Politecnico di Torino

Master of science in Cybersecurity

Master Degree Thesis

A Docker-based Solution for Automated Firewall Update Management

Supervisors
prof. Daniele Brighenti
prof. Riccardo Sisto
prof. Fulvio Valenza

Candidate
Matteo Scursatone

Academic Year 2025-2026

Abstract

In complex network environments, balancing network functionality with robust security is a critical challenge. Relying only on manual intervention by security managers and network administrators is no longer feasible due to the scale of infrastructures and the speed of evolving threats. Consequently, there is a need for automated tools that can dynamically orchestrate security responses. This thesis presents a Docker-based demonstrator managed with a Graphical User Interface. The system is based on a pre-existing solution for automated firewall update management, which can schedule the reconfiguration steps required after an attack.

Contents

List of Figures

Listings

Chapter 1

Introduction

1.1 Thesis introduction

The evolution of modern computer networks, driven by virtualization paradigms such as Software-Defined Networking (SDN) and Network Functions Virtualization (NFV), has introduced unprecedented levels of scalability and flexibility. However, this growing size and complexity have rendered traditional, manual approaches to network security configuration highly impractical. When network components were static and under the direct control of administrators, manual configuration was sufficient. Today, the dynamic nature of next-generation networks demands a different approach.

To manage this complexity, automated policy-based management has gained significant traction. In these systems, administrators define high-level security requirements (e.g., specifying which potentially malicious traffic flows must be blocked), and an automated process computes the necessary low-level configurations for the network security functions. Packet filtering firewalls remain the most effective defense against a wide array of cyber attacks, much of the literature focuses on automating distributed firewall configurations calculating both the optimal allocation of firewall instances across a logical topology and their specific filtering rules.[1][2]

Contemporary cyber threats are increasingly characterized by their brevity and their capacity to rapidly mutate across multiple attack vectors. In light of these trends, there is the need for a mitigation strategy whose response times directly align with the accelerated pace of modern attacks. Such a system must be capable of reconfiguring the network infrastructure to dynamically adapt to highly volatile threat landscapes.

In this context various approaches have been proposed to automate the configuration of security such as [3][4], based on the idea of a looping process where the system continuously monitors the network, detects potential threats and applies the patches to mitigate the attack.

In this thesis a demonstrator is presented, based on a Docker environment, that integrates the automation of firewall configuration and the scheduling of mitigation steps in response to simulated attack scenarios. The scheduling of mitigation steps is based on pre-existing software, that aims to maximize the security of the

transient states during the update of the firewalls configuration in different nodes of the network. Establishing a safe scheduling of the configuration changes can significantly limit the number of unsecure states and decrease the time period where the network may be at risk.[5]

Moreover, the possibility to schedule the mitigation steps based on the impact of the attack allows to prioritize the mitigation efforts and allocate resources effectively, while ensuring that the most critical vulnerabilities are addressed first.

1.2 Network Security Automation

Modern computer networks have been facing a progressive evolution in the latest years. On one hand, network size is constantly increasing, due to the digitization of every activity, while on the other hand, the heterogeneity of functions and technologies exploited in building networked architectures is increasing.

Enforcing the desired security properties in modern computer networks is a troublesome task for security managers. The main reason is that the configuration of security functions (e.g., firewalls, anti-spam filters, etc.) is traditionally performed manually, with a trial-and-error approach; whenever an attack is detected, the configuration is modified accordingly. This work paradigm is not scalable, and it is prone to several errors due to the fallibility of humans.[6]

As stated by Verizon[7] in its 2024 Data Breach Investigations Report, the human element was a component of 68% of breaches. Moreover, the report highlights Errors have increased substantially this year and misconfigurations is seen in 10% of the breaches.

To address this issue, automation has been recently leveraged by research to improve the state of the art of network security configuration. The main goal is to provide as automatic as possible configuration of security services to minimize human intervention. An automated process can be also combined with optimization techniques to avoid unnecessary resource consumption and, with formal verification, to identify or prevent configuration mistakes.[8][2]

The main reason why the automation of network security configuration has became so relevant are:

- Increasing network size
- Increasing network complexity.
- Increasing network heterogeneity
- Trial-and-error manual configuration approaches
- Impact of security breaches

In an automatic system, the core principle is the minimization of human interventions: After the system receives an external input from a human being or from another system, it should be able to work without requiring other external

contributions. In network security, the introduction of automation represents a possible solution to human errors characterizing manual configuration of security functions.[3]

Network Functions Virtualization (NFV)[9] and Software-Defined Networking (SDN)[10] are two paradigms that are enabling the automation of network configuration due to their flexibility and agility in service function deployment, since each function is a software program running on a general-purpose server.

Low-level configuration of many Virtual Network Functions (VNFs) relies on specific parameters, whose values have to be selected and set manually by the network administrator and because of this, suitable automated software tools are required, enabling operators to check networks before service deployment, thus preventing the violation of security properties and service downtimes.

Based on paper like[11] there is also the possibility to automatically identify and removing all the anomalies appearing in the firewall policies, modelling rule conditions as complex predicates and splitting them into simpler ones. This simpler rules are called atomic rules and they are easier to analyze and manage.

In this context, VEREFOO (VErified REFinement and Optimized Orchestration) emerges as a framework designed to work with modern networking paradigms, such as SDN and NFV, to automate the configuration of security functions. Other than that, the solution found by VEREFOO is guaranteed to be optimal and formally correct thanks to the Z3 solver[12] which solves a problem framed internally by VEREFOO as a partial weighted Maximum Satisfiability Modulo Theories (MaxSMT) problem and provides formal guarantees for the firewall configuration correctness and minimizes the size of the firewall allocation scheme and rule set.[13]

Moreover, The redistribution of the security functions in the logical topology of the virtual network and their reconfiguration are now more frequent and happen in quite strict times and this high dynamism characterizing the management of modern computer networks has severely impacted the preservation of the security during the updates. When this configuration update aspects a distributed security function a transient starts from the moment the first configuration change is applied to when the last one has been applied. In-between the initial and last change, there are several intermediate states where the configuration of the distributed function has already been modified with respect to the initial one, but the update is not yet complete. The order in which the configuration changes are applied in this period of time might result in unsecure transient states.

An incorrect scheduling of the changes for the configuration of a distributed firewall might result in a provisional service interruption or, even worse, in a temporary breach which attackers might exploit, thus endangering the systems connected to the network.[5].

Subsequently, VEREFOO can be integrated with existing software that schedule the deployment or update of the firewall configuration in steps, starting from the one which mitigates the most impactful attack. This approach allows to prioritize the mitigation efforts and allocate resources effectively, while ensuring that the most critical vulnerabilities are addressed first.

1.3 Thesis description

This thesis addresses the challenge of automating security configuration in modern network environments, where manual intervention is increasingly inadequate due to the scale and complexity of infrastructures and the rapid evolution of cyber threats. The work presents a Docker-based demonstrator that integrates automated firewall configuration and scheduling of mitigation steps in response to simulated attack scenarios.

Chapter 1 introduces the context and motivations behind network security automation, highlighting the limitations of traditional manual configuration approaches and the need for automated, formally verified solutions in dynamic network environments. Moreover, it explores the state of the art in Network Security Automation, examining how paradigms such as Network Functions Virtualization (NFV) and Software-Defined Networking (SDN) enable automation of network configuration. The chapter discusses the critical importance of automation in addressing human errors, which are responsible for a significant percentage of security breaches, and presents the challenges of maintaining security during configuration updates, particularly the risks associated with transient states in distributed firewall systems.

Chapter 2 defines the specific objectives of the thesis. It identifies the three main components of the demonstrator and articulates the concrete goals for each: the client application, the server application and the external tools.

Chapter 3 provides a comprehensive overview of the tools and technologies employed in the demonstrator. It begins with Docker, explaining containers and their security features, then presents VEREFOO (VErified REFinement and Optimized Orchestration), a framework that combines formal verification and optimization for distributed firewall configuration using Z3 solver and MaxSMT problems. The chapter also covers ReactFlow, a customizable React component for building node-based diagrams used to visualize network topologies, and Elkjs, the Eclipse Layout Kernel for automatic graph layouting.

Chapter 4 describes the design and implementation of the demonstrator in detail. The architecture integrates five main components: a client application with GUI, a Python-based server, VEREFOO, a scheduling backend, and a Docker environment. The demonstrator is based on a SCADA topology inspired by the Texas 2000 infrastructure and simulates three distinct attack scenarios: Denial of Service, data exfiltration, and infiltration/data injection. The GUI, built with React, comprises five major components for step navigation, network topology visualization, message and command display, attack management, and 3D impact matrix visualization. The attack impact is evaluated using three metrics (Asset Value, Vulnerability Severity, and Threat Strength) combined through a weighted average, enabling the scheduling backend to prioritize mitigation steps. The chapter demonstrates the complete workflow from topology definition through VEREFOO validation, Docker deployment, attack simulation, impact evaluation, and sequential application of mitigation steps.

Chapter 5 concludes the thesis by summarizing the contributions and discussing limitations and future work.

Chapter 2

Thesis Objective

This chapter outlines the specific and concrete objectives of the thesis. While the introduction provides a broader context for network security automation challenges, this chapter articulates the concrete goals pursued through the development of the demonstrator and the work described in the following chapters.

As discussed in the introduction, the rapid evolution of cyber threats and the increasing complexity of modern network infrastructures necessitate automated approaches to security configuration. The primary objective of this thesis is to develop and validate a practical demonstrator that automates the configuration and scheduling of security updates in response to simulated attack scenarios. This demonstrator integrates automated firewall configuration with dynamic scheduling of mitigation steps, providing a proof-of-concept for coordinated security response in virtualized network environments.

The demonstrator has 3 main part as will be described in Chapter 4 with figure 4.1:

- Client Application with GUI
- Server Application
- External Tools (VEREFOO, Scheduling Backend, Docker)

2.1 Client application

The first objectives of the thesis include recreate the Client application with GUI starting from a previous prototype. To start with the older GUI was built in React and provided static visualizations of the network topology and attack scenarios. The network visualization built in this thesis is designed to be more interactive and user-friendly. Moreover it has been tested with different topologies in order to be integrated in a more general framework and reusable in different contexts. Thanks to the architecture the topology is given from the server and the GUI is able to visualize very different topology with large number of nodes and edges without any change in the code.

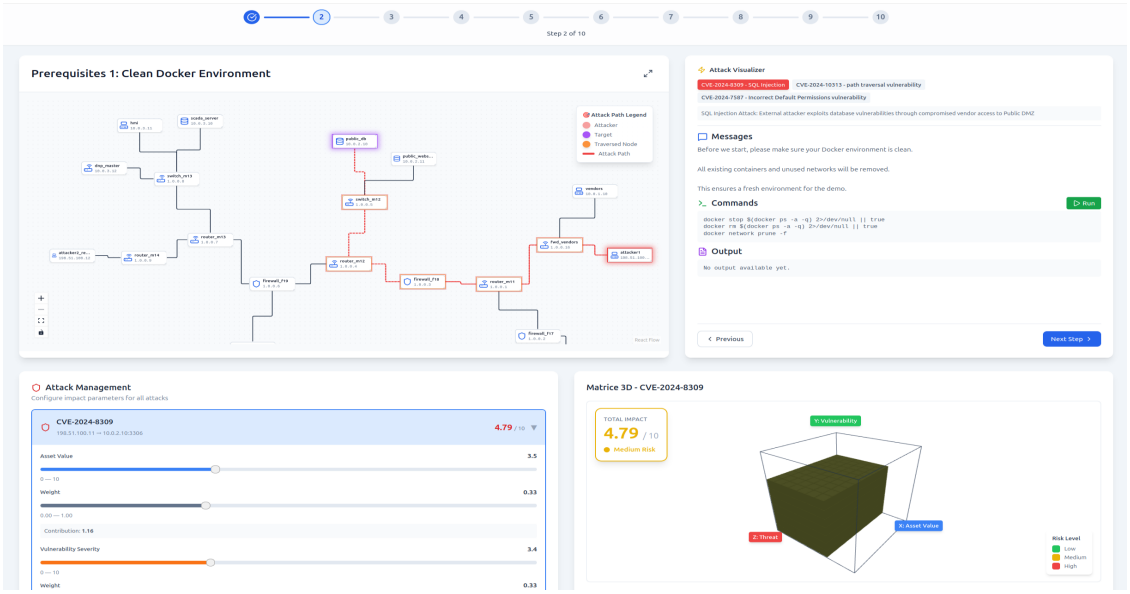


Figure 2.1: Comprehensive Graphical User Interface of the demonstrator

Furthermore, the newer GUI includes different new components such as the attack management and the 3D impact matrix visualization. Based on the attack simulation that is performed in this thesis there are 3 different attack scenarios and the attack management component allows to easily select and manage them. It is possible to select the attack scenario, modify the six components of the attack impact evaluation and visualize the impact of the attack in a 3D matrix. The 3D impact matrix visualization is designed to provide an intuitive representation of the attack impact based on the three metrics (Asset Value, Vulnerability Severity, and Threat Strength) and their weighted average. This allows users to quickly grasp the severity of the attack and make informed decisions about mitigation strategies. The interactive nature of the GUI, combined with the comprehensive attack management and impact visualization features, aims to enhance user engagement and facilitate a deeper understanding of the security implications of different attack scenarios.

2.2 Server application

The second objective of the thesis is to develop a server application that integrates with the client GUI and manages the overall workflow of the demonstrator. Also this work starts with a previous prototype, but the server application has been redesigned and implemented to be more modular and reusable.

Most of the API endpoints have been redesigned to be more general and reusable in different contexts and to support different topologies and attack scenarios without requiring substantial changes to the codebase. The server application is responsible for handling requests from the client GUI, orchestrating interactions with external tools such as VEREFOO and the scheduling backend, and managing the deployment of configurations in the Docker environment. The server application is implemented in Python and provides a RESTful API that allows the client GUI to

communicate with the backend services. It also manages the state of the demonstrator, including the current network topology, attack scenarios, and mitigation steps.

2.3 External tools

The third objective of the thesis is to integrate external tools such as VEREFOO, the scheduling backend, and Docker into a cohesive workflow that enables automated security configuration and dynamic mitigation in response to simulated attack scenarios.

The tools are already ready to use and functional, but the work of this thesis is to integrate them in a complete workflow and to test them in different attack scenarios and mitigation strategies. The integration of these tools allows the demonstrator to automate the process of configuring distributed firewalls, evaluating attack impacts, and scheduling mitigation steps based on the severity of the attack. By combining formal verification through VEREFOO with dynamic scheduling and containerized deployment using Docker, the demonstrator provides a comprehensive platform for exploring automated security responses in virtualized network environments.

Chapter 3

Tools and Technology

In this chapter, an overview of the main tools and technologies used in this thesis will be provided.

3.1 Docker

Docker[14] is an open platform for developing, shipping, and running applications. Docker enables to separate applications from the infrastructure in order to deliver software quickly and reliably.

Docker is based on containers, which are lightweight, portable, and self-sufficient units that can run in any environment.

A container[15] 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.

Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging. Some of their properties are:

- **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

Docker security relies on three factors: [16]

- isolation of processes at the userspace level managed by the Docker daemon
- enforcement of this isolation by the kernel
- network operations security

Docker containers isolation rely exclusively on Linux kernel features, including namespaces, cgroups, hardening, and capabilities. Namespace isolation and capabilities drop are enabled by default, but cgroups limitations aren't. However, the default configuration is relatively strict, but can be lowered by options, triggered at container launch.

Host hardening through Linux kernel security modules enforces security-related limitation constraints imposed on containers, such as compromising a container and escaping to the host operating system. Therefore, while default hardening protects the host from containers, it doesn't protect containers from other containers. This security aspect can be addressed by writing specific profiles that depend individually on the containers.

Docker uses network resources for image distribution and remote control of the Docker daemon. To distribute images, Docker verifies images downloaded from a remote repository with a hash and the connection to the registry is made over TLS. Moreover, the Docker Content Trust architecture now lets developers sign their images before pushing them to a repository. Content Trust relies on the update framework (TUF) which was specifically designed to address package manager flaws. TUF can recover from a key compromise, mitigate replay attacks by embedding expiration timestamps in signed images, and so on.

