/>
        <Property graph="0" name="ReachabilityProperty" src="198.51.100.11"
            dst="10.0.5.20" lv4proto="OTHER"/>
        <Property graph="0" name="ReachabilityProperty" src="10.0.5.20"
            dst="198.51.100.11" lv4proto="OTHER"/>
        <Property graph="0" name="ReachabilityProperty" src="198.51.100.11"
            dst="10.0.5.10" lv4proto="OTHER"/>
        <Property graph="0" name="ReachabilityProperty" src="10.0.5.10"
            dst="198.51.100.11" lv4proto="OTHER"/>

        <Property graph="0" name="IsolationProperty" src="198.51.100.11"
            dst="10.0.4.11" lv4proto="OTHER"/>
        <Property graph="0" name="IsolationProperty" src="10.0.4.11"
            dst="198.51.100.11" lv4proto="OTHER"/>
        <Property graph="0" name="IsolationProperty" src="198.51.100.11"
            dst="10.0.4.10" lv4proto="OTHER"/>
        <Property graph="0" name="IsolationProperty" src="10.0.4.10"
            dst="198.51.100.11" lv4proto="OTHER"/>

        </PropertyDefinition>
```

The properties defined within the XML file have been designed with a general-purpose approach, aiming to approximate realistic network scenarios as closely as possible. All client nodes in the topology are associated with reachability and isolation properties ensuring bidirectional traffic specifications between sources and destinations. This bidirectionality reflects typical enterprise environments, where communication between nodes must often be validated in both directions for security and operational reasons.

In the properties the layer 4 protocols specified are TCP and UDP, which represent the most commonly used transport protocols, as well as the OTHER category, which is used in VEREFOO to indicate any protocol not explicitly classified as TCP or UDP, which allows the system to be flexible. Furthermore, no specific source or destination port numbers are defined in the current set of properties. Although such granularity could certainly be added to refine traffic filtering, its absence does not impact the structural outcome or behavior of the resulting virtual topology. In fact, incorporating port's constraints would require only minimal adjustments to the property definitions and would not alter the placement of firewalls, nor the core logic of the system, just modifications of firewalls' rules.

A brief summary of the defined properties, with reference to 4.5 figure, is provided below to make 4.3 more clear:

- Host1 (10.0.1.11)
    - Can communicate with: Server1 (10.0.4.10) via TCP, Host2 (10.0.1.10) via TCP, Host4 (10.0.5.10) via OTHER.
    - Cannot communicate with: Server1 (10.0.4.10) via TCP, Server3 (10.0.4.12) via TCP and Host3 (10.0.5.20) via OTHER.
- Host2 (10.0.1.10)
    - Can communicate with: Server2 (10.0.4.11) via TCP and Host4 (10.0.5.10) via OTHER.
    - Cannot communicate with: Server1 (10.0.410) via TCP, Server3 (10.0.4.12) via TCP and Host3 (10.0.4.20) via OTHER.

- Host3 (10.0.5.20)
    - Can communicate with: Server 1 (10.0.4.10) via TCP and Server3 (10.0.4.12) via TCP.
    - Cannot communicate with: Server2 (10.0.4.11) via TCP. From the other properties it cannot also communicate with Host1.
- Host4 (10.0.5.10)
    - Can communicate with: Server1 (10.0.4.10) via TCP and Server3 (10.0.4.12) via TCP. From the other properties it can communicate with Host1 and Host2.
    - Cannot communicate with: Server2 (10.0.4.11) via TCP.
- Attacker (198.51.100.11)
    - Can communicate with: Server3 (10.0.4.12) via OTHER, Host3 (10.0.5.20) via OTHER, Host4 (10.0.5.10) via OTHER.
    - Cannot communicate with: Server2 (10.0.4.11) via OTHER and Server1 (10.0.4.10) via OTHER.

4.1.3 OutputConfiguration.xml

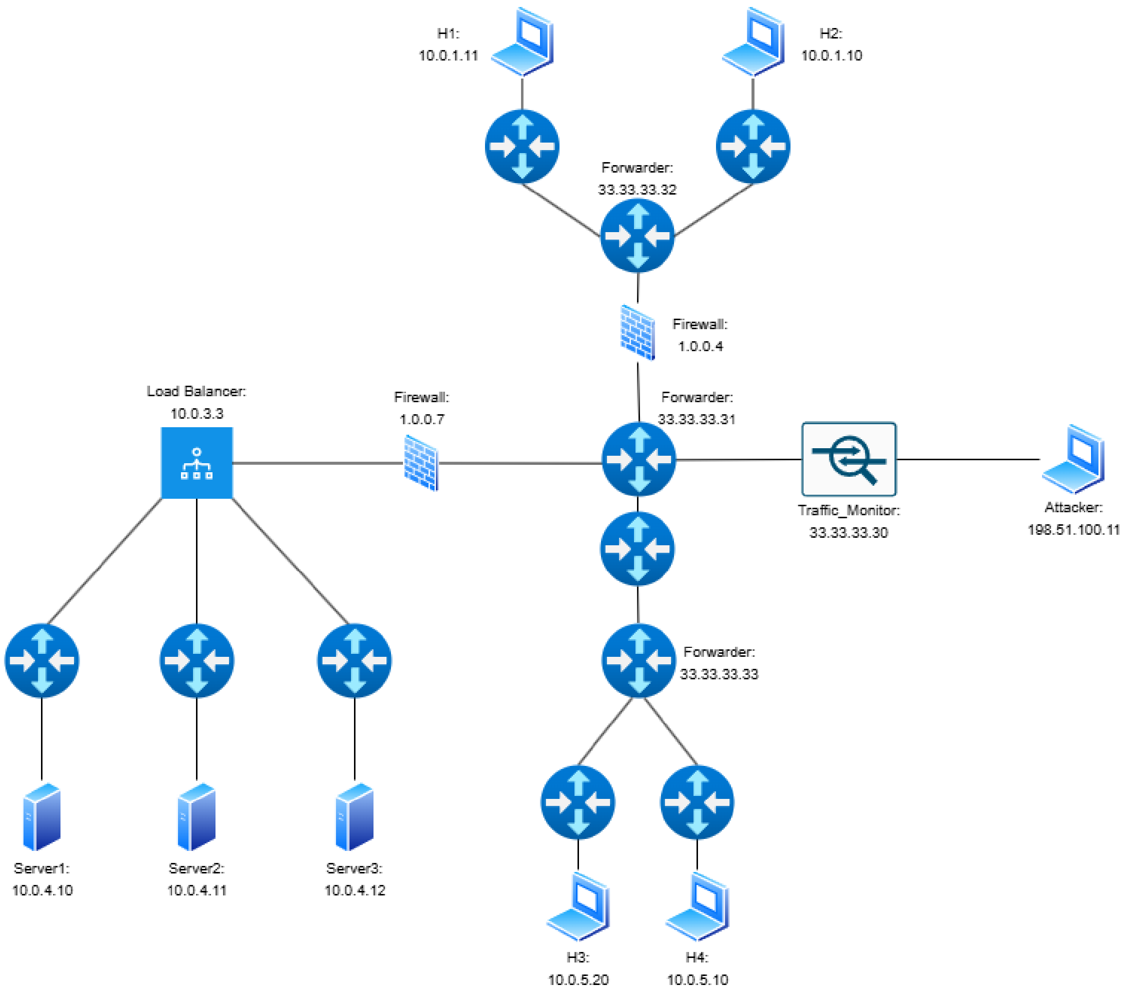


Figure 4.6. OutputConfiguration.xml

4.6 is the result of the workflow's first phase 4.1. The DemoTopology.xml file is provided as input to VEREFOO, which, by leveraging the defined allocation places and the specified Network Security Requirements (NSRs), it creates a MaxSMT problem that is solved with Z3 and produces as output the OutputConfiguration.xml file. During this process, all allocation places are initially transformed into forwarders, which are nodes responsible solely for forwarding packets between other nodes, however, a subset of these allocation places is further promoted to function as firewalls. Specifically, the allocation places at addresses 1.0.0.7 and 1.0.0.4 are converted into firewall instances and the exact configuration of these firewalls is determined by the NSRs defined within the DemoTopology.xml input, ensuring that the deployed security mechanisms reflect the intended communication policies and isolation constraints.

The XML file produced as output is similar to the one presented in 4.1, for this reason, only firewalls will be cited here.

Listing 4.4. Firewalls defined in OutputConfiguration.xml

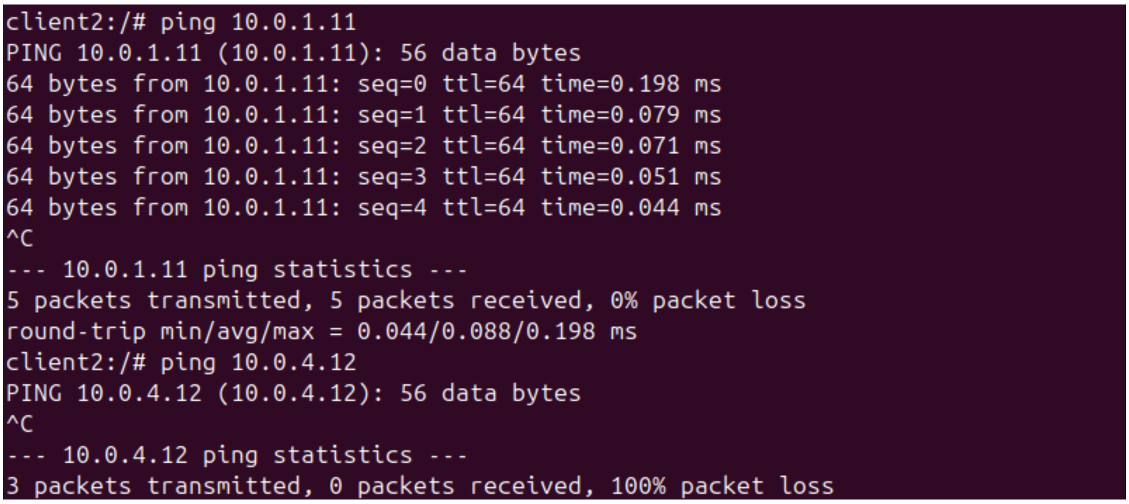
```
<node name="1.0.0.7" functional_type="FIREWALL">
    <neighbour name="33.33.33.31"/>
    <neighbour name="10.0.3.3"/>
    <configuration name="AutoConf" description="1">
        <firewall defaultAction="ALLOW">
            <elements>
                <action>DENY</action>
                <source>10.0.4.11</source>
                <destination>10.0.5.10</destination>
                <protocol>TCP</protocol>
                <src_port>0-65535</src_port>
                <dst_port>0-65535</dst_port>
            </elements>
            <elements>
                <action>DENY</action>
                <source>10.0.4.11</source>
                <destination>198.51.100.11</destination>
                <protocol>OTHER</protocol>
                <src_port>0-65535</src_port>
                <dst_port>0-65535</dst_port>
            </elements>
            <elements>
                <action>DENY</action>
                <source>10.0.5.10</source>
                <destination>10.0.4.11</destination>
                <protocol>TCP</protocol>
                <src_port>0-65535</src_port>
                <dst_port>0-65535</dst_port>
            </elements>
            <elements>
                <action>DENY</action>
                <source>10.0.4.11</source>
                <destination>10.0.5.20</destination>
                <protocol>TCP</protocol>
                <src_port>0-65535</src_port>
                <dst_port>0-65535</dst_port>
            </elements>
            <elements>
                <action>DENY</action>
                <source>198.51.100.11</source>
                <destination>10.0.4.11</destination>
                <protocol>OTHER</protocol>
                <src_port>0-65535</src_port>
                <dst_port>0-65535</dst_port>
            </elements>
            <elements>
                <action>DENY</action>
                <source>10.0.5.20</source>
                <destination>10.0.4.11</destination>
                <protocol>TCP</protocol>
                <src_port>0-65535</src_port>
                <dst_port>0-65535</dst_port>
            </elements>
            <elements>
                <action>DENY</action>
```

