POLITECNICO DI TORINO

Master's Degree in Computer Engineering
Cybersecurity

Master's Degree Thesis

Security formal verification engine for OT infrastructures

Supervisors

Prof. Cataldo Basile

Ing. Alessio Viticchié, PhD

Candidate

Giulio Sunder

November 2024

Summary

The work presented in this dissertation has been carried out in the context of an internal R&D project of the *Alphawaves Srl.* company, which hosted the activities of this thesis.

Specifically the project addresses the need for cybersecurity solutions within the world of operational technology (OT), by proposing a plug and play tool capable of performing automatic scanning, validation and reporting of the status of security within an OT network, in a continuous monitoring cycle.

Within the context of the project, this thesis focuses on the development of a formal verification engine that receives as input a network description, containing information about host and network configuration, topology, and potential vulnerabilities; and generates possible attack paths and sequences of validation tasks to be executed in order to test the actual exploitability of a target system.

The computation of the attack paths is achieved by leveraging MulVAL, as suggested by the analysed literature, a popular open source framework that allows to conduct multi-host and multi-stage vulnerability analysis through the use of attack graphs, and extending it to better represent common OT scenarios.

In order to ensure easy integration within the global company project, a standard network description model has been defined for representing OT networks in a complete and structured manner. Consequently, a parser module has been developed to translate the structured input data over to the modelling language used by MulVAL to represent its target networks. A custom pruning algorithm has then been deployed to reduce the high complexity of the attack graphs generated by MulVAL. Following which, the attack paths are then extracted from the reduced graph and saved into a corresponding data structure. For each successfully extracted attack path, a sequence of validation tasks, for which task description models have been defined, is selected according to the type of exploits the path entails. The final output is then stored in an appropriately defined data model, ready to be used for the execution phase (not included in the scope of this work).

The validity of the formal verification engine proposed in this work has been tested against a manually constructed OT network configuration, considering two different initial attacker access points. And for both cases it successfully identified

and extracted a list of attack paths and corresponding validation tasks to execute.

Thanks to *AlphaWaves*, it has also been possible to present the findings of this thesis in an academic research paper at the 2024 ICSC Intelligent Cybersecurity Conference held in Valencia. The paper is entitled "*Enhancing OT Threat Modelling: An Effective Rule-Based Approach for Attack Graph Generation*", and it documents the methodology brought forward by this work regarding the use of attack graphs for developing standard solutions in the context of OT security assessment.

Acknowledgements

I would like to thank the AlphaWaves Srl. company for giving me the possibility to expand my knowledge in regards to cybersecurity in the field of operational technology, and challenge myself by taking part in a project addressing real world scenarios.

A special thanks goes to my company supervisor, Ing. Alessio Viticchié, to Prof. Alessandro Aliberti, to Dott. Alberto Colletto, and to Dott. Sara Raimondi for all the support they have given me throughout the period of the thesis, and especially for the incredible opportunity to write my first academic research paper documenting the project I worked on.

I express my deepest gratitude also to my university supervisor, Prof. Cataldo Basile, for taking me on and for providing important feedback regarding both the thesis work, and the research paper.

To all the other people of the company, I would like to extend my most sincere appreciation for welcoming me in a fun and positive working environment, and for making me feel part of the team.

Last but not least, a great thanks goes to my family, for all the love and support they have given me throughout not only my university career but my whole life, and without which I would not be writing this thesis today.

Table of Contents

List of Tables

List of Figures

Acronyms

OT
operational technology

ICS
industrial control system

SCADA
supervisory control and data acquisition

HMI
human machine interface

PLC
programmable logic controller

RTU
remote terminal unit

DCS
distributed control system

EWS
engineering workstation

IED
intelligent electronic device

IIoT
industrial internet of things

AG

attack graph

LAG

logical attack graph

EDG

exploit dependency graph

SG

state graph

AT

attack tree

Chapter 1

Introduction

In the fast paced world that we live in today, always more dependent on technology to satisfy everyone's daily needs, the importance of cybersecurity is becoming the utmost priority and therefore it requires special attention in all branches of technology and must be addressed from the very beginning of the design phase of each project, on a par with the actual functionality requirements.

In regards to the world of Operational Technology (OT), which is responsible for automation and control of physical processes, cybersecurity was never given enough importance, but never more so than today, with the increasing need of internet connectivity, has it become one of those fields that requires it the most.

In order to address this issue, this work contributes to an internal project of the *AlphaWaves* company, which, as a whole, aims at developing a plug and play system capable of performing automatic scanning, vulnerability analysis, testing, risk assessment, and impact analysis of OT networks, with the goal of producing detailed reports about the overall status of security, including information regarding discovered issues, criticality and suggested mitigation strategies.