3.2 Verefoo

VErified REFinement and Optimized Orchestration (VEREFOO) is a framework that relies on a formal approach which combines automation, formal verification and optimization for the configuration of distributed firewalls.[13][17][18]

The high-level architecture of VEREFOO is organized in three main phases:

- Network Description and Intent Specification
- Firewall Allocation and Configuration
- Low-Level Rules Generation

Network Description and Intent Specification

In the first phase, the user describes the network topology and specifies the security intents. The network topology is described in an XML format, which includes information about the nodes, links, and their properties. The security intents are also specified in an XML format, which includes information about the desired security properties and the potential attacks that should be mitigated. The network and intent representations are expressed with a user-friendly language, which represents a link between humans and the automated process.

Firewall Allocation and Configuration

When the user has created all the inputs, the GUI or the REST Client sends an HTTP Request to VEREFOO, including them in its body with XML format. This allows the framework to validate the inputs against an XML schema, checking their syntax and preventing the framework from failures due to bad requests. After this initial check, VEREFOO internally builds the constraints composing the constraint programming problem modeling the firewall configuration problem. Then it employs a state-of-the-art MaxSMT solver by Microsoft Research, called Z3, to solve the MaxSMT problem. If a correct solution for the built problem cannot be found (e.g., a user-specified intent cannot be enforced in the corresponding topology), VEREFOO raises an error to the user. Instead, it generates an XML file describing the firewall allocation scheme and configuration, and sends it back in an HTTP Response. The GUI parses it and graphically shows the outputs, enabling the user to modify them for a next run of the framework.

Low-Level Rules Generation

In case of positive outcome, VEREFOO also performs the conversion of the XML representation of the firewall configuration to the syntax of a specific firewall implementation selected by the user (e.g., Fig. 5 shows this translation for an iptables configuration). Then, the produced low-level configuration is pushed in firewall containers deployed in a virtual network orchestrated by Docker Compose. The user can thus test the correctness of the generated rules, by pinging endpoints to assess if the communications are blocked or allowed. All these operations are executed quickly, compatibly with the dynamism of network virtualization.

Input and Output of VEREFOO

After having described Verefoo from an high level perspective, it is important for this thesis to analyze more in depth the input and output expected by the framework. The input is composed by one XML file describing the network topology and the security intents. In this demo we will have different node in the network topology and they are described in the XML with different tags.

Listing 3.1: Verefoo input example

```
<NFV xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
    noNamespaceSchemaLocation="../xsd/nfvSchema.xsd">
  <graphs>
    <graph id="0">
      <!-- Client  -->
      <node functional_type="WEBCLIENT" name="198.51.100.11">
        <neighbour name="1.0.0.16" />
        <configuration description="attacker1" name="attacker1">
          <webclient nameWebServer="10.0.2.11" />
        </configuration>
      </node>
      <!-- Server -->
      <node functional_type="WEBSERVER" name="10.0.2.10">
```

```
        <neighbour name="1.0.0.5" />
        <configuration description="E3" name="public_db">
          <webserver>
            <name>PublicDB</name>
          </webserver>
        </configuration>
      </node>
      <!-- Forwarder -->
      <node functional_type="FORWARDER" name="1.0.0.16">
        <neighbour name="198.51.100.11" />
        <neighbour name="10.0.1.10" />
        <neighbour name="1.0.0.1" />
        <configuration description="fwd-vendors" name="fwd_vendors">
          <forwarder>
            <name>ForwarderVendors</name>
          </forwarder>
        </configuration>
      </node>
      <!-- Firewall F21 -->
      <node functional_type="FIREWALL" name="1.0.0.14">
        <neighbour name="1.0.0.12" />
        <neighbour name="10.0.5.10" />
        <configuration description="F21" name="firewall_f21">
          <firewall defaultAction="DENY" index="1">
            <elements>
              <action>ALLOW</action>
              <source>10.0.5.10</source>
              <destination>10.0.4.10</destination>
              <protocol>OTHER</protocol>
              <src_port>*</src_port>
              <dst_port>*</dst_port>
            </elements>
            <elements>
              <action>ALLOW</action>
              <source>10.0.5.10</source>
              <destination>10.0.4.11</destination>
              <protocol>OTHER</protocol>
              <src_port>*</src_port>
              <dst_port>*</dst_port>
            </elements>
          </firewall>
        </configuration>
      </node>
    </graph>
  </graphs>
  <PropertyDefinition>
    <Property graph="0" name="IsolationProperty" src="10.0.3.12" dst="
    10.0.4.10" lv4proto="TCP"/>
    <Property graph="0" name="ReachabilityProperty" src="10.0.4.10" dst="
    10.0.3.12" lv4proto="TCP"/>
  </PropertyDefinition>

</NFV>
```

As we can observe in the listing 3.1 we will have 4 different nodes:

- WebClient: a node representing a web client, which is a device that accesses

web services.

- Forwarder: a node representing a forwarder, which is a device that forwards network traffic between different segments of a network.
- Webserver: a node representing a web server, which is a device that hosts web services and responds to requests from web clients.
- Firewall: a node representing a firewall, which is a device that monitors and controls incoming and outgoing network traffic based on predetermined security rules.

Additionally the file contains the property graphs that represent the security intents using the `Property` tag. In this example there are two properties: an isolation property and a reachability property that could be used to represent the intent of isolating a node from another one or the intent of allowing the communication between two nodes respectively. Verefoo will takes this type of files as input and will produce an output file if the configuration is correct.

Listing 3.2: Verefoo output example

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<NFV>
    <graphs>
        <graph id="0">

            <node id="1" name="198.51.100.11" functional_type="WEBCLIENT">
                <neighbour id="100" name="1.0.0.16"/>
                <configuration name="attacker1" description="attacker1">
                    <webclient nameWebServer="10.0.2.11"/>
                </configuration>
            </node>

      <node id="2" name="10.0.2.10" functional_type="WEBSERVER">
        <neighbour name="1.0.0.5" />
        <configuration description="E3" name="public_db">
          <webserver>
            <name>PublicDB</name>
          </webserver>
        </configuration>
      </node>

            <node id="3" name="1.0.0.16" functional_type="FORWARDER">
                <neighbour id="101" name="198.51.100.11"/>
                <neighbour id="102" name="10.0.1.10"/>
                <neighbour id="103" name="1.0.0.1"/>
                <configuration name="fwd_vendors" description="fwd-vendors">
                    <forwarder>
                        <name>ForwarderVendors</name>
                    </forwarder>
                </configuration>
            </node>

            <node id="4" name="1.0.0.14" functional_type="FIREWALL">
                <neighbour id="149" name="1.0.0.12"/>
                <neighbour id="150" name="10.0.5.10"/>
```

```
                <configuration name="firewall_f21" description="1">
                    <firewall defaultAction="DENY">
                        <elements>
                            <action>ALLOW</action>
                            <source>10.0.5.10</source>
                            <destination>10.0.4.10</destination>
                            <protocol>OTHER</protocol>
                            <src_port>*</src_port>
                            <dst_port>*</dst_port>
                        </elements>
                        <elements>
                            <action>ALLOW</action>
                            <source>10.0.5.10</source>
                            <destination>10.0.4.11</destination>
                            <protocol>OTHER</protocol>
                            <src_port>*</src_port>
                            <dst_port>*</dst_port>
                        </elements>
                    </firewall>
                </configuration>
            </node>
        </graph>
    </graphs>

    <PropertyDefinition>
        <Property name="IsolationProperty" graph="0" src="10.0.1.10" dst="
    10.0.2.11" lv4proto="TCP" isSat="true"/>
        <Property name="ReachabilityProperty" graph="0" src="10.0.2.11" dst="
    10.0.1.10" lv4proto="TCP" isSat="true"/>
    </PropertyDefinition>
</NFV>
```

The output file is an XML file, as we can observe, the listing 3.2 shows the firewall configuration generated by VEREFOO when the input file is correct based on our configuration. The file contains the same nodes of the input file but with additional information as the id of the node. Moreover, we can see that the `Property` tag has the additional information of the correctness of the property indicated by the `isSAt` attribute.

Firewall configuration

This specific firewall configuration is represented as Allowlist rules, but Verefoo permits all the standard firewall configurations. In the XML file we can modify the firewall rules by changing the `defaultAction` attribute of the `firewall` tag, which can be set to either `ALLOW` or `DENY` to describe a denylist or allowlist approach. Based on the default action, the internal rules will be generated accordingly. Returning to the example of the listing 3.2, we can see that the default action is set to `DENY` and there are two rules that allow the communication between nodes[19] The rules are specified as follows:

- `action`: the action to be taken when the rule is matched, which can be either ALLOW or DENY.

- `source`: the source IP address or network that the rule applies to.
- `destination`: the destination IP address or network that the rule applies to.
- `protocol`: the protocol that the rule applies to, such as TCP, UDP or OTHER.
- `src_port`: the source port that the rule applies to, which can be a specific port number or a range of ports.
- `dst_port`: the destination port that the rule applies to, which can be a specific port number or a range of ports.

Generator of Docker's file

VEREFOO also generates a Docker Compose file and all the specific files for every firewall that will be used to deploy the virtual network. The Docker Compose file is a YAML file that describes the services and networks for a Docker application. The file generated by VEREFOO contains the configuration for all the firewall containers that need to be deployed based on the input file and the generated firewall configuration. This file can be used to deploy the virtual network with all the firewall rules in place, allowing to test the correctness of the configuration by pinging endpoints and checking if the communication is allowed or blocked based on the firewall rules.

Example of the generated firewall configuration file is the following:

Listing 3.3: Firewall file .sh

```
#!/bin/sh
cmd="sudo␣iptables"
${cmd} -F
${cmd} -P INPUT DROP
${cmd} -P FORWARD DROP
${cmd} -P OUTPUT DROP
${cmd} -A FORWARD -p icmp -s 10.0.5.10/32 -d 10.0.4.10/32 -j
   ACCEPT
${cmd} -A FORWARD -p icmp -s 10.0.5.10/32 -d 10.0.4.11/32 -j
   ACCEPT
```

After having generated all the necessary files, the virtual network can be deployed using the docker compose and the file `startScript.sh` which is also generated by VEREFOO and contains all the necessary commands to start the virtual network and to execute the firewall configuration files in the correct order.

3.3 ReactFlow

React Flow[20] is a customizable React component for building node-based editors and interactive diagrams. It provides a powerful and flexible way to create complex

and dynamic user interfaces for applications such as flowcharts, mind maps, and data visualizations. Moreover, developers can easily create and manage nodes, edges, and other elements of a graph-based interface, allowing users to interact with and manipulate the data in real-time. Most of the important features needed for this research comes out of the box.

At its core, React Flow is a collection of nodes connected by edges. Nodes represent individual components or entities, while edges represent the relationships or connections between them. React Flow provides a simple and intuitive API for creating and managing these nodes and edges, allowing developers to easily build complex diagrams and visualizations.

A handle is the attachment point where an edge connects to a node, and it can be either a source handle (where an edge starts) or a target handle (where an edge ends). Handles are used to define the flow of data or control between nodes in a diagram. They can be customized with different styles and properties, such as position, size, and color, to enhance the visual representation of the diagram

A powerful feature of React Flow is the customization options it provides. Developers can create custom node types, edge types, and handle types to suit their specific needs. This allows for a high degree of flexibility in designing the user interface and tailoring it to the requirements of the application.

Other important features include the layouting possibilities that offers, React Flow has not implemented their own layouting solution yet, but they offer integration with third-party libraries such as Dagre and ELK. These libraries provide powerful algorithms for automatically arranging nodes and edges in a visually appealing way, making it easier to create complex diagrams without having to manually position each element. For this research, Elkjs is the library chosen for the automatic layouting of the graph as it is the most powerful and flexible solution available.

In this research the NFV XML of the SCADA Topology is converted into a graph structures based on nodes and edges in order to be used with React Flow. Every node has 8 different handles (4 source and 4 target) based on the position of the handle on the node (top, right, bottom, left). This allows to have a more clear and intuitive representation of the graph and to easily understand the flow of data between the nodes. Moreover, every nodes has different information:

- id: a unique identifier for the node
- position: the position of the node in the graph defined by x and y coordinates (all starts position are 0,0 as the layouting algorithm will take care of the final position of the nodes)
- type: the type of the node, which can be used to define different styles and properties for different types of nodes
- data: an object that can contain any additional data related to the node such as label, ip, logical name.

The type of the nodes ensures different render styles, in the topology used on the research there are 5 different types of nodes:

1. WebClient
2. Forwarder
3. Webserver
4. Firewall
5. default (in case of nodes that doesn't belong to any of the previous types)

The edges, instead, have only 3 different information:

- id: a unique identifier for the edge
- source: the id of the source node
- target: the id of the target node

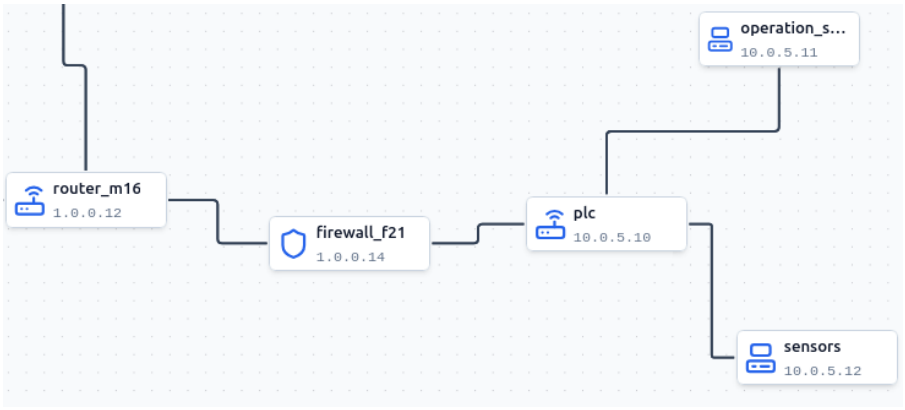


Figure 3.1: Graph elements in React Flow.

In Figure 3.1 there is an example of the graph elements in react flow, different types of nodes have different name and icon, inside of the node there is also the ip address of the node, it is important for the demo in order to ping the desired node and check if the communication is allowed or blocked based on the firewall rules.

React Flow allows to highlight nodes and edges based on different conditions as shown in Figure 3.2. This feature is used in the demo to highlight the nodes and edges that are involved in the attack simulation, in order to have a more clear representation of the attack and its impact on the network.

After having retrieved and transformed the data from the XML, the graph is given to elkjs[3.4] for the automatic layouting. After that a component called InteractiveNetworkGraph is used to render the graph and to make it interactive, allowing the user to click on the node and zoom in and out of the graph.

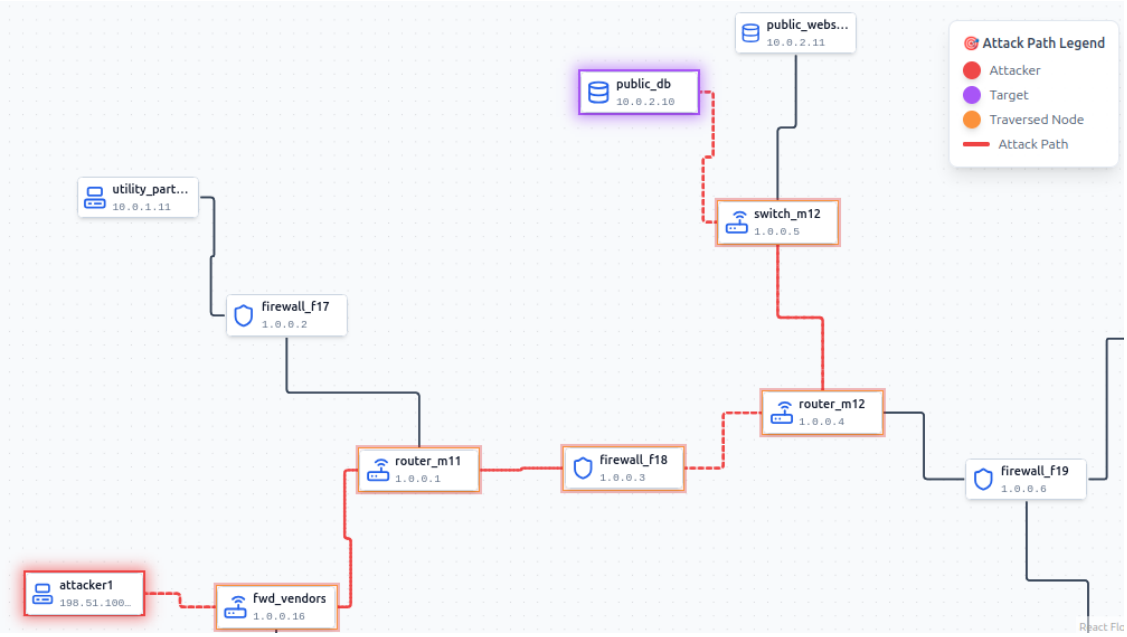


Figure 3.2: Highlighting nodes and edges in React Flow.

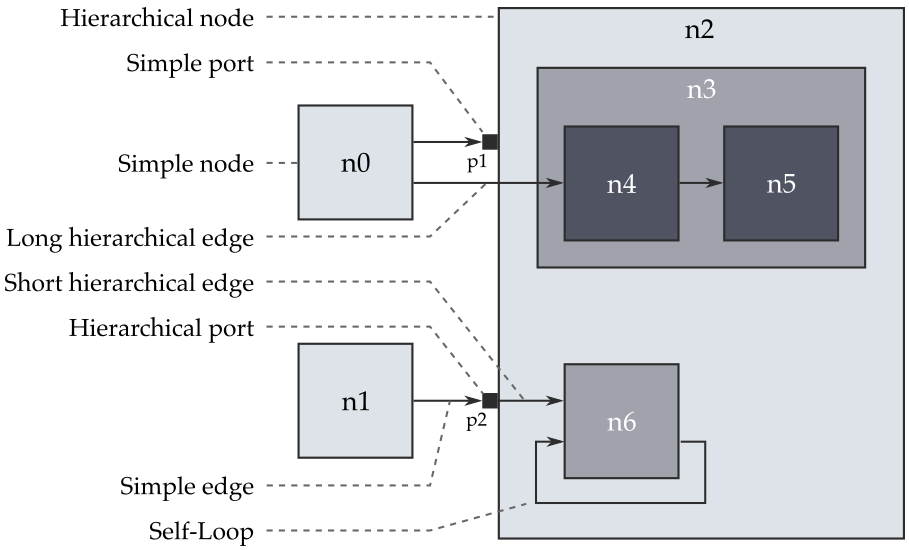


Figure 3.3: Graph elements in ELK.

3.4 Elkjs

The Eclipse Layout Kernel[21] (or ELK) is two things: a collection of layout algorithms and an infrastructure that bridges the gap between layout algorithms and diagram viewers and editors. ELK itself doesn't render the drawing but only computes positions (and possibly dimensions) for the diagram elements.

Elkjs[22] takes the layout-relevant part of ELK and makes it available to the JavaScript world. It is based on the Java implementation of ELK and his code is generated using GWT (Google Web Toolkit).

To represent graphs that should be laid out, the Eclipse Layout Kernel provides a robust EMF-based data structure with some definitions for graph elements:

- Inclusion Tree: The parent-child relationships of all nodes. This is basically the node containments in the EMF data structure. Note that in an ELK graph, the tree always has a single root node that represents the drawing area. Top-level nodes are children of that root node.
- Graph: A set of nodes and edges and whatever else belongs to them (labels, ports, ...).
  - Simple Graph: All children of a single node. The node represents that simple graph. This means that if the objects that the graph consists of are to be passed to a layout algorithm, this is done by simply passing the node object that represents that graph.
  - Hierarchical Graph: All descendants of a single node. Although the word all is a bit too strong here: we may choose to exclude all descendants of one of the node's descendants. Similar to simple graphs, that single node represents the hierarchical graph. The root node represents the whole graph (see below) and is simply a special case of this rule.
- Node:
  - Simple Node: A node that does not contain child nodes.
  - Hierarchical Node: A node that contains child nodes.
- Edge:
  - Simple Edge: An edge that connects two nodes in the same simple graph. This of course implies that the edge's source and target nodes both have the same parent node.
  - Short Hierarchical Edge: A hierarchical edge that only has to leave (or enter) a single hierarchical node to get to its target. Thus, a short hierarchical edge connects nodes in adjacent layers of hierarchy.
  - Long Hierarchical Edge: A hierarchical edge that is not a short hierarchical edge.
- Port: An explicit connection point on a node for edges to connect to.
  - Simple Port: A port on a simple node, or a port that does not have incident hierarchical edges.
  - Hierarchical Port: A port on a hierarchical node that has incident hierarchical edges.
- Root Node: The root of the inclusion tree.

A small example of the use of elkjs is shown in the following code snippet:

Listing 3.4: ELK basic usage example