```
                    <source>10.0.4.10</source>
                    <destination>198.51.100.11</destination>
                    <protocol>OTHER</protocol>
                    <src_port>0-65535</src_port>
                    <dst_port>0-65535</dst_port>
                </elements>
                <elements>
                    <action>DENY</action>
                    <source>198.51.100.11</source>
                    <destination>10.0.4.10</destination>
                    <protocol>OTHER</protocol>
                    <src_port>0-65535</src_port>
                    <dst_port>0-65535</dst_port>
                </elements>
            </firewall>
        </configuration>
    </node>

    <node name="1.0.0.4" functional_type="FIREWALL">
        <neighbour name="33.33.33.32"/>
        <neighbour name="33.33.33.31"/>
        <configuration name="AutoConf" description="2">
            <firewall defaultAction="DENY">
                <elements>
                    <action>ALLOW</action>
                    <source>10.0.1.11</source>
                    <destination>10.0.4.11</destination>
                    <protocol>TCP</protocol>
                    <src_port>0-65535</src_port>
                    <dst_port>0-65535</dst_port>
                </elements>
                <elements>
                    <action>ALLOW</action>
                    <source>10.0.1.10</source>
                    <destination>10.0.5.10</destination>
                    <protocol>OTHER</protocol>
                    <src_port>0-65535</src_port>
                    <dst_port>0-65535</dst_port>
                </elements>
                <elements>
                    <action>ALLOW</action>
                    <source>10.0.4.11</source>
                    <destination>10.0.1.11</destination>
                    <protocol>TCP</protocol>
                    <src_port>0-65535</src_port>
                    <dst_port>0-65535</dst_port>
                </elements>
                <elements>
                    <action>ALLOW</action>
                    <source>10.0.5.10</source>
                    <destination>10.0.1.11</destination>
                    <protocol>OTHER</protocol>
                    <src_port>0-65535</src_port>
                    <dst_port>0-65535</dst_port>
                </elements>
                <elements>
                    <action>ALLOW</action>
                    <source>10.0.4.11</source>
                    <destination>10.0.1.10</destination>
                    <protocol>TCP</protocol>
                    <src_port>0-65535</src_port>
                    <dst_port>0-65535</dst_port>
                </elements>
                <elements>
                    <action>ALLOW</action>
                    <source>10.0.1.11</source>
                    <destination>10.0.5.10</destination>
                    <protocol>OTHER</protocol>
                    <src_port>0-65535</src_port>
                    <dst_port>0-65535</dst_port>
                </elements>
                <elements>
                    <action>ALLOW</action>
                    <source>10.0.5.10</source>
                    <destination>10.0.1.10</destination>
                    <protocol>OTHER</protocol>
                    <src_port>0-65535</src_port>
                    <dst_port>0-65535</dst_port>
                </elements>
                <elements>
                    <action>ALLOW</action>
                    <source>10.0.1.10</source>
                    <destination>10.0.4.11</destination>
                    <protocol>TCP</protocol>
                    <src_port>0-65535</src_port>
                    <dst_port>0-65535</dst_port>
                </elements>
            </firewall>
        </configuration>
    </node>
```

- Firewall in 1.0.0.7 has as a default action ALLOW, meaning that all that is not explicitly forbidden regarding packets that traverse the firewall, will be accepted. Furthermore, this firewall is mainly in charge of blocking all the traffic that reaches server1, server2 and server3, from the various webclient defined in the topology. It takes from the NSR defined, all the crucial information such as IP addresses and protocols, furthermore, since any port is specified in the NSR, the rules inside the firewall has as a src_port and dst_port all the possible ports (from 0 to 65535).
- Firewall in 1.0.0.4 has as a default action DENY, meaning that all that is not explicitly accepted regarding packets that traverse the firewall, will be dropped. This firewall is mainly in charge of regulating all the traffic that regards the communication between weblclients defined inside DemoTopology.xml.

OutputConfiguration.xml file is then passed once again to VEREFOO in order to generate all the necessary Docker files required for the deployment of a fully virtualized network which enables the practical testing of the firewall rules defined by VEREFOO. Leveraging Docker's capabilities, each node in the network topology is instantiated as an independent container, allowing for a testing environment and a fully working virtual network.