```
const ELK = require('elkjs')
const elk = new ELK()
const graph = {
```

```
    id: "root",
    layoutOptions: { 'elk.algorithm': 'layered' },
    children: [
        { id: "n1", width: 30, height: 30 },
        { id: "n2", width: 30, height: 30 },
        { id: "n3", width: 30, height: 30 }
    ],
    edges: [
        { id: "e1", sources: [ "n1" ], targets: [ "n2" ] },
        { id: "e2", sources: [ "n1" ], targets: [ "n3" ] }
    ]
}
elk.layout(graph)
    .then(console.log)
    .catch(console.error)
```

The layout options are used to configure the layout algorithm. Attaching a layoutOptions object to the graph element that holds key/value pairs representing the desired layout options. In the documentations there are more than 20 different layout options and for each one of this options there are different option that can be set based on the preferences and needs of the user. For example, for the layered layout algorithm, there are options to set the direction of the layout (top-bottom, left-right, etc.), the spacing between nodes and edges, and the alignment of nodes within layers. By adjusting these options, it is possible to customize the appearance of the diagrams and optimize them for readability and clarity.

Most of the layout options are useful to be used on hierarchical graphs similar to trees, but for a network topology it is unlikely to have a straight hierarchy top to bottom or left to right. For this reason in this demo, and also for others used as testing, the layout algorithm used is the *org.eclipse.elk.stress* that minimizes the stress within a layout using stress majorization. Stress exists if the euclidean distance between a pair of nodes doesn't match their graph theoretic distance, that is, the shortest path between the two nodes. The method allows to specify individual edge lengths. Moreover it is from the category of force-directed layout algorithms that follow physical analogies by simulating a system of attractive and repulsive forces.

Listing 3.5: ELK.js stress layout configuration

```
const elkGraph = {
    id: 'root',
    layoutOptions: {
        // Use elk's stress algorithm (force category)
        // Identifier: org.eclipse.elk.stress `elk.
        algorithm': `org.eclipse.elk.stress',
        `elk.direction': `RIGHT',
        // Stress layout specific parameters
        // smaller epsilon => more precise layout (more iterations)
        `org.eclipse.elk.stress.epsilon': `0.000001',
        // desired distance between connected nodes (edge length)
        `org.eclipse.elk.stress.desiredEdgeLength': `250', },
```

```
        children: nodes.map((node) => ({
            id: node.id,
            width: NODE_WIDTH,
            height: NODE_HEIGHT, })),
        edges: edges.map((edge) =>({
            id: edge.id,
            sources: [edge.source],
            targets: [edge.target], })),
            }
```

Chapter 4

Demonstrator

The following section will describe the demonstrator from an high level overview, describing the architecture and the workflow of the system and then it will go in depth in the implementation of the demonstrator, describing the different components and how they interact with each other to provide a comprehensive view of the attack scenarios and the mitigation steps. Finally, it will describe the attack simulation and the reaction to the attack.

4.1 Architecture

The architecture of the demo is described in the Figure 4.1 and it is composed by 5 different components. The first component is the Client application, which is responsible for providing the graphical user interface (GUI), the network topology visualization, the attack simulation and a command execution interface used to execute the commands and to interact with the backend.

The second component is the Server application, which is responsible for providing the API used by the client. The server is written in Python with flask and it is responsible for receiving the requests from the client, processing them and sending the responses back to the client. The API provided by the server are different but we can focus on the most important ones:

- `GET /api/steps`: this API is used to fetch all the tutorial steps including the description of the step and the commands to be executed.
- `GET /api/get-graphs?step=<number>`: Retrieve network topology XML for a specific tutorial step since it can change during the demo.
- `POST /api/run-command`: Execute arbitrary shell commands on the server.
- `POST /api/open-terminal`: Open an interactive terminal to a Docker container.
- `POST /api/mito/start-attack`: Start the attack simulation by sending the attack parameters from the client to the scheduling backend.

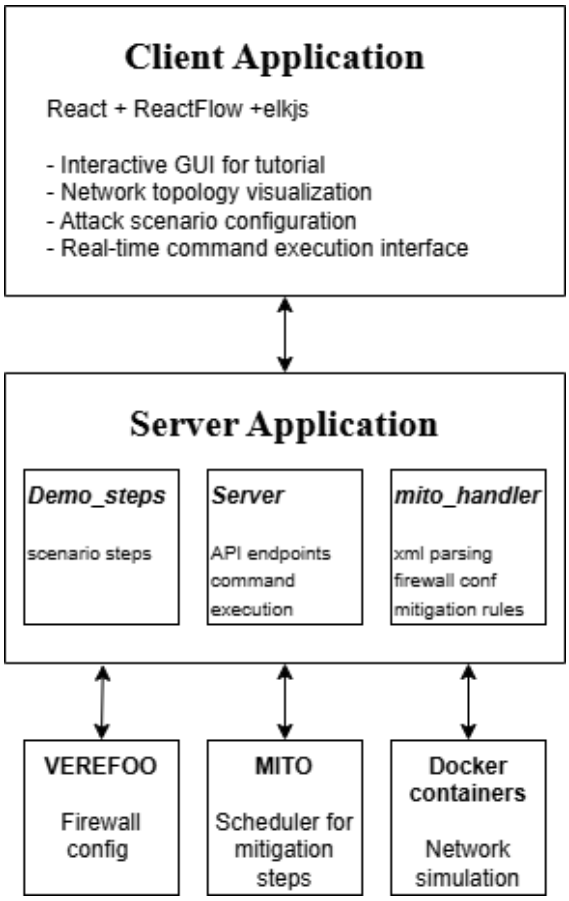


Figure 4.1: Demonstrator workflow architecture

- `POST /api/mito/execute-step/<step_number>`: Execute a specific mitigation step by sending the step number to the scheduling backend.

This API are used by the client to interact with the backend and to execute the commands needed for the demo, allowing the user to focus on the attack simulation and the mitigation steps without worrying about the underlying implementation details. Moreover, the server manage this API using different python scripts that are responsible for processing the requests and executing the commands.

The third component is VEREFOO, which is responsible for checking the validity of the network configuration and for generating the output configuration file. VEREFOO is implemented in Java and it uses the Z3 solver to check the validity of the configuration and to generate the output configuration file.

The forth component is the scheduling backend, which is responsible for receiving the attack impact evaluation from the client and for computing the mitigation steps based on the attack impact and the current network configuration

The fifth and last component is the Docker environment, which is responsible for deploying the virtual network topology and for simulating the attack scenarios. The Docker environment is managed through docker-compose and it is responsible for starting all the containers and for configuring them based on the configuration files created by the backend. Additionally, the Docker environment is also responsible

for simulating the attack scenarios by executing the commands needed to simulate the attacks and to apply the mitigation steps.

4.2 System design and workflow

This section provides an high level overview of the demonstrator, describing the system design and workflow.

Figure 4.2: Demo workflow 1

The first step of the workflow, figure 4.2 is the creation of the scenario, which is described by the `Scada_Topology.xml` file that will be analyzed later in section 4.3.2. This file is then sent to VEREFOO that will check the configuration to ensure the correctness of the network topology and if the scenario is valid. If the configuration is correct it will output a file called `Output_Configuration.xml`.

Figure 4.3: Demo workflow 2

The second step is described in the the figure 4.3 where the `Output_Configuration.xml` file is sent again to the backend, which will analyze the configuration and extract all the information needed to create the docker's file to prepare the virtual environment. Moreover, the backend will create the docker-compose file and all the necessary configuration files to deploy the virtual network. Thereafter it is possible to deploy the docker network and start the simulation of the attack scenarios.

The third step is described in the figure 4.4 where the attack scenarios are simulated and the reaction takes place. The attack scenarios are simulated with the help of the GUI in which the user can manipulate the attack parameters for all the 3 attack scenarios. The attack impact is calculated based on the metrics described in the section 4.3.2 and then sent to the backend. The Backend will analyze the attack impact of the 3 attacks and will schedule the mitigation steps taking care of applying the most effective mitigation step first. Since the attack scenarios are 3, the backend will indicate 3 steps to mitigate the attack, indicating the ip of the firewall to be updated or to be added. The demo will apply the mitigation steps one by one permitting the user to check the effectiveness of the single mitigation step by simulating the attack scenarios again after each one.

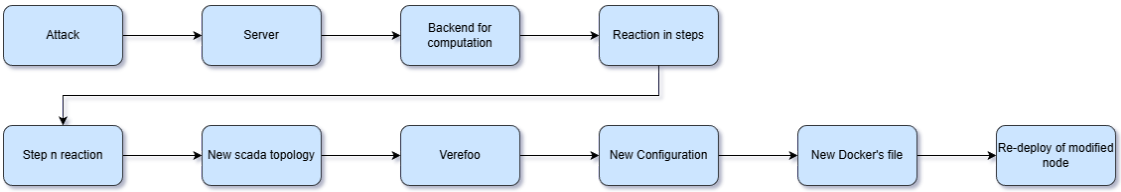


Figure 4.4: Demo workflow 3

4.3 Demonstrator implementation

4.3.1 Setup

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 22.04
- Docker Engine (desktop-linux) & Docker-compose
- Python
- openjdk
- z3 solver

4.3.2 SCADA Topology

For this demo we have chosen a SCADA topology a SCADA system inspired by the real topology model of the Texas 2000 infrastructure[23] and modified parts like the Vendor's zone and the Utility Partner's zone as depicted in the figure 4.5.[24]

There, two possible attack points were considered. The first concerns an attack vector external to the SCADA environment, where the access credentials or communication channels of users linked to the Vendors' Zone are compromised and malicious actors gain entry from outside the perimeter. This scenario reflects the risks associated with third-party access and supply chain connections, which often represent a weak point in industrial networks. The second scenario, on the other hand, focuses on an attack originating from inside the system, hypothesizing a successful infiltration into the remote communications with the HMI. This could occur through compromised maintenance accounts, misconfigured VPNs, or infected devices used by authorized operators. By leveraging these two entry points, the attacker is able to disrupt normal operations either by penetrating the network from the outside or by exploiting trusted internal channels, demonstrating how both external and internal vectors can threaten the integrity of a SCADA system. The demo aims to simulate 3 different attacks scenarios.

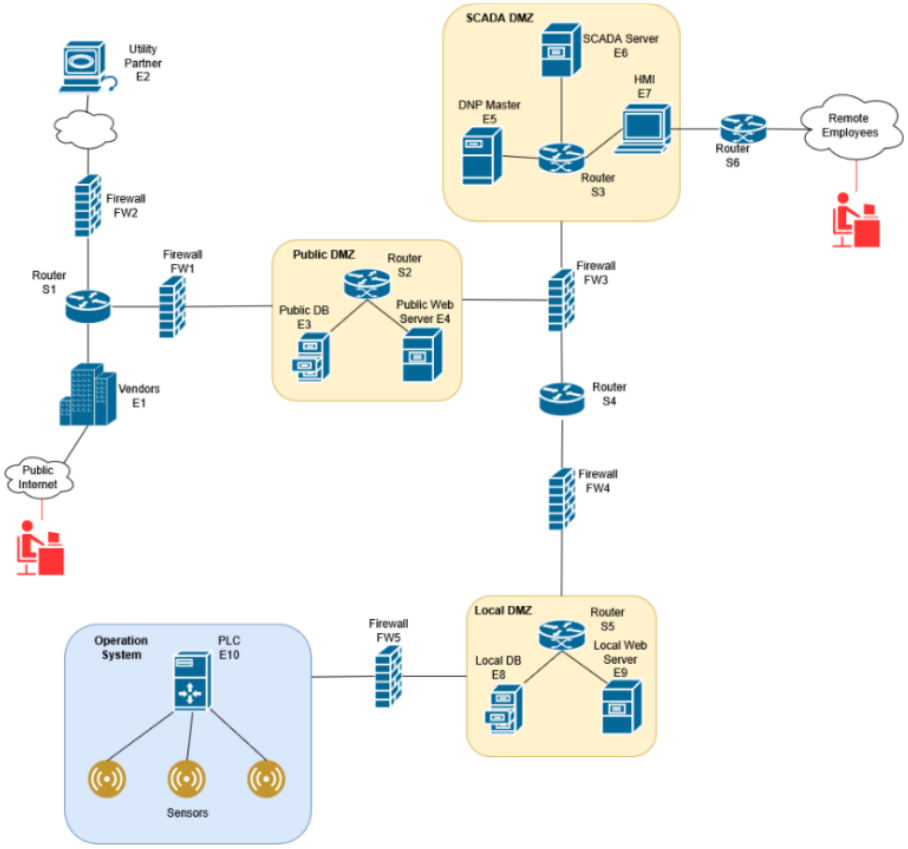


Figure 4.5: SCADA topology

The first attack scenario is a Denial of Service (DoS) attack, where the attacker, on the left in figure 4.5 floods integration of the SCADA system exploiting a vulnerability corresponding to the CVE-2024-7587[25] present on the utilities Partner's devices.

The second attack scenario is a data exfiltrate attack coming from the same attacker, design flaws to exfiltrate data relating to the system in the Public Database situated in the public DMZ. The vulnerability exploited in this case is the CVE-20248309[26] which allows prompt injection on specific port is not correctly filtered or closed.

The third considered attack is a combination of infiltration and data injection, coming from a different attack source, located inside the SCADA environment, on the right in figure 4.5. The exploitable vulnerability is CVE-2024-10313[27].

Based on the figure 4.5 there are some approximations like the 3 clouds that represent the internet. The topology used for the demo translates those clouds in forwarders nodes that have the task to forward the traffic to the correct destination. Also for the zones denominated as "SCADA DMZ", "Public DMZ", "Local DMZ" and "Operation System" the connections from outside to inside are translated in the demo with direct connection between router and firewall.

The topology is described in the `Scada_Topology.xml` file that contains all the information about the network configuration. A snippet of the entire file is reported in listing 4.1. Just the firewall and some nodes are reported in the listing, but the

actual file contains all the information about the network configuration, including all the nodes and the connections between them.

Listing 4.1: SCADA topology XML configuration

```
<NFV xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
    noNamespaceSchemaLocation="../xsd/nfvSchema.xsd">
  <graphs>
    <graph id="0">
      <node functional_type="WEBCLIENT" name="198.51.100.11">
        <neighbour name="1.0.0.16" />
        <configuration description="attacker1" name="attacker1">
          <webclient nameWebServer="10.0.2.11" />
        </configuration>
      </node>
      <node functional_type="FORWARDER" name="1.0.0.16">
        <neighbour name="198.51.100.11" />
        <neighbour name="10.0.1.10" />
        <neighbour name="1.0.0.1" />
        <configuration description="fwd-vendors" name="fwd_vendors">
          <forwarder>
            <name>ForwarderVendors</name>
          </forwarder>
        </configuration>
      </node>
      <node functional_type="WEBCLIENT" name="198.51.100.12">
        <neighbour name="1.0.0.9" />
        <configuration description="attacker2" name="
    attacker2_remote_employees">
          <webclient nameWebServer="10.0.3.11" />
        </configuration>
      </node>
      <node functional_type="WEBCLIENT" name="10.0.1.10">
        <neighbour name="1.0.0.16" />
        <configuration description="E1" name="vendors">
          <webclient nameWebServer="10.0.2.11" />
        </configuration>
      </node>
      <node functional_type="WEBCLIENT" name="10.0.1.11">
        <neighbour name="1.0.0.2" />
        <configuration description="E2" name="utility_partner">
          <webclient nameWebServer="10.0.3.10" />
        </configuration>
      </node>
      <node functional_type="FORWARDER" name="1.0.0.1">
        <neighbour name="1.0.0.2" />
        <neighbour name="1.0.0.3" />
        <neighbour name="1.0.0.16" />
        <configuration description="M11" name="router_m11">
          <forwarder>
            <name>RouterM11</name>
          </forwarder>
        </configuration>
      </node>
      <node functional_type="FIREWALL" name="1.0.0.2">
        <neighbour name="10.0.1.11" />
        <neighbour name="1.0.0.1" />
        <configuration description="F17" name="firewall_f17">
```

```
        <firewall defaultAction="ALLOW" index="1">

        </firewall>
      </configuration>
    </node>
    <node functional_type="FIREWALL" name="1.0.0.3">
      <neighbour name="1.0.0.1" />
      <neighbour name="1.0.0.4" />
      <configuration description="F18" name="firewall_f18">
        <firewall defaultAction="ALLOW" index="2">

        </firewall>
      </configuration>
    </node>
    <node functional_type="FIREWALL" name="1.0.0.6">
      <neighbour name="1.0.0.4" />
      <neighbour name="1.0.0.7" />
      <neighbour name="1.0.0.10" />
      <configuration description="F19" name="firewall_f19">
        <firewall defaultAction="ALLOW" index="3">
        </firewall>
      </configuration>
    </node>
    <node functional_type="FIREWALL" name="1.0.0.11">
      <neighbour name="1.0.0.10" />
      <neighbour name="1.0.0.12" />
      <configuration description="F20" name="firewall_f20">
        <firewall defaultAction="ALLOW" index="4">
        </firewall>
      </configuration>
    </node>
    <node functional_type="FIREWALL" name="1.0.0.14">
      <neighbour name="1.0.0.12" />
      <neighbour name="10.0.5.10" />
      <configuration description="F21" name="firewall_f21">
        <firewall defaultAction="DENY" index="5">
          <elements>
            <action>ALLOW</action>
            <source>10.0.5.10</source>
            <destination>10.0.4.10</destination>
            <protocol>OTHER</protocol>
            <src_port>*</src_port>
            <dst_port>*</dst_port>
          </elements>
          <elements>
            <action>ALLOW</action>
            <source>10.0.5.10</source>
            <destination>10.0.4.11</destination>
            <protocol>OTHER</protocol>
            <src_port>*</src_port>
            <dst_port>*</dst_port>
          </elements>
          <elements>
            <action>ALLOW</action>
            <source>10.0.5.11</source>
            <destination>10.0.4.10</destination>
            <protocol>OTHER</protocol>
```

```
    <src_port>*</src_port>
    <dst_port>*</dst_port>
  </elements>
  <elements>
    <action>ALLOW</action>
    <source>10.0.5.11</source>
    <destination>10.0.4.11</destination>
    <protocol>OTHER</protocol>
    <src_port>*</src_port>
    <dst_port>*</dst_port>
  </elements>
  <elements>
    <action>ALLOW</action>
    <source>10.0.5.12</source>
    <destination>10.0.4.10</destination>
    <protocol>OTHER</protocol>
    <src_port>*</src_port>
    <dst_port>*</dst_port>
  </elements>
  <elements>
    <action>ALLOW</action>
    <source>10.0.5.12</source>
    <destination>10.0.4.11</destination>
    <protocol>OTHER</protocol>
    <src_port>*</src_port>
    <dst_port>*</dst_port>
  </elements>
  <elements>
    <action>ALLOW</action>
    <source>10.0.4.10</source>
    <destination>10.0.5.10</destination>
    <protocol>OTHER</protocol>
    <src_port>*</src_port>
    <dst_port>*</dst_port>
  </elements>
  <elements>
    <action>ALLOW</action>
    <source>10.0.4.10</source>
    <destination>10.0.5.11</destination>
    <protocol>OTHER</protocol>
    <src_port>*</src_port>
    <dst_port>*</dst_port>
  </elements>
  <elements>
    <action>ALLOW</action>
    <source>10.0.4.10</source>
    <destination>10.0.5.12</destination>
    <protocol>OTHER</protocol>
    <src_port>*</src_port>
    <dst_port>*</dst_port>
  </elements>
  <elements>
    <action>ALLOW</action>
    <source>10.0.4.11</source>
    <destination>10.0.5.10</destination>
    <protocol>OTHER</protocol>
    <src_port>*</src_port>
```

```
            <dst_port>*</dst_port>
          </elements>
          <elements>
            <action>ALLOW</action>
            <source>10.0.4.11</source>
            <destination>10.0.5.11</destination>
            <protocol>OTHER</protocol>
            <src_port>*</src_port>
            <dst_port>*</dst_port>
          </elements>
          <elements>
            <action>ALLOW</action>
            <source>10.0.4.11</source>
            <destination>10.0.5.12</destination>
            <protocol>OTHER</protocol>
            <src_port>*</src_port>
            <dst_port>*</dst_port>
          </elements>
        </firewall>
      </configuration>
    </node>
  </graph>
 </graphs>
 <Constraints>
  <NodeConstraints/>
  <LinkConstraints />
 </Constraints>
 <PropertyDefinition>
 <Property graph="0" name="ReachabilityProperty" src="10.0.1.10" dst="
  10.0.2.11" lv4proto="TCP"/>
 <Property graph="0" name="ReachabilityProperty" src="10.0.2.11" dst="
  10.0.1.10" lv4proto="TCP"/>
 <Property graph="0" name="ReachabilityProperty" src="10.0.1.10" dst="
  10.0.1.11" lv4proto="TCP"/>
 </PropertyDefinition>
</NFV>
```

The `Scada_Topology.xml` file is provided as input to the VEREFOO which will check the configuration creating a MaxSMT problem that is solved by Z3 solver and produces the `Output_Configuration.xml` file.

A snippet of the entire file produced is in listing 4.2. As before, just the firewall and some nodes are reported in the listing, but the actual file contains all the information about the network configuration, including all the nodes and the connections between them.

Listing 4.2: Output topology XML configuration

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<NFV>
    <graphs>
        <graph id="0">
            <node id="1" name="198.51.100.11" functional_type="WEBCLIENT">
                <neighbour id="100" name="1.0.0.16"/>
                <configuration name="attacker1" description="attacker1">
                    <webclient nameWebServer="10.0.2.11"/>
                </configuration>
```

```
            </node>
            <node id="2" name="1.0.0.16" functional_type="FORWARDER">
                <neighbour id="101" name="198.51.100.11"/>
                <neighbour id="102" name="10.0.1.10"/>
                <neighbour id="103" name="1.0.0.1"/>
                <configuration name="fwd_vendors" description="fwd-vendors">
                    <forwarder>
                        <name>ForwarderVendors</name>
                    </forwarder>
                </configuration>
            </node>
            <node id="3" name="198.51.100.12" functional_type="WEBCLIENT">
                <neighbour id="104" name="1.0.0.9"/>
                <configuration name="attacker2_remote_employees" description="
    attacker2">
                    <webclient nameWebServer="10.0.3.11"/>
                </configuration>
            </node>
            <node id="4" name="10.0.1.10" functional_type="WEBCLIENT">
                <neighbour id="105" name="1.0.0.16"/>
                <configuration name="vendors" description="E1">
                    <webclient nameWebServer="10.0.2.11"/>
                </configuration>
            </node>
            <node id="5" name="10.0.1.11" functional_type="WEBCLIENT">
                <neighbour id="106" name="1.0.0.2"/>
                <configuration name="utility_partner" description="E2">
                    <webclient nameWebServer="10.0.3.10"/>
                </configuration>
            </node>
            <node id="6" name="1.0.0.1" functional_type="FORWARDER">
                <neighbour id="107" name="1.0.0.2"/>
                <neighbour id="108" name="1.0.0.3"/>
                <neighbour id="109" name="1.0.0.16"/>
                <configuration name="router_m11" description="M11">
                    <forwarder>
                        <name>RouterM11</name>
                    </forwarder>
                </configuration>
            </node>
            <node id="7" name="1.0.0.2" functional_type="FIREWALL">
                <neighbour id="110" name="10.0.1.11"/>
                <neighbour id="111" name="1.0.0.1"/>
                <configuration name="firewall_f17" description="1">
                    <firewall defaultAction="ALLOW"/>
                </configuration>
            </node>
            <node id="8" name="1.0.0.3" functional_type="FIREWALL">
                <neighbour id="112" name="1.0.0.1"/>
                <neighbour id="113" name="1.0.0.4"/>
                <configuration name="firewall_f18" description="2">
                    <firewall defaultAction="ALLOW"/>
                </configuration>
            </node>
            <node id="13" name="1.0.0.6" functional_type="FIREWALL">
                <neighbour id="122" name="1.0.0.4"/>
                <neighbour id="123" name="1.0.0.7"/>
```

```
        <neighbour id="124" name="1.0.0.10"/>
        <configuration name="firewall_f19" description="3">
            <firewall defaultAction="ALLOW"/>
        </configuration>
    </node>
    <node id="21" name="1.0.0.11" functional_type="FIREWALL">
        <neighbour id="139" name="1.0.0.10"/>
        <neighbour id="140" name="1.0.0.12"/>
        <configuration name="firewall_f20" description="4">
            <firewall defaultAction="ALLOW"/>
        </configuration>
    </node>
    <node id="26" name="1.0.0.14" functional_type="FIREWALL">
        <neighbour id="149" name="1.0.0.12"/>
        <neighbour id="150" name="10.0.5.10"/>
        <configuration name="firewall_f21" description="5">
            <firewall defaultAction="DENY">
                <elements>
                    <action>ALLOW</action>
                    <source>10.0.5.10</source>
                    <destination>10.0.4.10</destination>
                    <protocol>OTHER</protocol>
                    <src_port>*</src_port>
                    <dst_port>*</dst_port>
                </elements>
                <elements>
                    <action>ALLOW</action>
                    <source>10.0.5.10</source>
                    <destination>10.0.4.11</destination>
                    <protocol>OTHER</protocol>
                    <src_port>*</src_port>
                    <dst_port>*</dst_port>
                </elements>
                <elements>
                    <action>ALLOW</action>
                    <source>10.0.5.11</source>
                    <destination>10.0.4.10</destination>
                    <protocol>OTHER</protocol>
                    <src_port>*</src_port>
                    <dst_port>*</dst_port>
                </elements>
                <elements>
                    <action>ALLOW</action>
                    <source>10.0.5.11</source>
                    <destination>10.0.4.11</destination>
                    <protocol>OTHER</protocol>
                    <src_port>*</src_port>
                    <dst_port>*</dst_port>
                </elements>
                <elements>
                    <action>ALLOW</action>
                    <source>10.0.5.12</source>
                    <destination>10.0.4.10</destination>
                    <protocol>OTHER</protocol>
                    <src_port>*</src_port>
                    <dst_port>*</dst_port>
                </elements>
```

```
                <elements>
                    <action>ALLOW</action>
                    <source>10.0.5.12</source>
                    <destination>10.0.4.11</destination>
                    <protocol>OTHER</protocol>
                    <src_port>*</src_port>
                    <dst_port>*</dst_port>
                </elements>
                <elements>
                    <action>ALLOW</action>
                    <source>10.0.4.10</source>
                    <destination>10.0.5.10</destination>
                    <protocol>OTHER</protocol>
                    <src_port>*</src_port>
                    <dst_port>*</dst_port>
                </elements>
                <elements>
                    <action>ALLOW</action>
                    <source>10.0.4.10</source>
                    <destination>10.0.5.11</destination>
                    <protocol>OTHER</protocol>
                    <src_port>*</src_port>
                    <dst_port>*</dst_port>
                </elements>
                <elements>
                    <action>ALLOW</action>
                    <source>10.0.4.10</source>
                    <destination>10.0.5.12</destination>
                    <protocol>OTHER</protocol>
                    <src_port>*</src_port>
                    <dst_port>*</dst_port>
                </elements>
                <elements>
                    <action>ALLOW</action>
                    <source>10.0.4.11</source>
                    <destination>10.0.5.10</destination>
                    <protocol>OTHER</protocol>
                    <src_port>*</src_port>
                    <dst_port>*</dst_port>
                </elements>
                <elements>
                    <action>ALLOW</action>
                    <source>10.0.4.11</source>
                    <destination>10.0.5.11</destination>
                    <protocol>OTHER</protocol>
                    <src_port>*</src_port>
                    <dst_port>*</dst_port>
                </elements>
                <elements>
                    <action>ALLOW</action>
                    <source>10.0.4.11</source>
                    <destination>10.0.5.12</destination>
                    <protocol>OTHER</protocol>
                    <src_port>*</src_port>
                    <dst_port>*</dst_port>
                </elements>
            </firewall>
```

```
                </configuration>
            </node>
        </graph>
    </graphs>
    <Constraints>
        <NodeConstraints/>
        <LinkConstraints/>
    </Constraints>
    <PropertyDefinition>
        <Property name="ReachabilityProperty" graph="0" src="10.0.1.10" dst="
    10.0.2.11" lv4proto="TCP" isSat="true"/>
        <Property name="ReachabilityProperty" graph="0" src="10.0.2.11" 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"/>
    </PropertyDefinition>
</NFV>
```

4.3.3 Graphical User Interface

In order to simulate the attack scenarios and to visualize the virtual network topology, a graphical user interface (GUI) has been developed. The GUI is built using React and has 5 mayor components:

- **Steps**: this component is responsible for displaying the current steps and all the others steps in order to navigate the demo.
- **Network Topology**: this component is responsible for visualizing the network topology, showing all the nodes and the connections between them.
- **Messages and commands**: this component is responsible for displaying all the messages and commands related to the attack simulation and mitigation steps.
- **Attack management**: this component is responsible for displaying and modifying the attack parameters for all the 3 attack scenarios.
- **3D matrix**: this component is responsible for visualizing the 3D impact matrix for the attack scenarios, showing the impact of each attack based on the calculated metrics.