```
client2:/# ping 10.0.1.11
PING 10.0.1.11 (10.0.1.11): 56 data bytes
64 bytes from 10.0.1.11: seq=0 ttl=64 time=0.198 ms
64 bytes from 10.0.1.11: seq=1 ttl=64 time=0.079 ms
64 bytes from 10.0.1.11: seq=2 ttl=64 time=0.071 ms
64 bytes from 10.0.1.11: seq=3 ttl=64 time=0.051 ms
64 bytes from 10.0.1.11: seq=4 ttl=64 time=0.044 ms
^C
--- 10.0.1.11 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 0.044/0.088/0.198 ms
client2:/# ping 10.0.4.12
PING 10.0.4.12 (10.0.4.12): 56 data bytes
^C
--- 10.0.4.12 ping statistics ---
3 packets transmitted, 0 packets received, 100% packet loss
```

Figure 4.7. Some pings sent from 10.0.1.10

Figure 4.7 illustrates the correct implementation of Network Security Requirements (NSRs) and firewall configurations. The host 10.0.1.10, referred to as client2 in the figure, is associated with a reachability property toward 10.0.1.11 and an isolation property with respect to the server at 10.0.4.12. These properties directly influence the configuration of the firewalls located at nodes 1.0.0.4 and 1.0.0.7.

Specifically, firewall 1.0.0.4 is configured with a default action set to deny, and no explicit rule is defined to allow traffic between client2 and the server at 10.0.4.12. As a result, any communication attempt from client2 to the server is correctly blocked, thereby enforcing the defined isolation policy.

4.1.4 Story of the attack

This section defines the attack scenario used in the demonstrator. As previously discussed, both the network topology and the attack have been designed to approximate a realistic environment as closely as possible. The topology was conceived as an abstraction of a much larger enterprise network, specifically, it includes multiple subnets that could represent different departments within a company.

The first two subnets, containing the hosts 10.0.1.11 and 10.0.1.10, and 10.0.5.20 and 10.0.5.10 respectively, could correspond to departmental or internal office networks. A third subnet, composed of the hosts 10.0.4.10, 10.0.4.11, and 10.0.4.12, represents a simplified data center, which serves as the central resource hub within the network infrastructure. In fact, those servers are also backed up by a load balancer which task is to divide the traffic among the different servers in his control.

The attack originates from the host 198.51.100.11, which is intentionally positioned outside the network defined in this topology to simulate either a compromised internal device or an external attacker that has gained unauthorized access to the network. Although this host is technically placed within the topology, it can be conceptually viewed as part of a broader enterprise infrastructure not fully represented in this abstraction.

To reflect real-world security practices, this node is positioned behind a TRAFFIC_MONITOR, which acts as an Intrusion Detection System (IDS), in particular a network intrusion detection system, since it is deployed inside a network node. In actual enterprise networks, it is common to deploy internal IDS instances to monitor east-west traffic, detect insider threats, or provide an additional layer of defense against threats or lateral movement within the infrastructure. The attacker node could have been compromised in various ways, for example, by an employee targeted and deceived by a phishing attack or an employee introducing a USB drive infected with malware, which subsequently enabled an external adversary to take control of the host. As another example, this host could be part of the IT department and, since generally they manage all the virtual network, has permission inside the whole network, they could be a valuable target from which attacks could start.

CISA (Cybersecurity & Infrastructure Security Agency), America's cyber defence agency, defines an insider as any person who has or had authorized access to or knowledge of an organization's resources, including personnel, facilities, information, equipment, networks, and systems, furthermore, insider threat is the potential for an insider to use their authorized access or understanding of an organization to harm that organization.

This harm can include malicious, complacent, or unintentional acts that negatively affect the integrity, confidentiality, and availability of the organization, its data, personnel, or facilities. CISA defines insider threat as the threat that an insider will use their authorized access, wittingly or unwittingly, to do harm to the department's mission, resources, personnel, facilities, information, equipment, networks, or systems. This threat can manifest as damage to the department through the following insider behaviors: espionage, terrorism, unauthorized disclosure of information, unintentional loss or degradation of departmental resources or capabilities, etc [25] [26]. Moreover, according to the recent report on insider threats of Cybersecurity Insiders 2024, 83% of organizations reported at least one insider attack in the last year. Even more surprising than this statistic is that organizations that experienced 11-20 insider attacks saw an increase of five times the amount of attacks they did in 2023 moving from just 4% to 21% in the last 12 months [27].

A notable real-world case illustrating the risks posed by insider threats and the theft of intellectual property involved two former employees of General Electric (GE), Jean Patrice Delia and Miguel Sernas. Over the course of several years, Delia downloaded over 8,000 confidential files, including sensitive cost models and a proprietary algorithm used to calibrate industrial turbines, with the intent to start a competing company. With insider access, Delia exploited his privileges to retrieve trade secrets that were instrumental to GE's competitive advantage in the energy sector. The situation escalated when the new company submitted a suspiciously low bid for a major contract in Saudi Arabia, prompting an internal investigation and later an FBI-led operation.

The investigation, which lasted several years, revealed extensive misuse of internal systems and data exfiltration through email and cloud storage, both individuals ultimately pleaded guilty to conspiracy to steal trade secrets. This case highlights how internal actors can leverage privileged access to inflict severe economic damage and underscores the importance of continuous monitoring, access control, and reactive security frameworks, particularly in environments handling sensitive or proprietary data [28].

The selected attack for this research, is an ICMP flood, a form of Denial of Service (DoS) attack. In this scenario, the attacker (198.51.100.11) is able to reach the data center server at 10.0.4.12, taking advantage of the existing firewall configurations that permit communication between these two nodes. The attack consists of generating a high volume of ICMP Echo Requests (commonly known as ping requests) with the goal of overwhelming the target and degrading its ability to respond to legitimate traffic. By flooding the server with requests, the attacker attempts to make it unreachable by other nodes in the infrastructure, effectively disrupting part of the enterprise services.

Despite being one of the oldest forms of Denial of Service (DoS) attacks, ICMP flood attacks remain a significant threat to network infrastructure, especially when scaled into Distributed Denial of Service (DDoS) scenarios. As discussed in [29], attackers can leverage botnets to generate high volumes of ICMP packets that overwhelm the target's bandwidth, rendering services inaccessible to legitimate users. The paper emphasizes that while ICMP is a fundamental protocol used for diagnostics and connectivity testing (e.g., through the ping utility), it can be easily exploited to saturate a victim's network. Moreover, the authors further present a detailed algorithmic model for carrying out ICMP-based attacks, demonstrating how even a modest number of nodes can inflict severe disruption when configured appropriately. The study highlights that legacy attack vectors like ICMP floods should not be underestimated, as they continue to pose a real threat, especially in environments with limited bandwidth or insufficient mitigation controls.

4.1.5 Attack simulation

This section presents the simulation of the attack scenario, which, as previously discussed, consists of an ICMP flood attack targeting one of the web servers within the virtualized network topology. To enable the detection of this type of malicious behavior, a custom Snort rule was developed and integrated into the Traffic Monitor node configured as an Intrusion Detection System (IDS).

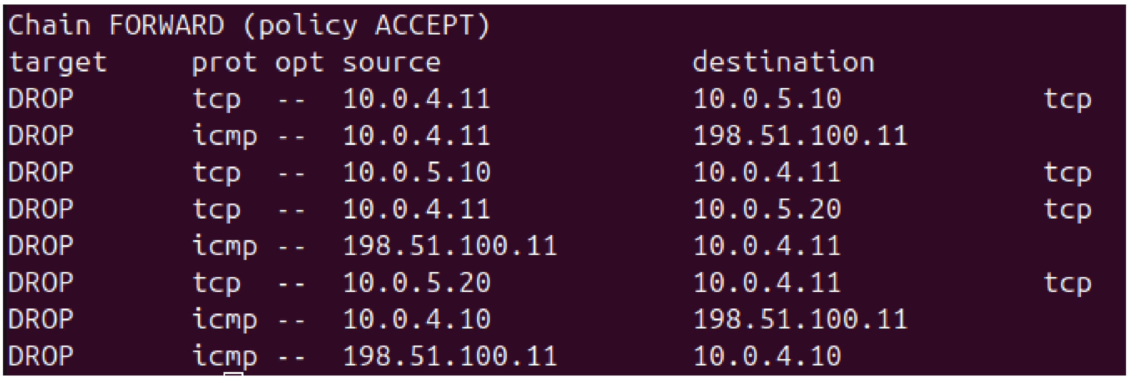
Listing 4.5. Snort's custom rule created for this research

```
Custom Snort Rule:
alert_icmp_$EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP flood attempt";
    detection_filter:track_by_src, count 10, seconds 1; sid:1000001; rev:1;)
```

As shown in listing 4.5 representing the detection rule, an alert is triggered whenever the IDS detects more than 10 ICMP packets per second originating from the same source IP address. This rule was carefully designed with two primary objectives in mind:

- To detect ICMP flood attacks. This specific type of Denial of Service attack relies on overwhelming the target server with a large number of ICMP echo request messages in a short time interval. The ultimate goal is to consume bandwidth and processing resources, thereby degrading the server's ability to respond to legitimate traffic and disrupting the availability of its services.

- To preserve the legitimate use of ICMP, particularly for diagnostic purposes. The rule was calibrated to avoid false positives by allowing a small number of ICMP packets per second, thus maintaining essential network functionality such as the standard ping command. The ping command, when executed with default parameters, typically sends only four ICMP packets, which remains below the detection threshold defined in the custom rule and, therefore, does not trigger an alert.

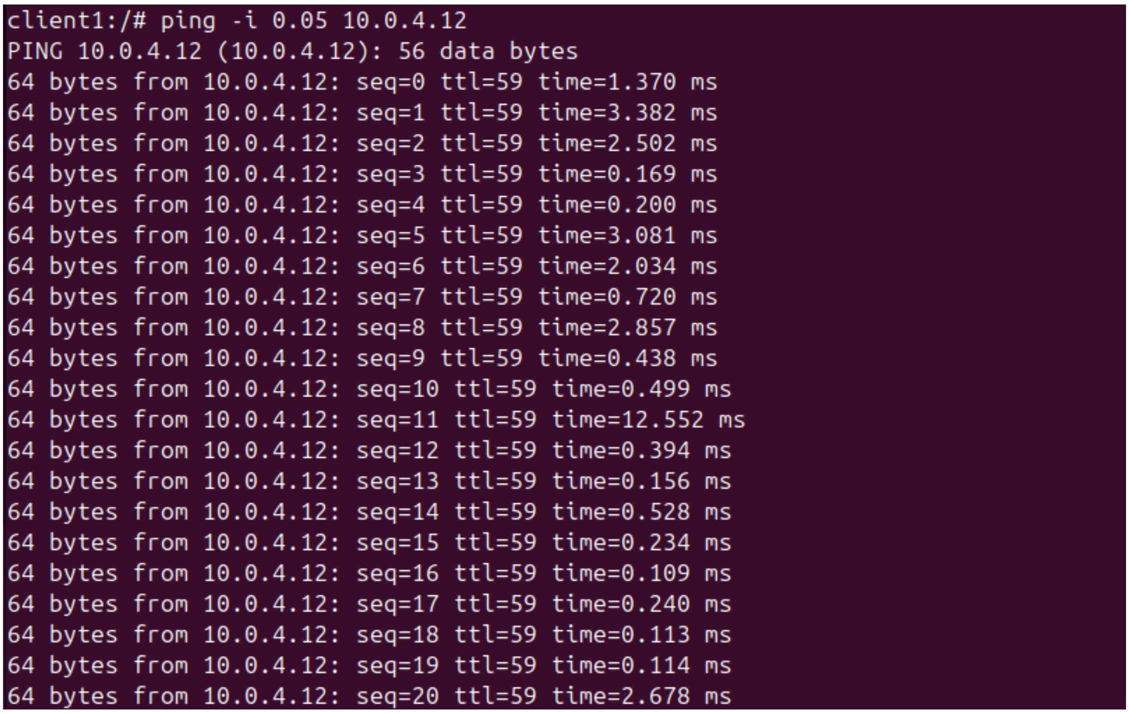


```
Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
DROP       tcp  --  10.0.4.11            10.0.5.10            tcp
DROP       icmp --  10.0.4.11            198.51.100.11
DROP       tcp  --  10.0.5.10            10.0.4.11            tcp
DROP       tcp  --  10.0.4.11            10.0.5.20            tcp
DROP       icmp --  198.51.100.11        10.0.4.11
DROP       tcp  --  10.0.5.20            10.0.4.11            tcp
DROP       icmp --  10.0.4.10            198.51.100.11
DROP       icmp --  198.51.100.11        10.0.4.10
```

Figure 4.8. Firewall 1.0.0.7 rules

Additionally, the figure 4.8 of the firewall configuration illustrates the default policy and the set of explicitly denied communications. The firewall 1.0.0.7 is configured with a default accept policy, meaning that all traffic is allowed unless explicitly blocked. Moreover, there is no rule that blocks communication between the host 198.51.100.11 and the server 10.0.4.12. This omission might be the result of a misconfiguration or simply a design choice, assuming that such communication is considered legitimate in the given network context. Regardless of the rationale, this represents a critical opportunity for an attacker, who can exploit this open channel to launch a successful ICMP flood, effectively bypassing existing security measures and targeting a vulnerable endpoint in the infrastructure.

The attack is then launched and the command *ping -i 0.05 10.0.4.12* will be executed inside the attacker terminal. This command means that an ICMP packet is sent every 0.05 seconds, meaning that 20 packets will be sent every 20 seconds. A much more smaller interval can be used but for testing purposes this interval is used as it can trigger anyways the IDS' rule. Sending ICMP echo request in a much smaller interval saturates bandit more quickly than this attack. As shown in Figure 4.9, the packet being sent has a payload size of 64 bytes, which corresponds to the default value used by the ping command. It is important to highlight that this size can be increased using the "-s" flag, for instance, specifying a value of



```
client1:/# ping -i 0.05 10.0.4.12
PING 10.0.4.12 (10.0.4.12): 56 data bytes
64 bytes from 10.0.4.12: seq=0 ttl=59 time=1.370 ms
64 bytes from 10.0.4.12: seq=1 ttl=59 time=3.382 ms
64 bytes from 10.0.4.12: seq=2 ttl=59 time=2.502 ms
64 bytes from 10.0.4.12: seq=3 ttl=59 time=0.169 ms
64 bytes from 10.0.4.12: seq=4 ttl=59 time=0.200 ms
64 bytes from 10.0.4.12: seq=5 ttl=59 time=3.081 ms
64 bytes from 10.0.4.12: seq=6 ttl=59 time=2.034 ms
64 bytes from 10.0.4.12: seq=7 ttl=59 time=0.720 ms
64 bytes from 10.0.4.12: seq=8 ttl=59 time=2.857 ms
64 bytes from 10.0.4.12: seq=9 ttl=59 time=0.438 ms
64 bytes from 10.0.4.12: seq=10 ttl=59 time=0.499 ms
64 bytes from 10.0.4.12: seq=11 ttl=59 time=12.552 ms
64 bytes from 10.0.4.12: seq=12 ttl=59 time=0.394 ms
64 bytes from 10.0.4.12: seq=13 ttl=59 time=0.156 ms
64 bytes from 10.0.4.12: seq=14 ttl=59 time=0.528 ms
64 bytes from 10.0.4.12: seq=15 ttl=59 time=0.234 ms
64 bytes from 10.0.4.12: seq=16 ttl=59 time=0.109 ms
64 bytes from 10.0.4.12: seq=17 ttl=59 time=0.240 ms
64 bytes from 10.0.4.12: seq=18 ttl=59 time=0.113 ms
64 bytes from 10.0.4.12: seq=19 ttl=59 time=0.114 ms
64 bytes from 10.0.4.12: seq=20 ttl=59 time=2.678 ms
```

Figure 4.9. Attack execution

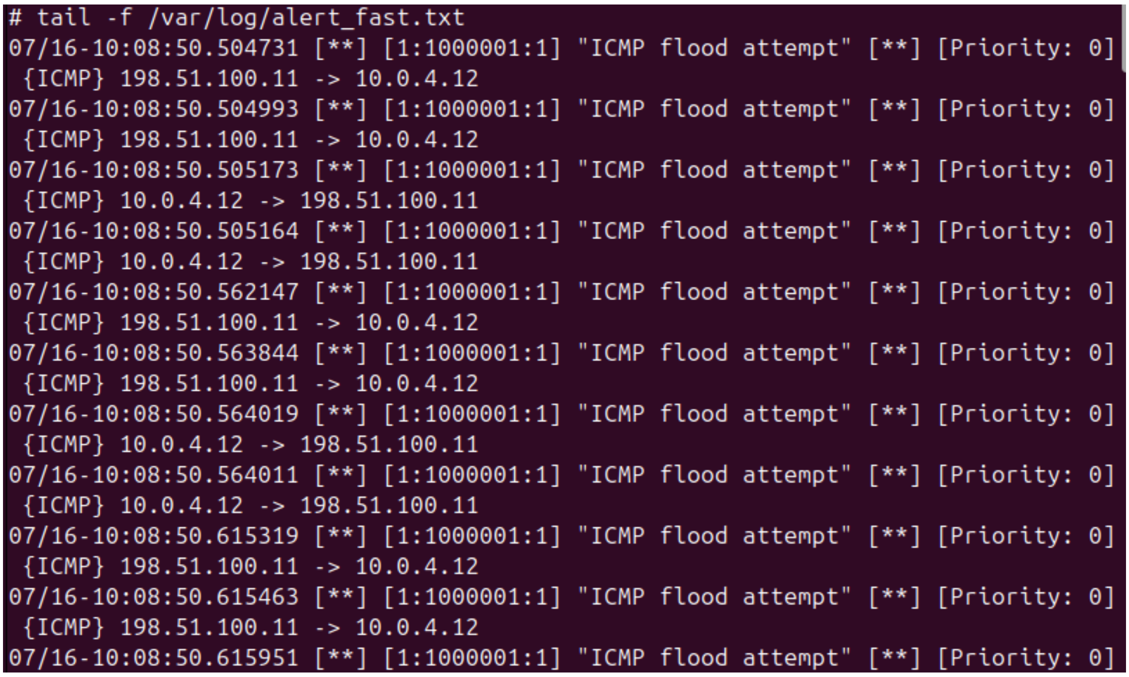
1472 bytes results in a total packet size of 1500 bytes, which is the Maximum Transmission Unit (MTU) for Ethernet networks which ensures that this packet is sent without any fragmentation.

Sending packets of this size to a server at very short intervals can have several severe consequences, larger packets consume significantly more bandwidth, quickly saturating network resources, they place a higher processing burden on the target server, as each packet must be processed individually, potentially exhausting CPU and memory resources and in the context of a distributed attack, where multiple systems send large ICMP packets simultaneously, the target server can become overwhelmed quickly, leading to service unavailability or even complete system failure. Furthermore, this highlights how even legacy attacks such as ICMP floods, when properly orchestrated, can remain highly effective and dangerous in modern network environments.

In 4.10, it is possible to observe the detection of the ICMP flood attack by Snort. The traffic_monitor, strategically placed in front of the attacking node 4.6, is responsible for inspecting all outbound traffic generated by the host at IP address 198.51.100.11. Since ICMP packet sent are more than 10 and they come from the same IP source, the custom Snort rule is triggered, resulting in the generation of an alert.

The alert message contains three critical pieces of information that are later extracted by the Sentinel Policy Extractor (SPE) for the reconfiguration process. These are:

- Source IP address of the malicious traffic (198.51.100.11).



```
# tail -f /var/log/alert_fast.txt
07/16-10:08:50.504731 [**] [1:1000001:1] "ICMP flood attempt" [**] [Priority: 0]
 {ICMP} 198.51.100.11 -> 10.0.4.12
07/16-10:08:50.504993 [**] [1:1000001:1] "ICMP flood attempt" [**] [Priority: 0]
 {ICMP} 198.51.100.11 -> 10.0.4.12
07/16-10:08:50.505173 [**] [1:1000001:1] "ICMP flood attempt" [**] [Priority: 0]
 {ICMP} 10.0.4.12 -> 198.51.100.11
07/16-10:08:50.505164 [**] [1:1000001:1] "ICMP flood attempt" [**] [Priority: 0]
 {ICMP} 10.0.4.12 -> 198.51.100.11
07/16-10:08:50.562147 [**] [1:1000001:1] "ICMP flood attempt" [**] [Priority: 0]
 {ICMP} 198.51.100.11 -> 10.0.4.12
07/16-10:08:50.563844 [**] [1:1000001:1] "ICMP flood attempt" [**] [Priority: 0]
 {ICMP} 198.51.100.11 -> 10.0.4.12
07/16-10:08:50.564019 [**] [1:1000001:1] "ICMP flood attempt" [**] [Priority: 0]
 {ICMP} 10.0.4.12 -> 198.51.100.11
07/16-10:08:50.564011 [**] [1:1000001:1] "ICMP flood attempt" [**] [Priority: 0]
 {ICMP} 10.0.4.12 -> 198.51.100.11
07/16-10:08:50.615319 [**] [1:1000001:1] "ICMP flood attempt" [**] [Priority: 0]
 {ICMP} 198.51.100.11 -> 10.0.4.12
07/16-10:08:50.615463 [**] [1:1000001:1] "ICMP flood attempt" [**] [Priority: 0]
 {ICMP} 198.51.100.11 -> 10.0.4.12
07/16-10:08:50.615951 [**] [1:1000001:1] "ICMP flood attempt" [**] [Priority: 0]
```

Figure 4.10. IDS alert

- Destination IP address targeted by the attack (10.0.4.12).
- Layer 4 protocol used for the communication (ICMP).

These parameters form the basis of the new Network Security Requirement (NSR) that will be used to initiate the security reaction mechanism.

4.1.6 Update of the topology and new requirement

Following the detection of the attack, the Intrusion Detection System (IDS) generates an alert message, as shown in figure 4.10. This alert contains critical information that is subsequently processed by the Security Policy Extractor (SPE), which as said before, is responsible for analyzing the content of the alert and translating it into a formal Network Security Requirement (NSR) extracting the crucial information.

In this specific case, the SPE extracts the following requirement:

Listing 4.6. Extracted Requirement

```xml
<PropertyDefinition>
    <Property graph="0" name="IsolationProperty" src="198.51.100.11"
    dst="10.0.4.12" lv4proto="OTHER"/>
</PropertyDefinition>
```

This newly extracted requirement is classified as an Isolation Requirement, aiming to prohibit ICMP communications between the source host 198.51.100.11 and the destination server 10.0.4.12. The protocol field is specified as *OTHER*, which, as previously discussed, includes all non-TCP/UDP protocols, such as ICMP. This requirement conflicts with one already present in the current NSR set, as shown in Listing 4.3:

Listing 4.7. Conflicting requirement