Those components will be described in more details in the following sections, showing how they are implemented and how they interact with each other to provide a comprehensive view of the attack scenarios and the mitigation steps.

Steps

As shown in the figure 4.6 the steps component is responsible for displaying the current step of the demo and all the other steps in order to navigate through the demo. Each step represents a specific phase of the demo that can be selected by

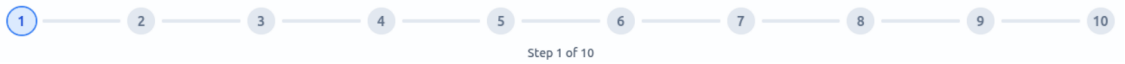


Figure 4.6: Steps component of the GUI

the user to visualize the corresponding information and actions. To navigate from one step to another, the user can click on the corresponding step in the list or can use the button "Next step" present in the component Messages and commands.

Network topology

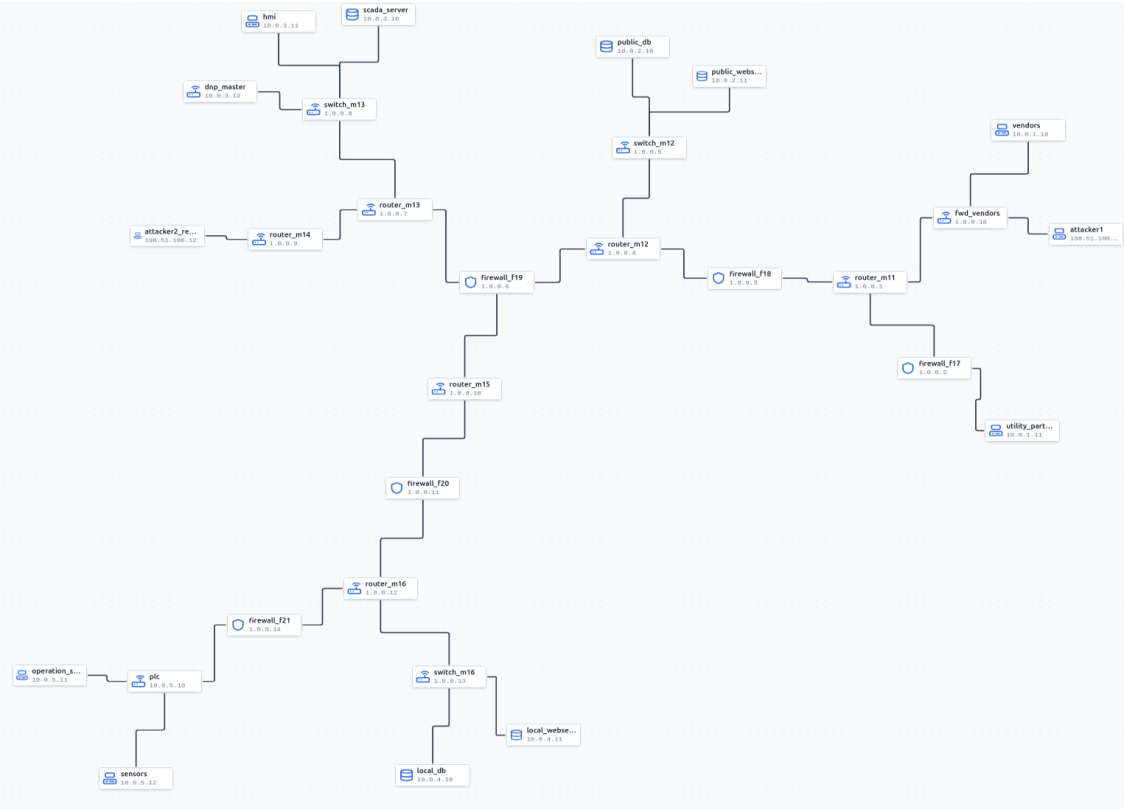


Figure 4.7: Network topology component of the GUI

The network topology component, as shown in the figure 4.7, is responsible for visualizing the network topology of the system. It shows all the nodes and the connections between them, allowing the user to understand the structure of the network and how the different components are connected. The topology is generated based on the information provided in the `SCADATopology.xml` file and use a the library reactFlow 3.3 to visualize the graph and then to allow an automatic layout of the nodes and the connections uses elkjs 3.4. As stated before, reactFlow consent to visualize the graph and have some interactive features like zooming and panning. The topology is updated based on the changes made to the network configuration during the demo, allowing the user to see how the topology evolves as the attack scenarios are simulated and mitigated. Elkjs modifies the layout of the graph based on the changes made to the network configuration, for example when

a new firewall is added the entire topology is recalculated based on the algorithm and parameters chosen.

In addiction, the network topology component also allows the user to visualize the attack paths and the affected nodes during the attack simulation, providing a clear view of how the attacks are propagating through the network and which nodes are affected. Moreover, there is the possibility to have a full screen view of the topology, allowing the user to have a better view of the entire network structure and the attack paths.

Messages and commands

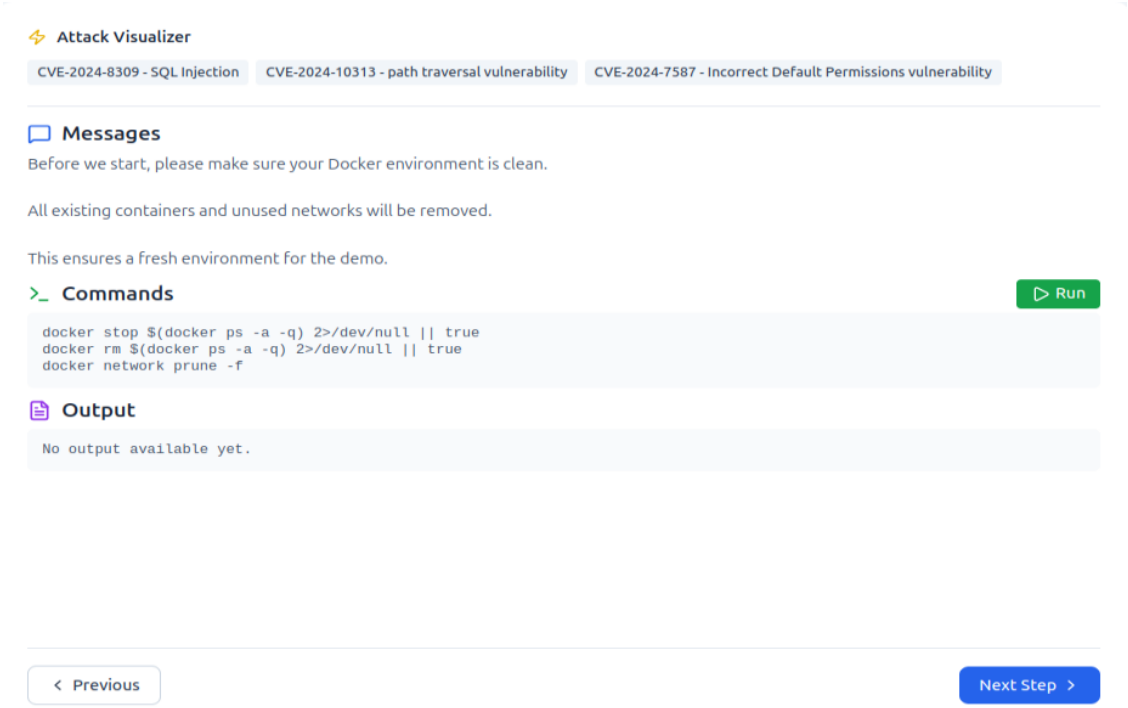


Figure 4.8: Messages and commands component of the GUI

The messages and commands component, as shown in the figure 4.8, is responsible for displaying all the messages and commands related to the attack simulation and mitigation steps. It provides the story of the demo, guiding the user through the different phases of the attack simulation and mitigation. On top of the components there is the attack visualizer that allows the user to visualize the 3 attack scenarios on the network topology components, showing the attack paths and the affected nodes. Below the attack visualizer there is a text area that displays all the messages related to the current step of the demo, providing information about what are the commands that will be executed and what are the expected results. After the messages there are the actual commands that will be executed during the demo. Moreover, there is the output of the commands that will be displayed after the execution, allowing the user to see the results of the commands and what are they doing to the system. Finally, there are the "Next step" and "Previous" that allows the user to navigate to the next step of the demo or the previous step, allowing to go back and forth between the different phases of the demo.

For some steps where there is not the need to execute commands, the component will display only the attacks visualizer and the messages, providing information about the current step and what is expected to happen in the next steps.

Attack management

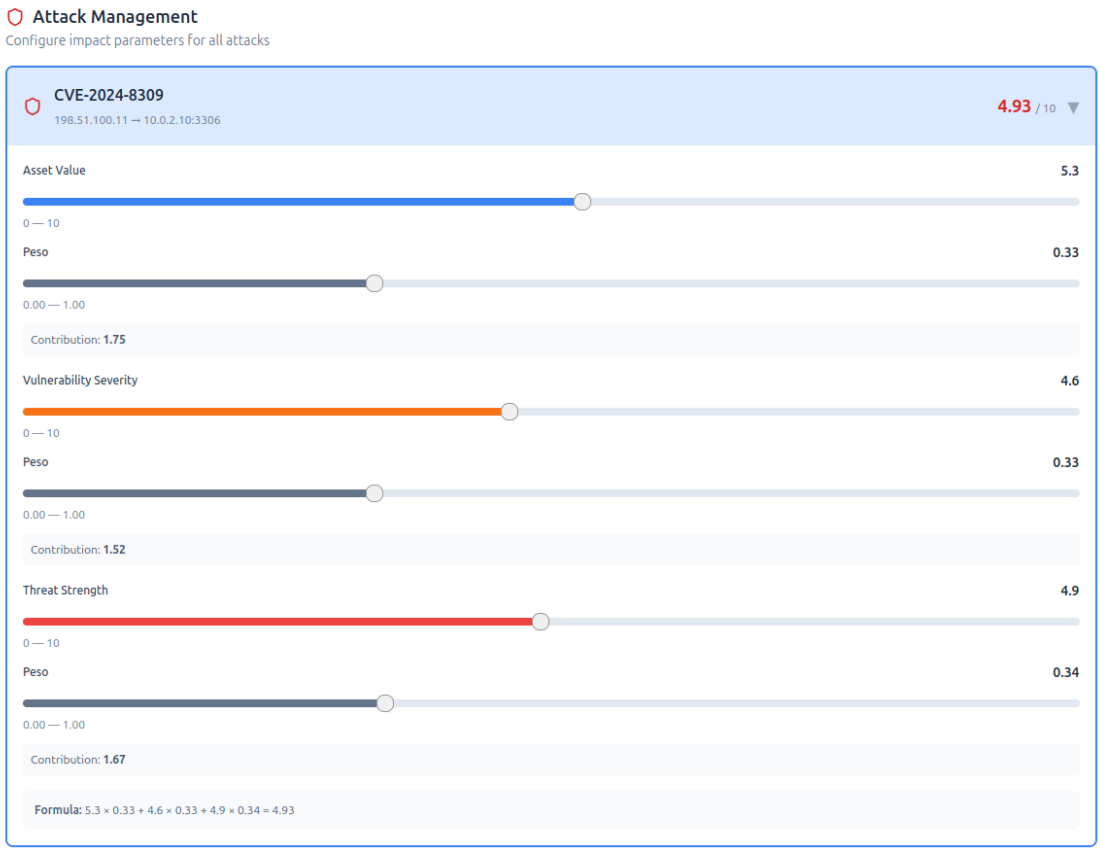


Figure 4.9: Metrics for attack impact evaluation

This 3 attacks are considered separately and for each of the them the impact is calculated based on 3 different metrics:

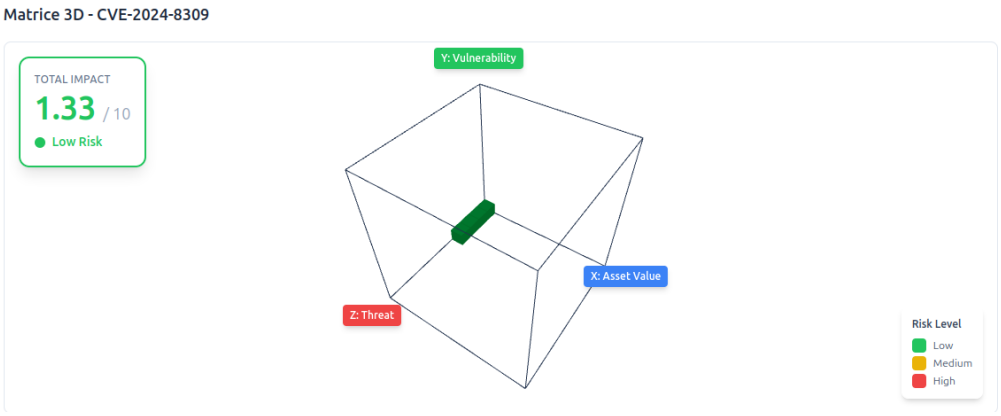
- Asset Value: which represents the worth of any data, device, or other component of the environment that can be illicitly accessed, used, disclosed, destroyed, and/or stolen, resulting in an economic and 6 reputational loss;
- Vulnerability Severity: which represents the severity of the vulnerability exploited by the attack, which can be evaluated based on established frameworks such as CVSS (Common Vulnerability Scoring System) or other relevant metrics that consider factors like exploitability, impact, and ease of exploitation;
- Threat Strength: meant as the sufferance of the node in terms of resource consumption.

This metrics are then combined with a weighted average to calculate the overall impact of the attack, which is used to prioritize mitigation efforts and allocate resources effectively.

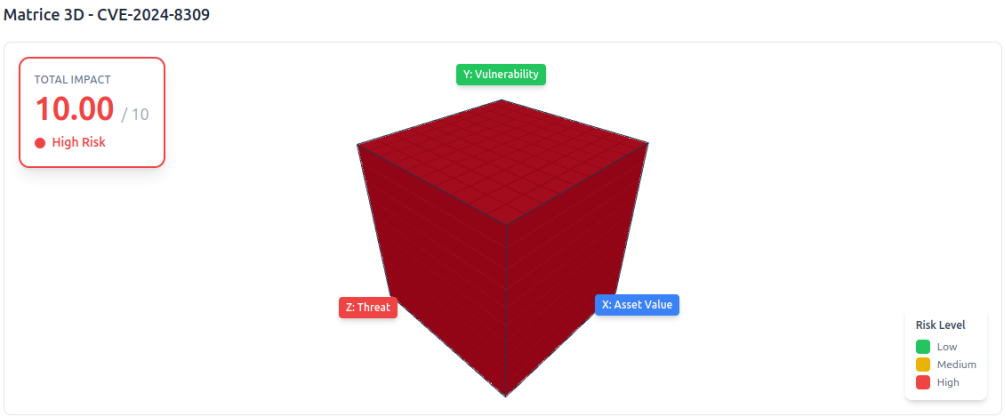
$$\text{Impact} = w_1 * \text{Asset Value} + w_2 * \text{Vulnerability Severity} + w_3 * \text{Threat Strength}$$

The impact of all the attacks will be sent to the backends in order to be evaluated and compute the steps needed to mitigate the attacks. Since the attacks are 3 the backend will indicate 3 different mitigation steps, starting from the most impactful to the least. In every steps one mitigation step will be applied, modifying the firewall rules, the network configuration, or the access control policies, and then the docker network will be reloaded to apply the changes. After each mitigation steps the attacks could be simulated again to check if the measures taken are effective or not.

3D impact matrix



(a) 3D Matrix - Low Impact Attack



(b) 3D Matrix - High Impact Attack

Figure 4.10: 3D impact matrices for two attack scenarios

The 3D impact matrix provides an intuitive visualization of attack severity through a three-dimensional representation. The matrix is structured as a $10 \times 10 \times 10$ cube, where each spatial dimension corresponds to one of the three attack impact metrics:

- **X-axis**: Asset Value (blue) - represents the criticality of targeted assets

- **Y-axis**: Vulnerability Severity (green) - represents the exploitability of discovered vulnerabilities
- **Z-axis**: Threat Strength (red) - represents the intensity or sophistication of the attack

The total attack impact is calculated as a weighted sum of these three metrics:

$$\text{Impact} = \text{AssetValue} \cdot w_{\text{asset}} + \text{VulnerabilitySeverity} \cdot w_{\text{vuln}} + \text{ThreatStrength} \cdot w_{\text{threat}}$$

where the weights can be interactively adjusted by the user to reflect different prioritization strategies.

The matrix visualization adapts dynamically based on the number of active metrics (i.e., metrics with non-zero values):

- **Single active metric**: A linear representation along the corresponding axis, with the line length proportional to the metric value (0-10). This clearly highlights which single dimension is driving the attack impact.
- **Two active metrics**: A planar representation in the corresponding 2D space, with each dimension scaled according to its metric value. This illustrates how two combined factors interact to produce the overall impact.
- **Three active metrics**: A full 3D volumetric representation where each dimension is scaled proportionally to the weighted contribution of its corresponding metric. The volume is normalized to reflect the total impact score, providing an intuitive sense of severity.

The color of the visualized cubes interpolates continuously from green (low impact) through yellow (medium impact) to red (high impact), based on the normalized total impact value. The matrix discretizes into three risk levels:

- **Low Risk**: Impact ≤ 3.3 (green)
- **Medium Risk**: $3.3 < \text{Impact} \leq 6.6$ (yellow)
- **High Risk**: Impact > 6.6 (red)

This dynamic visualization enables operators to intuitively understand not only the severity of an attack through color and magnitude, but also to identify which metrics are contributing most significantly to the overall risk. By interactively adjusting metric values and weights, users can explore impact sensitivity and better understand the implications of different threat scenarios, facilitating more informed decision-making regarding mitigation prioritization.

Since this 3D matrix is based on 6 different parameters (the 3 metrics and the 3 weights), this cannot be a classic risk assessment matrix which is based on only 3 parameters while the weights are equals and not modifiable. This 3D matrix offers a **qualitative** visualization of the attack impact, providing an intuitive representation of the severity of the attack based on the colors and an indication of which metrics are contributing more to the overall impact based on the number of cubes visualized in each dimension.

4.3.4 Attack simulation

The demonstration, as we saw before, has 10 steps and each one represents a specific phase of the demo.

The first step illustrates the initial topology of the network, showing all the nodes and the connections between them. Moreover, it presents the attacks that will be simulated during the demo, providing a brief description of each attack scenario with the responds needed to mitigate the attack.

Step 2 starts to prepare the environment for the attack simulation, cleaning the docker environment by removing all the containers and the networks. This step is necessary to ensure that the environment is clean and ready for the network deployment an to ensure a fresh environment for the demo

Listing 4.3: Script for cleaning enviroment

```
docker stop $(docker ps -a -q) 2>/dev/null || true
docker rm $(docker ps -a -q) 2>/dev/null || true
docker network prune -f
```

The third steps will starts the core component of the demo, which is VEREFOO and the scheduling backend. Two terminals will be opened and the commands to start the two components will be executed. The first terminal will be used to start VEREFOO, which will listen for incoming configuration files and will produce the output configuration file after checking the validity of the input. The second terminal will be used to start the scheduling backend, which will listen for incoming impact evaluations and will compute the mitigation steps based on the attack impact and the current network configuration.

Listing 4.4: Starting backend

```
gnome-terminal --title='VEREFOO' -- bash -c 'export␣
    LD_LIBRARY_PATH=../z3/bin:$LD_LIBRARY_PATH;␣java␣-Djava.
    library.path=../z3/bin␣-jar␣JarFiles/verefoo/verifoo-0.0.1-
    SNAPSHOT.jar;␣exec␣bash' 2>/dev/null

gnome-terminal --title='MITO' -- bash -c 'export␣LD_LIBRARY_PATH
    =../z3/bin:$LD_LIBRARY_PATH;␣java␣-D␣java.library.path=../z3/
    bin␣-jar␣JarFiles/mito/verifoo-0.0.1-SNAPSHOT.jar;␣exec␣bash'
    2>/dev/null
```

After having started the two core components of the demo, the next step(4) is to deploy the virtual network topology, initializing the docker environment staring from the file discussed before 4.3.2.

The deployment is based on 3 main steps:

- first step is to send the `Scada_Topology.xml` file to VEREFOO, which will check the validity of the configuration and will produce the `Output_Configuration.xml` file.

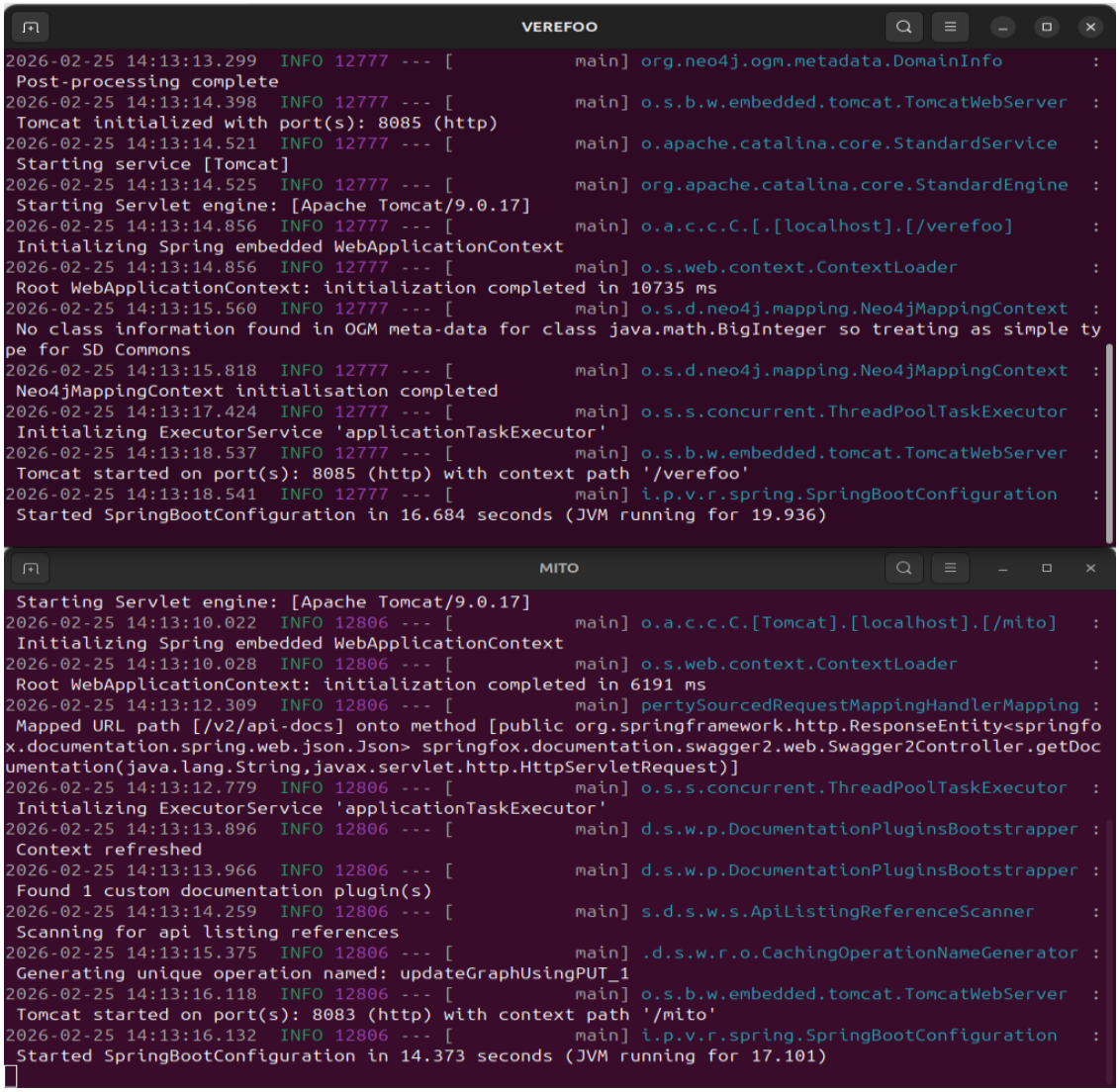


Figure 4.11: Starting VEREFOO and the scheduling backend

- second step is to send the `Output_Configuration.xml` file to Verefoo's backend, to generate all the files necessary for the configuration of the virtual network.
- third step is to deploy the docker network using the docker-compose file created by the backend, which will start all the containers and will configure them based on the configuration files created in the previous step.

All this 3 steps are automated in a series of commands that are executed in the terminal, allowing the user to deploy the virtual network with a single command.

The fifth step is to simulate the attack scenarios, which can be done through the GUI. The user can click on the nodes of the network topology to open the terminal of the corresponding container and execute the commands to simulate the attack scenarios.

The user is encouraged to simulate the attack scenarios one by one pinging from tha attacker to the target for all the 3 attacks. Moreover, the user can click on all

the firewalls to check the rules applied in any time.

The attacks can be freely visualized on the network topology, showing the attack paths and the affected nodes, allowing the user to understand how the attacks are propagating through the network and which nodes are affected.

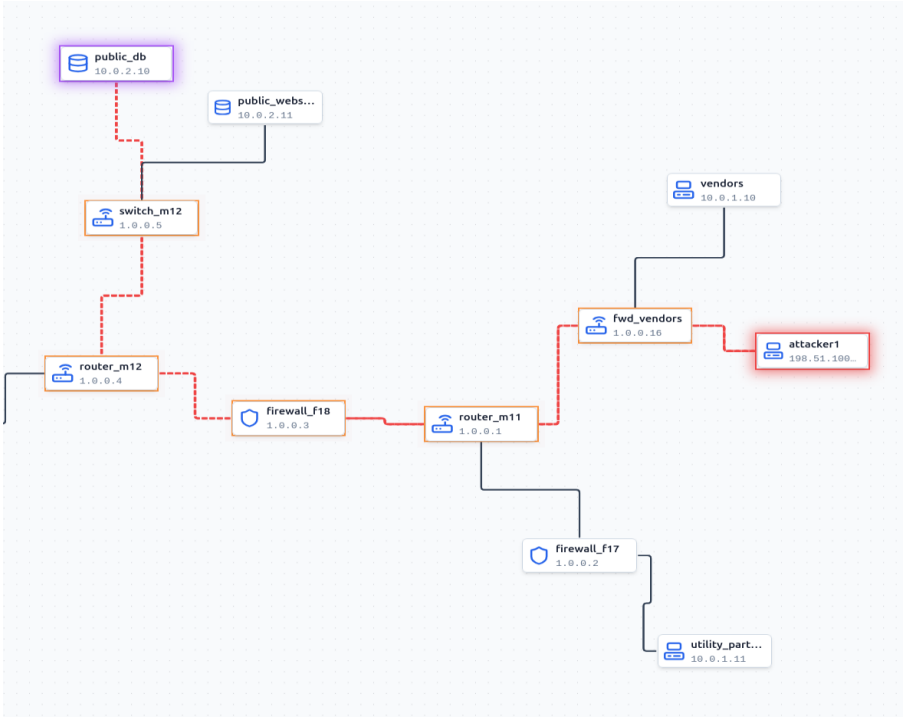


Figure 4.12: Attack 1 simulation on the network topology

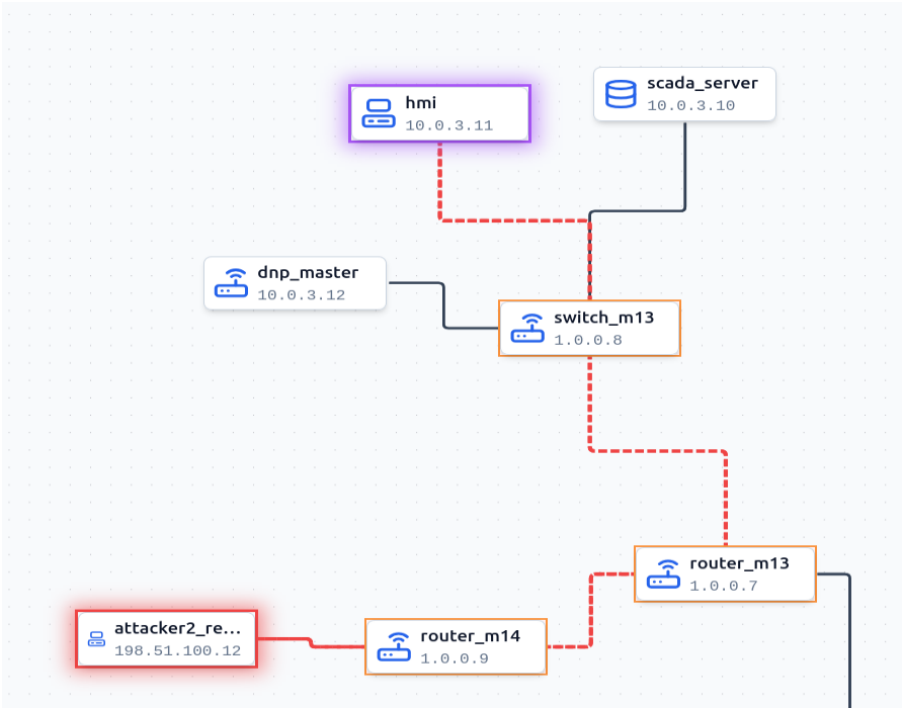


Figure 4.13: Attack 2 simulation on the network topology

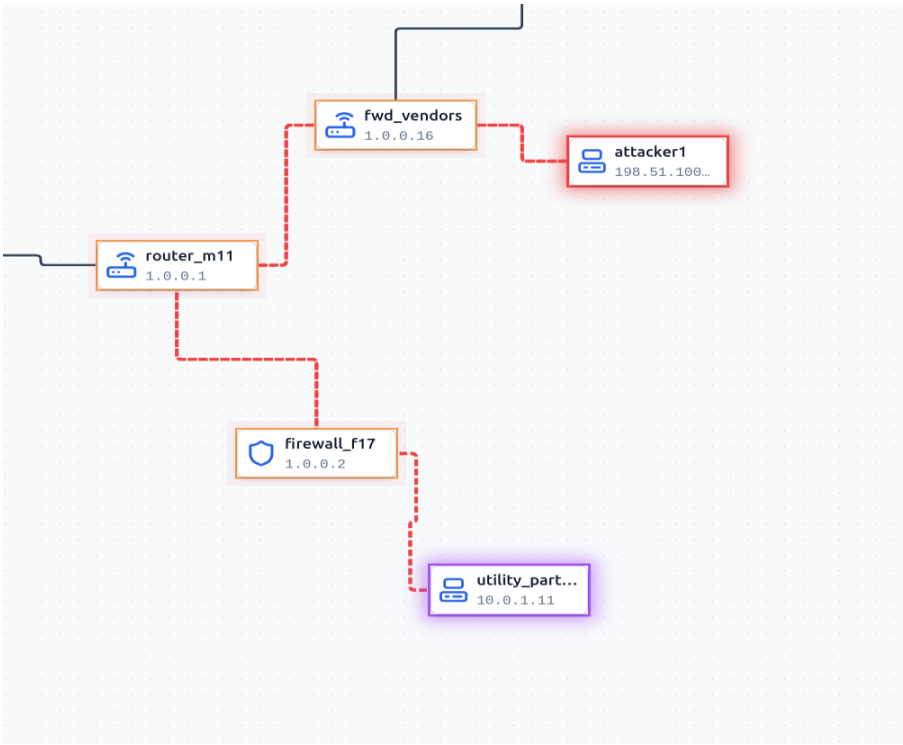


Figure 4.14: Attack 3 simulation on the network topology

One example of the attack simulation is shown in the figure 4.12, where the first attack is simulated, showing the attack path from the attacker to the target and the affected nodes. The same can be done for the other two attack scenarios, allowing the user to visualize all the attack paths and the traversed nodes for each attack scenario.