```
<Property graph="0" name="ReachabilityProperty" src="198.51.100.11"
    dst="10.0.4.12" lv4proto="OTHER"/>
```

Due to this conflict, the Conflicting Policy Merger (CPM) module is triggered to resolve the inconsistency and, for this reason, it replaces the existing requirement with the newly generated isolation requirement. This is an exact match so the behaviour is to remove the reachability requirement completely as explained in this subsection 2.3.2. The result is a revised NSR configuration that enforces the blocking of ICMP traffic between the attacker and the target server.

Listing 4.8. NSR of the newly generated XML file

```
<PropertyDefinition>
    <Property name="ReachabilityProperty" graph="0" src="10.0.1.11"
    dst="10.0.4.11" lv4proto="TCP" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.4.11"
        dst="10.0.1.11" lv4proto="TCP" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.1.11"
        dst="10.0.1.10" lv4proto="TCP" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.1.10"
        dst="10.0.1.11" lv4proto="TCP" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.1.11"
        dst="10.0.5.10" lv4proto="OTHER" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.5.10"
        dst="10.0.1.11" lv4proto="OTHER" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.1.11"
        dst="10.0.4.10" lv4proto="TCP" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.4.10"
        dst="10.0.1.11" lv4proto="TCP" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.1.11"
        dst="10.0.4.12" lv4proto="TCP" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.4.12"
        dst="10.0.1.11" lv4proto="TCP" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.1.11"
        dst="10.0.5.20" lv4proto="OTHER" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.5.20"
        dst="10.0.1.11" lv4proto="OTHER" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.1.10"
        dst="10.0.4.11" lv4proto="TCP" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.4.11"
        dst="10.0.1.10" lv4proto="TCP" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.1.10"
        dst="10.0.5.10" lv4proto="OTHER" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.5.10"
        dst="10.0.1.10" lv4proto="OTHER" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.1.10"
        dst="10.0.4.10" lv4proto="TCP" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.4.10"
        dst="10.0.1.10" lv4proto="TCP" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.1.10"
        dst="10.0.4.12" lv4proto="TCP" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.4.12"
        dst="10.0.1.10" lv4proto="TCP" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.1.10"
        dst="10.0.5.20" lv4proto="OTHER" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.5.20"
        dst="10.0.1.10" lv4proto="OTHER" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.5.20"
        dst="10.0.4.10" lv4proto="TCP" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.4.10"
        dst="10.0.5.20" lv4proto="TCP" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.5.20"
        dst="10.0.4.12" lv4proto="TCP" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.4.12"
        dst="10.0.5.20" lv4proto="TCP" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.5.20"
        dst="10.0.4.11" lv4proto="TCP" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.4.11"
        dst="10.0.5.20" lv4proto="TCP" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.5.10"
        dst="10.0.4.10" lv4proto="TCP" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.4.10"
        dst="10.0.5.10" lv4proto="TCP" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.5.10"
        dst="10.0.4.12" lv4proto="TCP" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.4.12"
        dst="10.0.5.10" lv4proto="TCP" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.5.10"
        dst="10.0.4.11" lv4proto="TCP" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.4.11"
        dst="10.0.5.10" lv4proto="TCP" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.4.12"
        dst="198.51.100.11" lv4proto="OTHER" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="198.51.100.11"
        dst="10.0.5.20" lv4proto="OTHER" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.5.20"
        dst="198.51.100.11" lv4proto="OTHER" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="198.51.100.11"
        dst="10.0.5.10" lv4proto="OTHER" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.5.10"
        dst="198.51.100.11" lv4proto="OTHER" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="198.51.100.11"
        dst="10.0.4.11" lv4proto="OTHER" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.4.11"
        dst="198.51.100.11" lv4proto="OTHER" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="198.51.100.11"
        dst="10.0.4.10" lv4proto="OTHER" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.4.10"
        dst="198.51.100.11" lv4proto="OTHER" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="198.51.100.11"
        dst="10.0.4.12" lv4proto="OTHER" isSat="true"/>
</PropertyDefinition>
<InitialProperty>
    <Property name="ReachabilityProperty" graph="0" src="10.0.1.11"
    dst="10.0.4.11" lv4proto="TCP" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.4.11"
        dst="10.0.1.11" lv4proto="TCP" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.1.11"
        dst="10.0.1.10" lv4proto="TCP" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.1.10"
        dst="10.0.1.11" lv4proto="TCP" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.1.11"
        dst="10.0.5.10" lv4proto="OTHER" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.5.10"
        dst="10.0.1.11" lv4proto="OTHER" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.1.11"
        dst="10.0.4.10" lv4proto="TCP" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.4.10"
        dst="10.0.1.11" lv4proto="TCP" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.1.11"
        dst="10.0.4.12" lv4proto="TCP" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.4.12"
        dst="10.0.1.11" lv4proto="TCP" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.1.11"
        dst="10.0.5.20" lv4proto="OTHER" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.5.20"
        dst="10.0.1.11" lv4proto="OTHER" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.1.10"
        dst="10.0.4.11" lv4proto="TCP" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.4.11"
        dst="10.0.1.10" lv4proto="TCP" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.1.10"
        dst="10.0.5.10" lv4proto="OTHER" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.5.10"
        dst="10.0.1.10" lv4proto="OTHER" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.1.10"
        dst="10.0.4.10" lv4proto="TCP" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.4.10"
        dst="10.0.1.10" lv4proto="TCP" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.1.10"
        dst="10.0.4.12" lv4proto="TCP" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.4.12"
        dst="10.0.1.10" lv4proto="TCP" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.1.10"
        dst="10.0.5.20" lv4proto="OTHER" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.5.20"
        dst="10.0.1.10" lv4proto="OTHER" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.5.20"
        dst="10.0.4.10" lv4proto="TCP" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.4.10"
        dst="10.0.5.20" lv4proto="TCP" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.5.20"
        dst="10.0.4.12" lv4proto="TCP" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.4.12"
        dst="10.0.5.20" lv4proto="TCP" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.5.20"
        dst="10.0.4.11" lv4proto="TCP" isSat="true"/>
    <Property name="IsolationProperty" graph="0" src="10.0.4.11"
        dst="10.0.5.20" lv4proto="TCP" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.5.10"
        dst="10.0.4.10" lv4proto="TCP" isSat="true"/>
    <Property name="ReachabilityProperty" graph="0" src="10.0.4.10"
        dst="10.0.5.10" lv4proto="TCP" isSat="true"/>
```

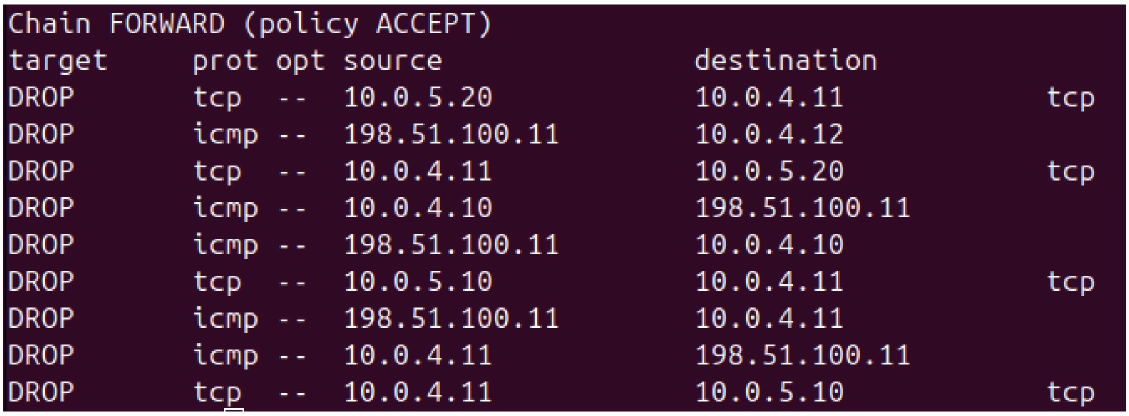
```
      <Property name="ReachabilityProperty" graph="0" src="10.0.5.10"
          dst="10.0.4.12" lv4proto="TCP" isSat="true"/>
      <Property name="ReachabilityProperty" graph="0" src="10.0.4.12"
          dst="10.0.5.10" lv4proto="TCP" isSat="true"/>
      <Property name="IsolationProperty" graph="0" src="10.0.5.10"
          dst="10.0.4.11" lv4proto="TCP" isSat="true"/>
      <Property name="IsolationProperty" graph="0" src="10.0.4.11"
          dst="10.0.5.10" lv4proto="TCP" isSat="true"/>
      <Property name="ReachabilityProperty" graph="0" src="198.51.100.11"
          dst="10.0.4.12" lv4proto="OTHER" isSat="true"/>
      <Property name="ReachabilityProperty" graph="0" src="10.0.4.12"
          dst="198.51.100.11" lv4proto="OTHER" isSat="true"/>
      <Property name="ReachabilityProperty" graph="0" src="198.51.100.11"
          dst="10.0.5.20" lv4proto="OTHER" isSat="true"/>
      <Property name="ReachabilityProperty" graph="0" src="10.0.5.20"
          dst="198.51.100.11" lv4proto="OTHER" isSat="true"/>
      <Property name="ReachabilityProperty" graph="0" src="198.51.100.11"
          dst="10.0.5.10" lv4proto="OTHER" isSat="true"/>
      <Property name="ReachabilityProperty" graph="0" src="10.0.5.10"
          dst="198.51.100.11" lv4proto="OTHER" isSat="true"/>
      <Property name="IsolationProperty" graph="0" src="198.51.100.11"
          dst="10.0.4.11" lv4proto="OTHER" isSat="true"/>
      <Property name="IsolationProperty" graph="0" src="10.0.4.11"
          dst="198.51.100.11" lv4proto="OTHER" isSat="true"/>
      <Property name="IsolationProperty" graph="0" src="198.51.100.11"
          dst="10.0.4.10" lv4proto="OTHER" isSat="true"/>
      <Property name="IsolationProperty" graph="0" src="10.0.4.10"
          dst="198.51.100.11" lv4proto="OTHER" isSat="true"/>
  </InitialProperty>
```

This XML configuration file, excluding the node definitions, which remain unchanged, contains two essential XML elements: *InitialProperty* and *PropertyDefinition*. The InitialProperty section retains the original set of NSRs, while PropertyDefinition includes both the updated isolation requirement extracted by the SPE and all previously non-conflicting requirements that must be be preserved. This file is then provided as input to React-VEREFOO which identifies the presence of a new property not previously enforced and as a result, the system triggers a reconfiguration process that focuses solely on the nodes traversed by the communication flow associated with the new requirement. In this case, the nodes identified for potential modification include:

- The forwarder at 33.33.33.31
- The firewall at 1.0.0.7
- The forwarder at 1.0.0.10

Since React-VEREFOO'intervention must be as small as possible in order to make modifications only to the smallest number of nodes to enforce the new requirement, only firewall 1.0.0.7 is reconfigured, as this modification alone is enough to block the malicious communication, ensuring compliance with the updated isolation requirement.

Figure 4.11 shows the updated firewall rules after reconfiguration and, when compared to the initial configuration shown in Figure 4.8, it is evident that a



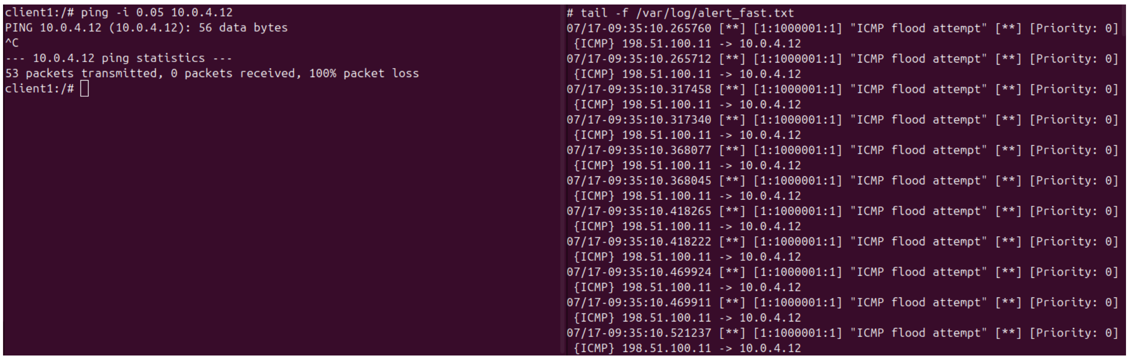
```
Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
DROP       tcp  --  10.0.5.20            10.0.4.11            tcp
DROP       icmp --  198.51.100.11        10.0.4.12
DROP       tcp  --  10.0.4.11            10.0.5.20            tcp
DROP       icmp --  10.0.4.10            198.51.100.11
DROP       icmp --  198.51.100.11        10.0.4.10
DROP       tcp  --  10.0.5.10            10.0.4.11            tcp
DROP       icmp --  198.51.100.11        10.0.4.11
DROP       icmp --  10.0.4.11            198.51.100.11
DROP       tcp  --  10.0.4.11            10.0.5.10            tcp
```

Figure 4.11. Updated Firewall rules (before attack)

new rule has been introduced, explicitly denying ICMP traffic from 198.51.100.11 to 10.0.4.12, thus effectively mitigating the attack while preserving all legitimate network functionalities.

4.1.7 Results and observations

This section presents the key findings derived from the designed demonstration and discusses the potential and benefits of adopting this framework as a whole.

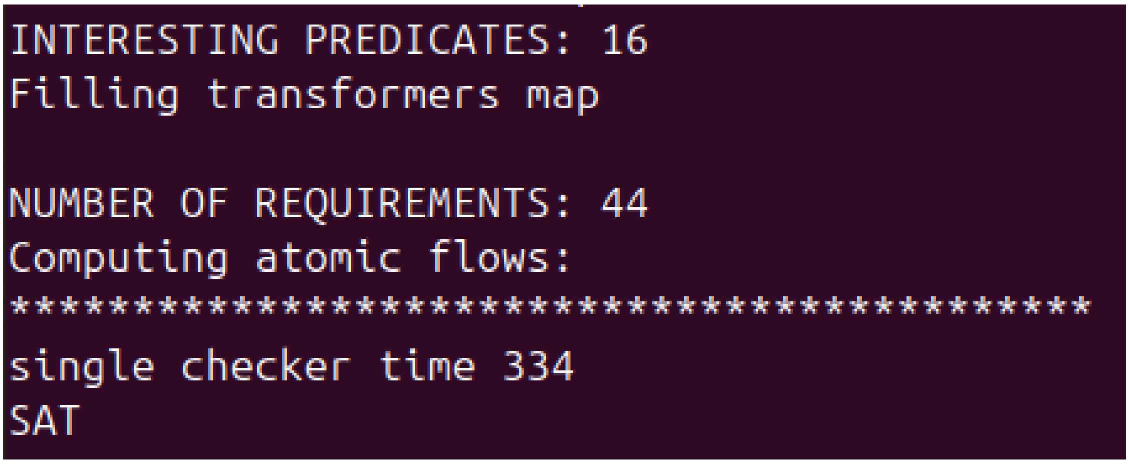