The effectiveness of the attack simulation can be checked by pinging from the attacker to the target, as shown in the figure 4.15, where the ping command is executed from the attacker to the target, showing the successful communication and the impact of the attack on the network.

The following steps(6) is to change and manage the attack parameters for all the 3 attack scenarios, allowing the user to modify the values of the metrics used to calculate the attack impact and to visualize how the change of the metrics influence the final impact and the mitigation steps needed to mitigate the attack.

In our simulation the attacks are characterized by the following values for the metrics:

- Attack 1: Asset Value = 7.7, Vulnerability Severity = 7.1, Threat Strength = 8, w1 = 0.33, w2 = 0.33, w3 = 0.34.

$$7.7 * 0.33 + 7.1 * 0.33 + 8 * 0.34 = 7.6$$

- Attack 2: Asset Value = 4.5, Vulnerability Severity = 5.5, Threat Strength = 5.6, w1 = 0.33, w2 = 0.33, w3 = 0.34.

$$4.5 * 0.33 + 5.5 * 0.33 + 5.6 * 0.34 = 5.2$$

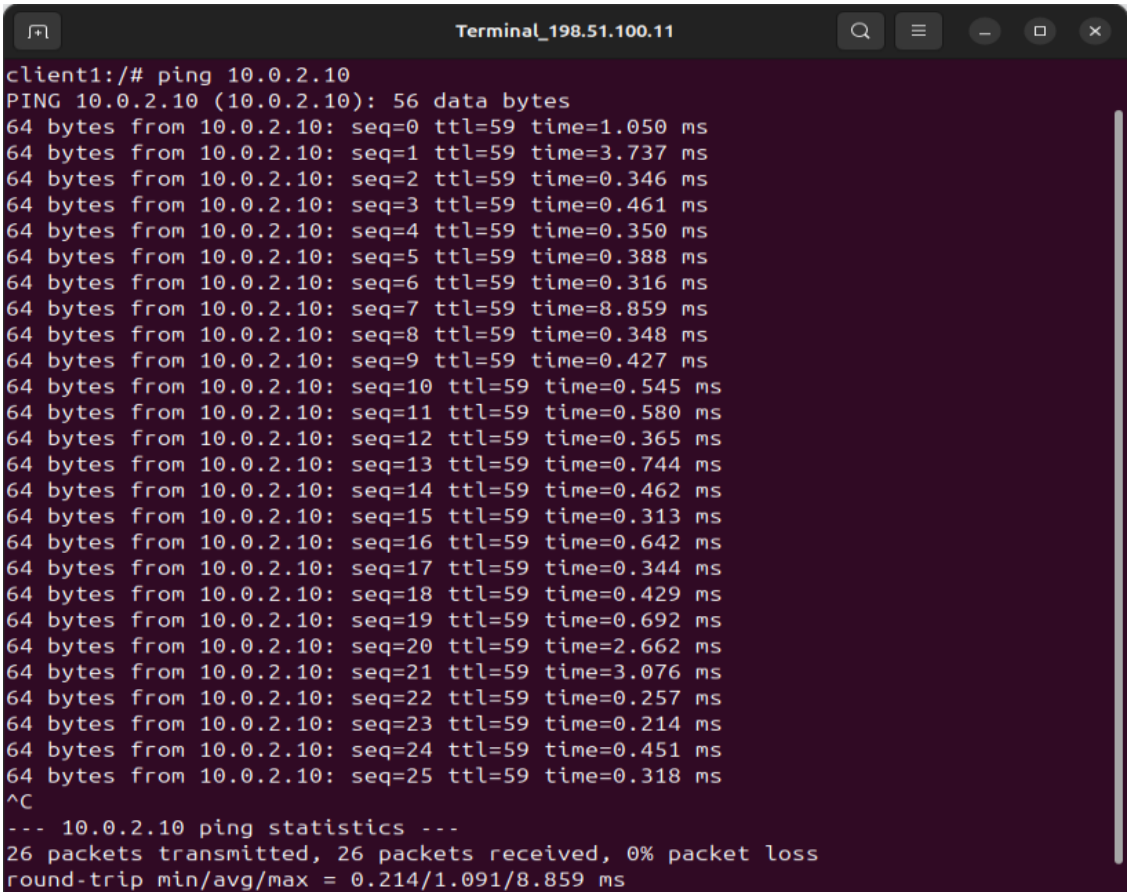


Figure 4.15: Ping Attack1

- Attack 3: Asset Value = 3.3, Vulnerability Severity = 3.8, Threat Strength = 3.4, w1 = 0.33, w2 = 0.33, w3 = 0.34.

$$3.3 * 0.33 + 3.8 * 0.33 + 3.4 * 0.34 = 3.5$$

After the information about the attack are chosen, the attack can start to be simulated using the commands calling the backend. The server in the backend will receive all the value of the components and will craft an ad-hoc XML file that will be sent to the scheduling backend. This file will be similar to the `SCADA_Topology.xml` file but with the addition of the attack impact information for each attack scenario on the bottom, as shown in the listing 4.5.

Additionally, the xml file will contain the information about the current network configuration and the next configuration in order to mitigate the attack. For example we can note that for firewall with ip 1.0.0.2, which is the one the first attack, the firewall is targeted with an `event = "u"` which means that the firewall rules will be updated to mitigate the attack. The configuration of the firewall pre-mitigation has the `name = "conf1"` while the configuration of the firewall post-mitigation has the `name = "conf2"`, allowing the backend to understand what are the changes needed to mitigate the attack and to apply them on the firewall. The same applies for all the other firewalls that are involved in the mitigation steps for all the 3 attack scenarios.

For the firewall with ip 1.0.0.15, instead, there is an `event = "a"` which means that the firewall will be added to the network configuration to mitigate the attack, since in the initial configuration there is no firewall on that node. The configuration of the firewall post-mitigation has the `name = "conf2"`, while there is no pre-mitigation configuration since the firewall is not present in the initial configuration.

Note that the listing 4.5 is just a snippet of the actual XML file sent to the backend, which contains all the information about the network configuration and the attack impact for all the attack scenarios.

Listing 4.5: Output topology XML configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<NFV xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
    noNamespaceSchemaLocation="../xsd/nfvSchema.xsd">
  <graphs>
    <graph id="0">
      <node functional_type="FIREWALL" name="1.0.0.2" event="u">
        <neighbour name="10.0.1.11" />
        <neighbour name="1.0.0.1" />
        <configuration description="A␣simple␣description" name="conf1">
          <firewall defaultAction="ALLOW" index="1" />
        </configuration>
        <configuration description="A␣simple␣description" name="conf2">
          <firewall defaultAction="ALLOW" index="2" />
          <elements>
            <action>DENY</action>
            <source>198.51.100.11</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>DENY</action>
            <source>198.51.100.11</source>
            <destination>10.0.1.11</destination>
            <protocol>OTHER</protocol>
            <src_port>*</src_port>
            <dst_port>*</dst_port>
          </elements>

        </configuration>
      </node>
      <node functional_type="FIREWALL" name="1.0.0.3" event="u">
        <neighbour name="1.0.0.1" />
        <neighbour name="1.0.0.4" />
        <configuration description="A␣simple␣description" name="conf1">
          <firewall defaultAction="ALLOW" index="1" />
        </configuration>
        <configuration description="A␣simple␣description" name="conf2">
          <firewall defaultAction="ALLOW" index="2" />
          <elements>
            <action>DENY</action>
            <source>198.51.100.11</source>
            <destination>10.0.2.10</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>10.0.2.10</destination>
            <protocol>OTHER</protocol>
            <src_port>*</src_port>
            <dst_port>*</dst_port>
          </elements>
        </configuration>
      </node>
      <node functional_type="FIREWALL" name="1.0.0.6">
        <neighbour name="1.0.0.4" />
        <neighbour name="1.0.0.7" />
        <neighbour name="1.0.0.10" />
        <configuration description="A␣simple␣description" name="conf1">
          <firewall defaultAction="ALLOW" index="1" />
        </configuration>
        <configuration description="A␣simple␣description" name="conf2">
          <firewall defaultAction="DENY" index="2" />
        </configuration>
      </node>
      <node functional_type="FIREWALL" name="1.0.0.11">
        <neighbour name="1.0.0.10" />
        <neighbour name="1.0.0.12" />
        <configuration description="A␣simple␣description" name="conf1">
          <firewall defaultAction="ALLOW" index="1" />
        </configuration>
        <configuration description="A␣simple␣description" name="conf2">
          <firewall defaultAction="DENY" index="2" />
        </configuration>
      </node>
      <node functional_type="FIREWALL" name="1.0.0.14">
        <neighbour name="1.0.0.12" />
        <neighbour name="10.0.5.10" />
        <configuration description="A␣simple␣description" name="conf1">
          <firewall defaultAction="DENY" index="1">
            <elements>
              <action>ALLOW</action>
              <source>10.0.5.10</source>
              <destination>10.0.4.10</destination>
              <protocol>OTHER</protocol>
              <src_port>*</src_port>
              <dst_port>*</dst_port>
            </elements>
            <elements>
              <action>ALLOW</action>
              <source>10.0.5.10</source>
              <destination>10.0.4.11</destination>
              <protocol>OTHER</protocol>
              <src_port>*</src_port>
              <dst_port>*</dst_port>
            </elements>
            <elements>
```

```
    <action>ALLOW</action>
    <source>10.0.5.11</source>
    <destination>10.0.4.10</destination>
    <protocol>OTHER</protocol>
    <src_port>*</src_port>
    <dst_port>*</dst_port>
  </elements>
  <elements>
    <action>ALLOW</action>
    <source>10.0.5.11</source>
    <destination>10.0.4.11</destination>
    <protocol>OTHER</protocol>
    <src_port>*</src_port>
    <dst_port>*</dst_port>
  </elements>
  <elements>
    <action>ALLOW</action>
    <source>10.0.5.12</source>
    <destination>10.0.4.10</destination>
    <protocol>OTHER</protocol>
    <src_port>*</src_port>
    <dst_port>*</dst_port>
  </elements>
  <elements>
    <action>ALLOW</action>
    <source>10.0.5.12</source>
    <destination>10.0.4.11</destination>
    <protocol>OTHER</protocol>
    <src_port>*</src_port>
    <dst_port>*</dst_port>
  </elements>
  <elements>
    <action>ALLOW</action>
    <source>10.0.4.10</source>
    <destination>10.0.5.10</destination>
    <protocol>OTHER</protocol>
    <src_port>*</src_port>
    <dst_port>*</dst_port>
  </elements>
  <elements>
    <action>ALLOW</action>
    <source>10.0.4.10</source>
    <destination>10.0.5.11</destination>
    <protocol>OTHER</protocol>
    <src_port>*</src_port>
    <dst_port>*</dst_port>
  </elements>
  <elements>
    <action>ALLOW</action>
    <source>10.0.4.10</source>
    <destination>10.0.5.12</destination>
    <protocol>OTHER</protocol>
    <src_port>*</src_port>
    <dst_port>*</dst_port>
  </elements>
  <elements>
    <action>ALLOW</action>
```

```
            <source>10.0.4.11</source>
            <destination>10.0.5.10</destination>
            <protocol>OTHER</protocol>
            <src_port>*</src_port>
            <dst_port>*</dst_port>
          </elements>
          <elements>
            <action>ALLOW</action>
            <source>10.0.4.11</source>
            <destination>10.0.5.11</destination>
            <protocol>OTHER</protocol>
            <src_port>*</src_port>
            <dst_port>*</dst_port>
          </elements>
          <elements>
            <action>ALLOW</action>
            <source>10.0.4.11</source>
            <destination>10.0.5.12</destination>
            <protocol>OTHER</protocol>
            <src_port>*</src_port>
            <dst_port>*</dst_port>
          </elements>
        </firewall>
      </configuration>
      <configuration description="A␣simple␣description" name="conf2">
        <firewall defaultAction="DENY" index="2" />
      </configuration>
    </node>
    <node functional_type="FIREWALL" name="1.0.0.15" event="a">
      <neighbour name="1.0.0.7" />
      <neighbour name="1.0.0.8" />
      <configuration description="F22" name="conf2">
        <firewall defaultAction="DENY" index="2" />
      </configuration>
    </node>
  </graph>
</graphs>
<Constraints>
  <NodeConstraints />
  <LinkConstraints />
</Constraints>
<PropertyDefinition>
  <Property graph="0" name="AttackMitigationProperty" src="198.51.100.11"
  dst="10.0.2.10">
    <impact i1="7.7" i2="7.1" i3="8" />
    <weights w1="0.33" w2="0.33" w3="0.34" />
  </Property>
  <Property graph="0" name="AttackMitigationProperty" src="198.51.100.12"
  dst="10.0.3.11">
    <impact i1="4.5" i2="5.5" i3="5.6" />
    <weights w1="0.33" w2="0.33" w3="0.34" />
  </Property>
  <Property graph="0" name="AttackMitigationProperty" src="198.51.100.11"
  dst="10.0.1.11">
    <impact i1="3.3" i2="3.8" i3="3.4" />
    <weights w1="0.33" w2="0.33" w3="0.34" />
  </Property>
```

```
    </PropertyDefinition>
    <ImpactMode>economic</ImpactMode>
</NFV>
```

The scheduling backend will receive the XML file and will analyze the attack impact for all the attack scenarios, prioritizing the mitigation steps based on the impact of the attacks. The backend will then indicate the steps needed to mitigate the attacks, starting from the most impactful to the least. For each step, the backend will indicate which firewall needs to be updated or added, and what are the changes needed to mitigate the attack as we can observe in the listing 4.6.

Listing 4.6: Output topology XML configuration

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Scheduling>
    <firewall event="u" ip="1.0.0.2" state="3"/>
    <firewall event="u" ip="1.0.0.3" state="1"/>
    <firewall event="n" ip="1.0.0.6" state="2"/>
    <firewall event="n" ip="1.0.0.11" state="2"/>
    <firewall event="n" ip="1.0.0.14" state="2"/>
    <firewall event="a" ip="1.0.0.15" state="2"/>
</Scheduling>
```

The output 4.6 have all the information needed to order the mitigation steps we need to apply.