```
client1:/# ping -i 0.05 10.0.4.12
PING 10.0.4.12 (10.0.4.12): 56 data bytes
^C
--- 10.0.4.12 ping statistics ---
53 packets transmitted, 0 packets received, 100% packet loss
client1:/# 
```

```
# tail -f /var/log/alert_fast.txt
07/17-09:35:10.265760 [**] [1:1000001:1] "ICMP flood attempt" [**] [Priority: 0]
 {ICMP} 198.51.100.11 -> 10.0.4.12
07/17-09:35:10.265712 [**] [1:1000001:1] "ICMP flood attempt" [**] [Priority: 0]
 {ICMP} 198.51.100.11 -> 10.0.4.12
07/17-09:35:10.317458 [**] [1:1000001:1] "ICMP flood attempt" [**] [Priority: 0]
 {ICMP} 198.51.100.11 -> 10.0.4.12
07/17-09:35:10.317340 [**] [1:1000001:1] "ICMP flood attempt" [**] [Priority: 0]
 {ICMP} 198.51.100.11 -> 10.0.4.12
07/17-09:35:10.368077 [**] [1:1000001:1] "ICMP flood attempt" [**] [Priority: 0]
 {ICMP} 198.51.100.11 -> 10.0.4.12
07/17-09:35:10.368045 [**] [1:1000001:1] "ICMP flood attempt" [**] [Priority: 0]
 {ICMP} 198.51.100.11 -> 10.0.4.12
07/17-09:35:10.418265 [**] [1:1000001:1] "ICMP flood attempt" [**] [Priority: 0]
 {ICMP} 198.51.100.11 -> 10.0.4.12
07/17-09:35:10.418222 [**] [1:1000001:1] "ICMP flood attempt" [**] [Priority: 0]
 {ICMP} 198.51.100.11 -> 10.0.4.12
07/17-09:35:10.469924 [**] [1:1000001:1] "ICMP flood attempt" [**] [Priority: 0]
 {ICMP} 198.51.100.11 -> 10.0.4.12
07/17-09:35:10.469911 [**] [1:1000001:1] "ICMP flood attempt" [**] [Priority: 0]
 {ICMP} 198.51.100.11 -> 10.0.4.12
07/17-09:35:10.521237 [**] [1:1000001:1] "ICMP flood attempt" [**] [Priority: 0]
 {ICMP} 198.51.100.11 -> 10.0.4.12
```

Figure 4.12. Proof of the succesfull attack mitigation

Following the merging of policies and the subsequent firewall updates, the ICMP flood attack was successfully mitigated, as illustrated in Figure 4.12. The figure clearly shows that, although the Traffic_Monitor continues to generate alerts, due to its position upstream from the attacker node (198.51.100.11), the malicious ICMP packets are effectively blocked by the firewall located at 1.0.0.7 which is a direct consequence of the updated rule applied to that specific firewall. On the left-hand side of the figure, the ping statistics confirm this outcome: when packet are sent in a low interval from the same source, the packet loss reaches 100%, which is a clear indication that the attack has been completely neutralized before reaching its intended victim (10.0.4.12).

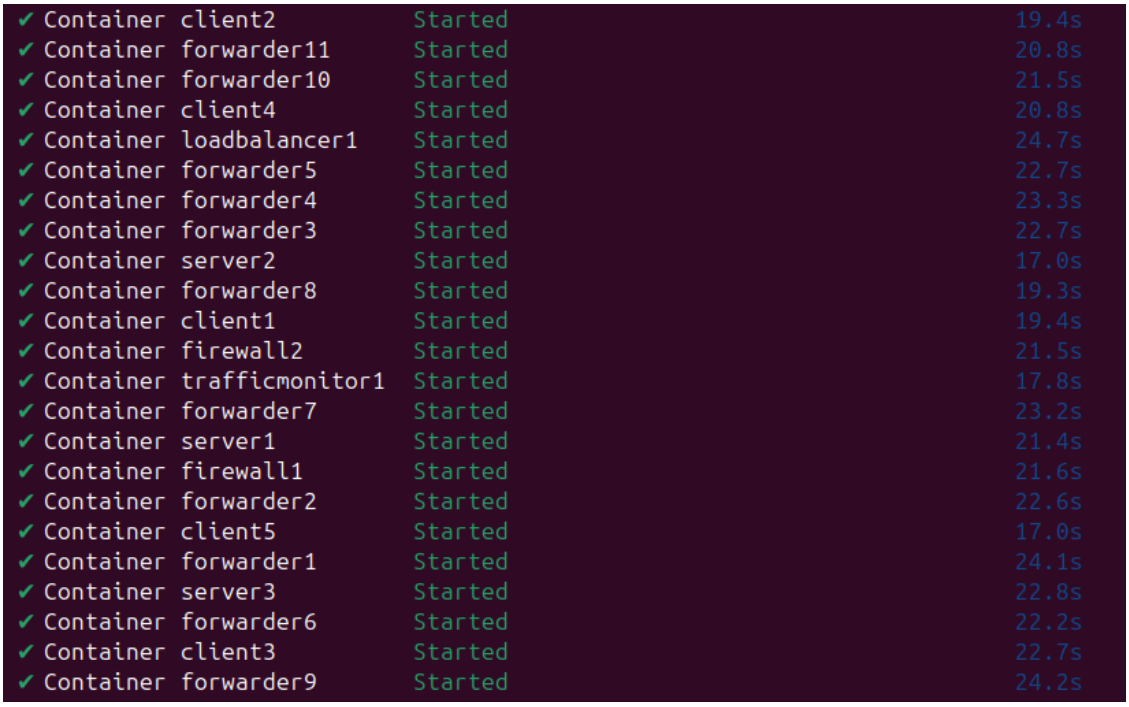


```
INTERESTING PREDICATES: 16
Filling transformers map

NUMBER OF REQUIREMENTS: 44
Computing atomic flows:
********************************************
single checker time 334
SAT
```

Figure 4.13. VEREFOO computation Time

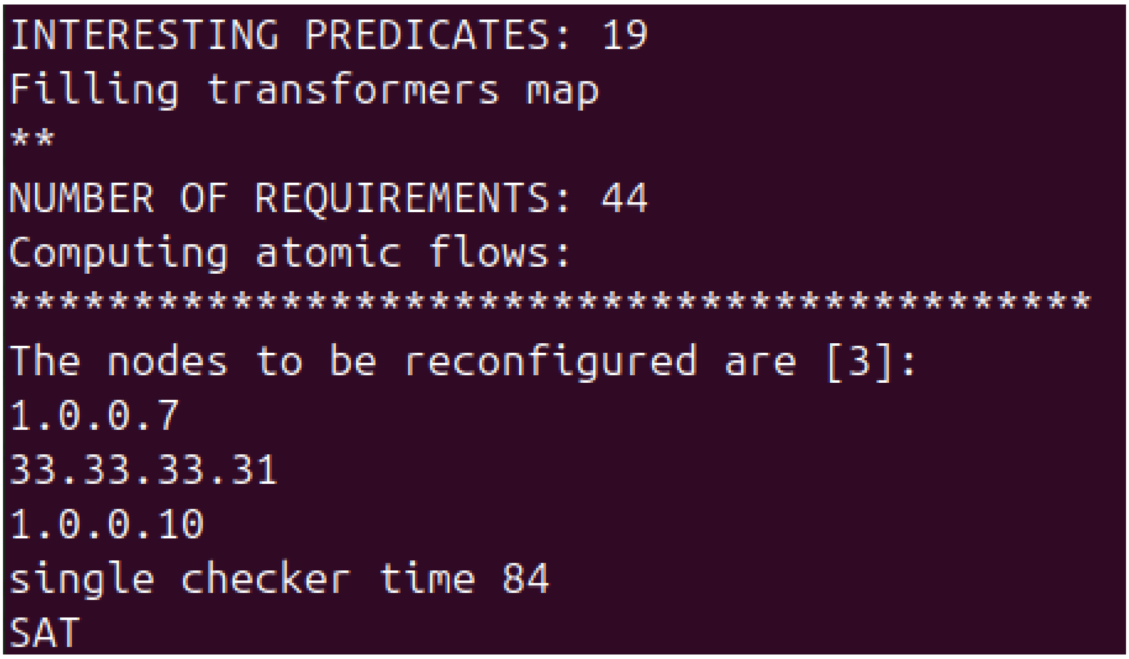
In terms of performance, the framework demonstrated excellent efficiency throughout both the initial deployment and the reactive update phase. When the topology and the initial NSR set were first submitted to VEREFOO 4.13, the entire computation and configuration generation process took approximately 334 milliseconds. Across multiple tests, using varying NSR configurations and topologies of similar size, the synthesis time consistently remained below one second or two seconds.



```
✔ Container client2          Started                                    19.4s
✔ Container forwarder11      Started                                    20.8s
✔ Container forwarder10      Started                                    21.5s
✔ Container client4          Started                                    20.8s
✔ Container loadbalancer1    Started                                    24.7s
✔ Container forwarder5       Started                                    22.7s
✔ Container forwarder4       Started                                    23.3s
✔ Container forwarder3       Started                                    22.7s
✔ Container server2          Started                                    17.0s
✔ Container forwarder8       Started                                    19.3s
✔ Container client1          Started                                    19.4s
✔ Container firewall2        Started                                    21.5s
✔ Container trafficmonitor1  Started                                    17.8s
✔ Container forwarder7       Started                                    23.2s
✔ Container server1          Started                                    21.4s
✔ Container firewall1        Started                                    21.6s
✔ Container forwarder2       Started                                    22.6s
✔ Container client5          Started                                    17.0s
✔ Container forwarder1       Started                                    24.1s
✔ Container server3          Started                                    22.8s
✔ Container forwarder6       Started                                    22.2s
✔ Container client3          Started                                    22.7s
✔ Container forwarder9       Started                                    24.2s
```

Figure 4.14. Docker deployment time

The deployment of the virtual network using Docker containers was the most time-consuming phase 4.14, taking approximately 30 seconds to complete. However, this is largely dependent on the underlying hardware and system resources available. It is important to note that node deployments are handled in parallel, and the overall deployment time is generally constrained by the most resource intensive container, typically forwarders with numerous neighboring nodes or firewalls due to their rules configuration. However, the first deployment of the virtual network with nothing installed (Snort, iptable, etc.) is a large time-consuming task since everything must be firstly downloaded and then installed.