There are 3 possible events that can be indicated in the output file for each firewall:

- **u**: which means that the firewall rules will be updated to mitigate the attack, since the firewall is already present in the initial configuration.
- **a**: which means that the firewall will be added to the network configuration to mitigate the attack, since in the initial configuration there is no firewall on that node.
- **n**: which means that there is no need to update or add a firewall to mitigate the attack, since the current configuration is already effective to mitigate the attack.

The ip indicates which firewall is targeted, but the most important information is given by the state attribute. This attribute indicates which is the order of the mitigation steps, a lower state indicates a higher priority, so in the example of the listing 4.6 the first step to apply is to update the firewall with ip 1.0.0.3, then to add the firewall with ip 1.0.0.15 and finally to update the firewall with ip 1.0.0.2.

The following 3 steps(7,8,9) are dedicated to apply the mitigation steps indicated by the backend, applying one step at a time and then simulating the attack scenarios again to check if the measures taken are effective or not. The order of the mitigation steps is dynamic, so in every simulation with different attack parameters the order of the mitigation steps could change, allowing the user to understand how the change of the attack parameters can influence the mitigation steps needed to mitigate the attack.

On one hand we are taking in consideration the attack number 1 and we will go through the cycle to mitigate the attack into one single steps. The first step is to call the backend in order to prepare the file needed to update the single firewall. For the simulation that we are considering the backend will find that, based on the output file shown in the listing 4.6, the first step to mitigate the attack is to update the firewall with ip 1.0.0.3. The backend will prepare the file needed to update the firewall modifying the rules of the firewall to mitigate the attack. It will take the initial `SCADA-Topology.xml` file and modify the rules as stated in the `name = "conf2"` for the corresponding firewall, creating a new file named `SCADA-Topologystep1.xml` that will be sent to VEREFOO. After having regenerate the configuration files the docker network will be redeployed with the new configuration, applying the changes needed to mitigate the attack.

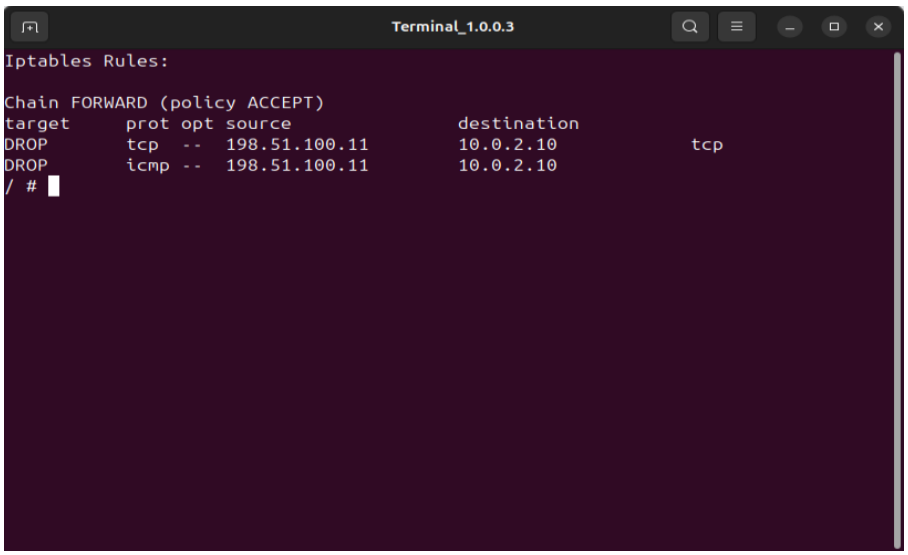


Figure 4.16: Attack 1 blocked after the first mitigation step

After the first mitigation step, clicking on the firewall, we will see the updated rules, which will be the same as the ones indicated in the `name = "conf2"`.(figure 4.16)

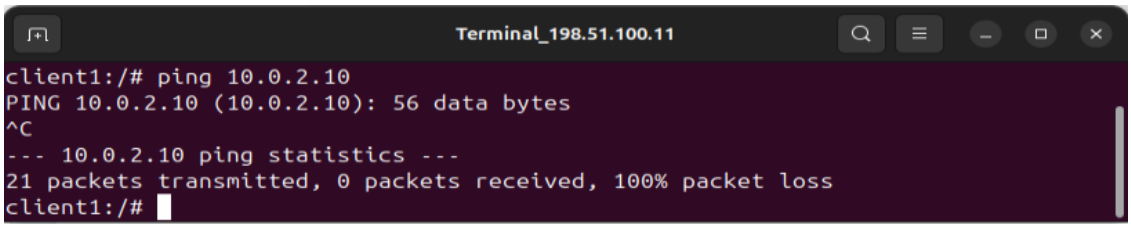


Figure 4.17: Attack 1 blocked visualized on the network topology

With this different configuration the attack will be mitigated, since the firewall rules will block the attack traffic, as we can see in the figure 4.17 where the ping from the attacker to the target is blocked by the firewall, showing the effectiveness of the mitigation step taken.

On the other hand, if we consider the attack number 2, the steps are similar, but the backend will act in a different way. Based on the output file 4.6 for our

simulation the second step is adding a firewall on the node with ip 1.0.0.15. The first step is the same with the call to the backend to prepare the file needed to add the firewall, but in this case the backend will modify the `SCADA-Topology.xml` file adding a new firewall on the node with the correct IP, the right rules to mitigate the attack and modifying the connections to the new firewall, creating a new file named `SCADA-Topologystep2.xml` that will be sent to VEREFOO. After having regenerate the configuration files the docker network will be redeployed with the new configuration, applying the changes needed to mitigate the attack.

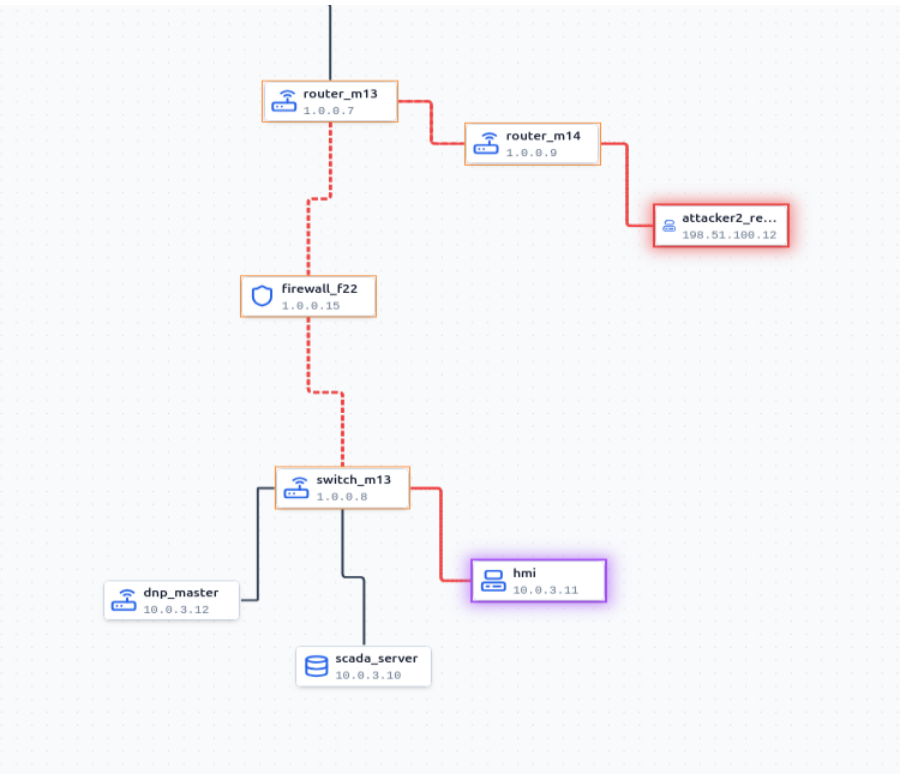


Figure 4.18: New configuration with the added firewall

The topology after the second mitigation step is shown in the figure 4.18, where we can see the new firewall added and the new connections. The rules added are similar to the ones shown in figure 4.16 but with the correct IP for the firewall and the correct rules to mitigate the second attack scenario. With this new configuration the second attack will be mitigated, since the new firewall will block the attack traffic in a similar way as the first attack, showing the effectiveness of the mitigation step taken.

The third and last attack is not shown in depth since the steps are similar to the ones described for the first attack, with the modification of the firewall rules on the firewall with ip 1.0.0.2.

After the 3 mitigation steps are applied, the attack scenarios can be simulated again to check if the measures taken are effective or not. Starting over the simulation of the attack scenarios, it is possible to play with the attack parameters to see how the change of the attack impact can influence the mitigation steps needed to mitigate the attack, showing how the framework can help in understanding the

attack scenarios and in taking the right mitigation steps based on the attack impact and the current network configuration.

Chapter 5

Conclusions and future work

5.1 Conclusions

The progressive growth in size, complexity, and heterogeneity of modern networks has made the traditional trial-and-error approach to security configuration increasingly inadequate. As discussed throughout this thesis, manual configuration and updates of security functions are not only difficult to scale, but are also intrinsically exposed to misconfigurations and human errors, which in turn may lead to service disruption or policy violations. Within this context, network security automation enabled by paradigms such as Network Functions Virtualization (NFV) and Software-Defined Networking (SDN) represent a key direction to support operators in preserving security properties while coping with frequent changes.

This thesis focused on the design and implementation of a Docker-based demonstrator, managed through a Graphical User Interface, that showcases an automated workflow for configuring and updating distributed firewalls. The core of the approach relies on VEREFOO (VErified REFinement and Optimized Orchestration), which frames the firewall allocation and configuration problem as a partial weighted Maximum Satisfiability Modulo Theories (MaxSMT) instance and employs the Z3 solver to provide formally correct and optimal configurations. In the demonstrator, VEREFOO is used as the engine that validates user-provided network descriptions and intents, synthesizes high-level firewall configurations, and generates low-level rules (e.g., `iptables`) along with the artifacts required to deploy the resulting virtual network through Docker Compose.

On top of this automated configuration layer, the demonstrator integrates a scheduling backend that orders mitigation actions when multiple attack scenarios are considered simultaneously. In particular, the workflow operationalize the idea of applying reconfiguration steps in a prioritized sequence, which is crucial in order to reduce the exposure window during the transient states that may occur when updating distributed security functions. The prioritization is driven by an impact evaluation model based on three metrics: Asset Value, Vulnerability Severity, and Threat Strength combined through a weighted average. By allowing the user to interactively manipulate these parameters, the GUI makes explicit the relationship between perceived risk and the resulting mitigation schedule.

The demonstrator was instantiated on a SCADA-inspired topology and used to simulate three representative attack scenarios (Denial of Service, data exfiltration, and infiltration/data injection). Moreover, the visualization layer, implemented with React Flow and automatic layouting via Elkjs, supports interoperability by enabling the inspection of the evolving topology and the affected paths during attack simulation and mitigation.

At the same time, the work also highlights the gap between a controlled demonstration environment and production-grade deployments. The current prototype relies on a predefined set of security intents and on simplified assumptions when translating a realistic industrial scenario into a virtual topology; it also validates connectivity primarily through interactive, scenario-driven checks. These limitations motivate the future research and engineering directions outlined in the next section.

5.2 Limitation and Future work

The demonstrator presented in this thesis represents a proof of concept for the integration of automated configuration schedulers within a user-friendly interface. However, there are several limitations that need to be addressed in order to make the framework more robust and applicable to real-world scenarios. Both the representation and the security intents are currently simplified and predefined, which limits the flexibility and scalability of the framework, even if the Framework used are designed to scale well.

Future work could focus on several aspects to enhance the framework staring from integrating monitoring and detection capabilities to automatically trigger the mitigation process based on real-time data, rather than relying on user-initiated simulations or manage to simulate real attack scenarios and not only the one that are predefined in the system.

Bibliography

[1] D. Bringhenti, F. Pizzato, R. Sisto, and F. Valenza, "Autonomous attack mitigation through firewall reconfiguration," *Int. J. Netw. Manag.*, vol. 35, no. 1, Dec. 2024. [Online]. Available: https://doi.org/10.1002/nem.2307

[2] 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.

[3] 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.

[4] F. Pizzato, D. Bringhenti, R. Sisto, and F. Valenza, "A demonstration of an autonomous approach for cyberattack mitigation," in *2025 21st International Conference on Network and Service Management (CNSM)*, 2025, pp. 1–3.

[5] D. Bringhenti and F. Valenza, "Optimizing distributed firewall reconfiguration transients," *Computer Networks*, vol. 215, p. 109183, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S138912862200281X

[6] D. Bringhenti, G. Marchetto, R. Sisto, and F. Valenza, "Automation for network security configuration: State of the art and research trends," *ACM Comput. Surv.*, vol. 56, no. 3, Oct. 2023. [Online]. Available: https://doi.org/10.1145/3616401

[7] Verizon, "2024 data breach investigations report — verizon," 2024. [Online]. Available: https://www.verizon.com/business/resources/reports/2024-dbir-data-breach-investigations-report.pdf

[8] D. Bringhenti, G. Marchetto, R. Sisto, S. Spinoso, F. Valenza, and J. Yusupov, "Improving the formal verification of reachability policies in virtualized networks," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 713–728, 2021.

[9] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[10] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: an intellectual history of programmable networks," vol. 44, no. 2, pp. 87–98, Apr. 2014. [Online]. Available: https://doi.org/10.1145/2602204.2602219

[11] 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.

[12] L. De Moura and N. Bjørner, "Z3: an efficient smt solver," in *Proceedings of the Theory and Practice of Software, 14th International Conference*

on Tools and Algorithms for the Construction and Analysis of Systems, ser. TACAS'08/ETAPS'08. Berlin, Heidelberg: Springer-Verlag, 2008, p. 337–340.

[13] 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.

[14] Docker, "What is docker - containers," 2026. [Online]. Available: https://docs.docker.com/get-started/docker-overview/#containers

[15] ——, "What is docker - containers," 2026. [Online]. Available: https://www.docker.com/resources/what-container/

[16] T. Combe, A. Martin, and R. Di Pietro, "To docker or not to docker: A security perspective," *IEEE Cloud Computing*, vol. 3, no. 5, pp. 54–62, 2016.

[17] D. Bringhenti and F. Valenza, "Greenshield: Optimizing firewall configuration for sustainable networks," *IEEE Transactions on Network and Service Management*, vol. 21, no. 6, pp. 6909–6923, 2024.

[18] D. Bringhenti, S. Bussa, R. Sisto, and F. Valenza, "A two-fold traffic flow model for network security management," *IEEE Transactions on Network and Service Management*, vol. 21, no. 4, pp. 3740–3758, 2024.

[19] 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.

[20] React-Flow, "React flow - documentation," 2026. [Online]. Available: https://reactflow.dev/learn

[21] Eclipse, in *Eclipse Layout Kernel*.

[22] kieler, "Elkjs - github repository." [Online]. Available: https://github.com/kieler/elkjs

[23] P. Wlazlo, K. Price, C. Veloz, A. Sahu, H. Huang, A. Goulart, K. Davis, and S. Zounouz, "A cyber topology model for the texas 2000 synthetic electric power grid," in *2019 Principles, Systems and Applications of IP Telecommunications (IPTComm)*, 2019, pp. 1–8.

[24] F. Coriale, "An attack risk assessment model for network security automation." in *Master's thesis*, 2025.

[25] FIRST.Org, "Information disclosure, information tampering and denial of service (dos) vulnerability in genesis64 and mc works64." [Online]. Available: https://www.cve.org/CVERecord?id=CVE-2024-7587

[26] First.org, "Sql injection in langchain-ai/langchain." [Online]. Available: https://www.cve.org/CVERecord?id=CVE-2024-8309

[27] ——, "ininet solutions spidercontrol scada pc hmi editor path traversal,." [Online]. Available: https://www.cve.org/CVERecord?id=CVE-2024-10313