```
INTERESTING PREDICATES: 19
Filling transformers map
**
NUMBER OF REQUIREMENTS: 44
Computing atomic flows:
********************************************
The nodes to be reconfigured are [3]:
1.0.0.7
33.33.33.31
1.0.0.10
single checker time 84
SAT
```

Figure 4.15. React-VEREFOO computation Time

The Security Policy Extractor (SPE) and Conflicting Policy Merger (CPM) components also demonstrated high performance. In all tests, these components completed their tasks in under one second, with most executions being nearly instantaneous. The computation time required by React-VEREFOO to determine the minimal reconfiguration 4.15 was measured at 84 milliseconds, reinforcing the system's capability to provide a rapid reaction to detected threats.

One of the most significant advantages of React-VEREFOO is that it updates only the nodes affected by the newly detected communication path. In the scenario demonstrated, the system correctly identified 1.0.0.7 as the only component requiring reconfiguration, and the update was executed in place, without restarting or redeploying the entire network. This minimizes disruption and ensures continuity of service. VEREFOO itself would have required the redeployment of the whole network at least doubling the required time of react-VEREFOO. However, more complex situations may arise where the network must be restructured, new firewalls need to be instantiated, or existing ones relocated or removed and in such cases, the reconfiguration process might take more time. Nonetheless, even in these scenarios, React-VEREFOO is designed to compute the most efficient and resource aware reconfiguration plan, ensuring an optimal trade-off between responsiveness and correctness.

Finally, the proposed framework demonstrates strong potential for defending against a wide range of cyberattacks. In the current demonstration, a single custom rule was implemented in Snort3; however, the framework can be easily extended by incorporating Snort3's default rule sets along with additional custom rules. This makes the system highly adaptable and capable of addressing a broader spectrum of attack scenarios with minimal manual effort.

This approach also aligns well with the findings presented in the Verizon DBIR Cyber Threat Report [16]. As highlighted at the beginning of this research, a significant proportion of cyberattacks are due to human intervention, which, in complex networked environments, miscommunication between network designers and network manager can often introduce vulnerabilities that are later exploited by attackers. The proposed framework mitigates this risk by substantially reducing the degree of manual intervention required throughout the network defense lifecycle. While human involvement is still necessary during the design of the network topology, defining Snort rules, and verifying whether alerts are true or false positives, the framework introduces automation in critical areas such as policy extraction, conflict resolution, and dynamic reconfiguration.

4.1.8 Graphical User Interface

A graphical interface for the demonstrator was implemented as a web application following a client–server architecture. The front-end client is written in React and exposes a user-oriented interface that mirrors the features of the original Python demonstrator while the back-end server is implemented with Flask and provides a small REST API that the React client consumes. The back-end is responsible for two complementary tasks which are: exposing the sequence of demonstration steps to the UI and executing and forwarding shell commands to the environment, for instance, open an interactive terminal on a node of the virtual topology.

The server-side logic is split across two Python modules which are *demo_steps.py* and *server.py*. The first one contains a single function *get_demo_steps()* which returns an ordered array of step descriptors that are sent to the client each time the *Next Step* button is pressed. Each descriptor typically includes a human-readable message, references to illustrative images, and any metadata the client needs to render that step, like titles, step's ids, and optional parameters. The second python module is the Flask application that implements the REST API consumed by the React client that acts as the bridge between the UI and the local runtime environment (Docker containers, shell commands).

The structure of *server.py* python module is the following:

- **get_demo_steps()**: imports the helper from demo_steps.py to provide to the client messages, images and metadata that are used to enrich with information every step of the demonstrator. It calls get_demo_steps() and returns its result as JSON via jsonify. If the response is 200 OK it retrieves a JSON array containing the steps informations. This endpoint is read-only and deterministic because it exposes static content prepared for the demonstrator.

- **POST /api/run-command**: This REST API is used to execute shell commands on the host and return *stdout*, *stderr* and the process return code. The request payload is a JSON which is the command that needs to be executed that are sent by the client and obtained thanks to the previous call to demo_steps.py.
- **/api/open-terminal (POST)**: This REST API opens a new terminal attached to a specific Docker container. This REST API activates whenever the user click on one of the node in the topology. The request provides to the server a JSON object with the field *node* that is the name of the docker container to be opened and a small JSON message is provided as output to confirm that the terminal have been opened. When the terminal to the Docker container is opened the user can execute different commands and can manually perform the attack explained in this research.

The graphical interface of the demonstrator was implemented as a React client, designed to provide users with an interactive and intuitive view of the workflow. The interface is structured into two main sections: the left side displays the network topology while the right side provides contextual information, messages, commands, and output logs.

In the majority of the steps the right side is composed by a static image to always provide to users a graphical information related on the topology created for this research. When the step related to the manual execution of the attack is reached the topology becomes interactive and each network element (e.g., hosts, firewalls, load balancers, or servers) is represented by an icon, and clicking on a node triggers the opening of a dedicated terminal session through the server-side API */api/open-terminal*. This allows the user to directly inspect the configuration of the selected component, such as firewall rules, IDS alerts and perform manually the attack.

On the right-hand side, the interface contains several functional panels. A dedicated *Messages* panel displays textual information that guides the user through the steps of the demonstration. Below it, a *Next Step* button allows the user to progress through the predefined workflow in a sequential manner. The *Commands* panel presents the list of shell commands associated with the current step, with the option to execute all of them automatically via a *Run All* button that triggers the REST API */api/run-command*. Finally, the *Output* panel dynamically displays the results of executed commands, making it possible to observe the system's responses and the changes occurring in the underlying topology in real-time.

Chapter 5

Conclusions & future works

This thesis has focused on the development of a demonstrator and the refinement of the VEREFOO framework. The presented demonstrator provides a comprehensive and complete overview of the operation of the framework, showcasing each stage of its workflow and all internal mechanisms involved in its automated response to cyber threats. Through this implementation, the framework proves capable of reacting both efficiently and rapidly to any attack detectable by an Intrusion Detection System (IDS) by dynamically reconfiguring the network and updating firewall rules in real time, the system is also able to isolate compromised nodes and prevent attack propagation, thus offering strong resilience against a wide variety of threats across different network environments.

One of the most significant challenges addressed in this work was modifying the latest version of the framework to ensure compatibility and correct functionality across the diverse set of operations and input requests required by the demonstrator. Equally pivotal, was the design of a network topology that closely reflects real-world infrastructure, and the selection of a realistic attack scenario, specifically an ICMP flood, that could be effectively deployed and mitigated within this context. These elements contributed to demonstrating the practical viability and robustness of the framework in a controlled but realistic setting.

5.0.1 Limitations & Future Work

Although the current implementation of the framework is functional and shows promising results, several areas remain open for enhancement and expansion. At this stage, the framework supports only Snort3 as the IDS, for this reason, extending support to additional intrusion detection systems, such as FortiGuard or Suricata, would increase flexibility and interoperability, allowing the system to be adopted in a broader range of real-world environments and settings.

Similarly, the firewall configuration module is currently based solely on iptables, for this reason, incorporating support for more advanced or specialized technologies, such as nftables, Open vSwitch, or eBPF—could offer improved performance, scalability, and integration with emerging software-defined networking (SDN) paradigms.

Another enhancement that can be done could be related to the transport protocols implemented. At this stage the OTHER level four protocol implemented in the NSR rules means only ICMP, but it can be expanded to support a wider range of transport protocols.

While the framework has shown effective results in networks of moderate size, further testing on larger and more complex topologies would be essential to assess its scalability and efficiency in enterprise scale or cloud native deployments. Evaluating how well the system performs under high traffic volumes and simultaneous attacks would provide more insight into its limitations and resilience.

Additionally, automation of further stages in the detection and mitigation pipeline, such as automatic classification of alerts as true or false positives using machine learning models, can reduce even further human involvement, thereby enhancing the framework's autonomy and applicability in production environments.

In conclusion, this work not only validates the capabilities of VEREFOO as a reactive, policy-driven network security framework but also lays a solid foundation for future research and development aimed at building adaptive, intelligent, and fully autonomous defense systems for modern network infrastructures.

Bibliography

[1] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, “Introducing programmability and automation in the synthesis of virtual firewall rules,” pp. 473–478, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9165434

[2] netscout, “Netscout ddos threat intelligence report,” July 2024 to December 2024. [Online]. Available: https://www.netscout.com/threatreport/global-highlights

[3] OWASP, “Owasp top ten,” 2021. [Online]. Available: https://owasp.org/www-project-top-ten/

[4] J. Jabez and B. Muthukumar, “Intrusion detection system (ids): Anomaly detection using outlier detection approach,” *Procedia Computer Science*, vol. 48, pp. 338–346, 2015, international Conference on Computer, Communication and Convergence (ICCC 2015). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050915007000

[5] F. Pizzato, D. Bringhenti, R. Sisto, and F. Valenza, “Automatic and optimized firewall reconfiguration,” in *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, 2024, pp. 1–9. [Online]. Available: https://ieeexplore.ieee.org/document/10575212

[6] D. Bringhenti, F. Pizzato, R. Sisto, and F. Valenza, “A looping process for cyberattack mitigation,” in *2024 IEEE International Conference on Cyber Security and Resilience (CSR)*, 2024, pp. 276–281. [Online]. Available: https://ieeexplore.ieee.org/document/10679501

[7] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, “Automated firewall configuration in virtual networks,” *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 2, pp. 1559–1576, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/9737389

[8] M. Aljanabi, M. A. Ismail, R. Hasan, and J. Sulaiman, “Intrusion detection: A review,” *Mesopotamian Journal of Cyber Security*, vol. 2021, pp. 1–4, 01 2021. [Online]. Available: https://www.researchgate.net/publication/367569516_Intrusion_Detection_A_Review

[9] Q.-V. Dang, “Active learning for intrusion detection systems,” in *2020 RIVF International Conference on Computing and Communication Technologies (RIVF)*, 2020, pp. 1–3. [Online]. Available: https://ieeexplore.ieee.org/document/9140751

[10] Snort, “What is snort.” [Online]. Available: https://www.snort.org

[11] Docker, “What is a container?” [Online]. Available: https://www.docker.com/resources/what-container

[12] ——, “Docker engine security.” [Online]. Available: https://docs.docker.com/engine/security

[13] D. Bringhenti and F. Valenza, “A twofold model for vnf embedding and time-sensitive network flow scheduling,” *IEEE Access*, vol. 10, pp. 44 384–44 399, 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9762325

[14] D. Bringhenti, R. Sisto, and F. Valenza, “Security automation for multi-cluster orchestration in kubernetes,” in *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*, 2023, pp. 480–485. [Online]. Available: https://ieeexplore.ieee.org/document/10175419

[15] D. Bringhenti, S. Bussa, R. Sisto, and F. Valenza, “Atomizing firewall policies for anomaly analysis and resolution,” *IEEE Transactions on Dependable and Secure Computing*, vol. 22, no. 3, pp. 2308–2325, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10749982

[16] V. Business, “2022-dbir-data-breach-investigations-report,” Tech. Rep., 2022. [Online]. Available: https://www.verizon.com/business/resources/reports/2022-dbir-data-breach-investigations-report.pdf

[17] H. Abie, “An overview of firewall technologies,” 12 2000. [Online]. Available: https://www.researchgate.net/publication/2371491_An_Overview_of_Firewall_Technologies

[18] D. Bringhenti, R. Sisto, and F. Valenza, “A demonstration of verefoo: an automated framework for virtual firewall configuration,” in *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*, 2023, pp. 293–295. [Online]. Available: https://ieeexplore.ieee.org/document/10175442

[19] F. Valenza, C. Basile, D. Canavese, and A. Lioy, “Classification and analysis of communication protection policy anomalies,” *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2601–2614, 2017. [Online]. Available: https://ieeexplore.ieee.org/document/7967691

[20] L. de Moura and N. Bjørner, “Z3: an efficient smt solver,” vol. 4963, 04 2008, pp. 337–340. [Online]. Available: https://www.researchgate.net/publication/225142568_Z3_an_efficient_SMT_solver

[21] Y. Hobballah, “Automating the deployment of security functions in virtualized nerworks,” Master’s thesis, Politecnico di Torino, 2022. [Online]. Available: https://webthesis.biblio.polito.it/24644/

[22] L. Giglio, “Automatic security reaction in a virtualized environment,” Master’s thesis, Politecnico di Torino, 2024. [Online]. Available: https://webthesis.biblio.polito.it/31063/

[23] F. Pizzato, “Optimized and automatic firewall reconfiguration,” Master’s thesis, Politecnico di Torino, 2023. [Online]. Available: https://webthesis.biblio.polito.it/26915/

[24] D. Bringhenti, F. Pizzato, R. Sisto, and F. Valenza, “Autonomous attack mitigation through firewall reconfiguration,” *International Journal of Network Management*, vol. 35, 10 2024. [Online]. Available: https://www.researchgate.net/publication/385090245_Autonomous_Attack_Mitigation_Through_Firewall_Reconfiguration

[25] CISA, Cybersecurity Infrastructure Security Agency, “Defining insider threats.” [Online]. Available: https://www.cisa.gov/topics/physical-security/insider-threat-mitigation/defining-insider-threats

[26] D. Costa, "Cert definition of 'insider threat' - updated," Carnegie Mellon University, Software Engineering Institute's Insights (blog), Mar 2017. [Online]. Available: https://insights.sei.cmu.edu/blog/cert-definition-of-insider-threat-updated/

[27] Gurucul, "2024 report - insider threat," Tech. Rep., 2024. [Online]. Available: https://go1.gurucul.com/2024-insider-threat-report

[28] FBI - Federal Bureau of Investigation, "Trade secret theft," July 29, 2020. [Online]. Available: https://www.fbi.gov/news/stories/two-guilty-in-theft-of-trade-secrets-from-ge-072920

[29] N. Gupta, A. Jain, P. Saini, and V. Gupta, "Ddos attack algorithm using icmp flood," in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, 2016, pp. 4082–4084.