Specifically, this thesis will focus on the development of a formal verification engine capable of computing possible attack paths that an attacker could undertake to compromise a target network, by exploiting information regarding topology and vulnerabilities obtained from an initial scanning phase(which is not in the scope of this project); and consequently planning a list of tasks to be executed, in order to test the actual exploitability of the discovered paths(the execution of these tasks is also not in the scope of this project).

The need for this project comes from the fact that there are not many viable available solutions to automatically perform a complete security assessment of a network from start to finish, especially when it comes to OT. So the goal of the project is to provide a packaged, all-in-one and simple to use tool for automating this process. This can be achieved by leveraging and extending already existing tools, such as network scanners and attack graph generators, and integrating them

into a pipelined process.

The thesis is divided into six chapters.

Chapter 1 contains an introduction to the global project, and it briefly addresses the goals and the need for such a project.

Chapter 2 contains the thesis' background. It provides contextualization about the world of OT and describes how cybersecurity relates to this field. Some examples of attacks against OT infrastructures are also given in this chapter.

Chapter 3 discusses the state of the art. It starts by giving a brief theoretical overview of the architectural design of the project. It then proceeds to provide a detailed analysis of the existing literature and afterwards moves on to describe why the MulVAL framework was chosen as the best option for the job compared to others, how it works, and what are the limitations that this work will try to overcome.

Chapter 4 discusses the methodology that has been used to build the formal verification engine. It describes the main steps such as how the input model was designed, how MulVAL was extended, how the attack graphs were pruned, and how the validation task models were designed and selected.

Chapter 5 presents two test cases run on a test network configuration and discusses the obtained results.

Lastly there is Chapter 6, containing final remarks about the project and how it could be improved going forward.

Following the last chapter, Appendix A has been added for supplying the data structures that are addressed throughout the thesis in order not to disrupt the flow of the reader.

With the help of *AlphaWaves*, the work discussed in this thesis, has been used to write an academic research paper that has been presented at the *ICSC Intelligent Cybersecurity Conference* held in Valencia this year. The paper [1] is entitled "*Enhancing OT Threat Modelling: An Effective Rule-Based Approach for Attack Graph Generation*" and it summarizes the main methodology steps that have been used to enhance the MulVAL framework in order to incorporate it into the context of a standard solution for OT security assessment.

Chapter 2

Background

This chapter focuses on providing the reader with a general overview of the world of operational technology (OT), and how the need for security constitutes a key role in this field, alongside providing the main concepts and definitions that are used throughout the remainder of this thesis.

The chapter follows the definitions given by the National Institute of Standards and Technology (NIST) in their guide to OT security [2].

In the final part of the chapter, the need for a standard automatic solution for cybersecurity assessment in OT environments is introduced as it constitutes the main topic that this work tries to address.

2.1 General overview of OT

Following the official definition given by NIST [2], Operational Technology consists of a broad range of physical systems and devices that interact with the physical environment. These systems and devices detect or cause a direct change through the monitoring, and/or control of devices, processes, and events.

An OT system can be seen as a combination of a process and a controller. The process is the actual physical procedure that produces outputs, which the system is interested in regulating. While the controller is responsible for regulating the process in order to maintain conformity between the output and the system requirements.

There are three possible ways a system could be configured: open loop, closed loop, manual mode. In the open loop configuration the output is controlled by established parameters; closed loop differs from the previous, as in this case the output is fed back to the system as input and therefore plays a part in maintaining the desired control objective. Last but not least there is the possibility to have a manual configuration in which the process is completely controlled by humans.

OT differs from traditional information technology (IT), as the first focuses on the control and monitoring of physical systems, and therefore requires real time management capabilities for maintaining efficiency and safety. While the latter is more data oriented, as its main goal is to collect, process, and store data to effectively support decision making and communication.

Also security priorities differ substantially between the two, as OT focuses primarily on maintaining availability, real time operation, safety, component lifetime. While IT, being more data oriented, prioritizes properties such as data integrity and confidentiality at the cost of performance speed.

2.1.1 Role of OT in today's world

Nowadays, with the increasing global population and the constantly growing demand for resources, the need for automation is becoming of the utmost importance in order to meet the needs of today's life standards. Especially in regards to those type of tasks that require constant and efficient monitoring and operation, don't have high error tolerance, and cannot afford much downtime. Many can relate to this description and therefore fall under the category of OT.

This branch of technology is of paramount importance in today's world, and even though most people may not notice its presence in their everyday lives, everyone relies on it and is somehow impacted by its correct functioning, as it drives most of the processes that human life has become dependent on, some of which encountered on a daily basis, such as public transportation.

Many industries and infrastructures also depend on OT to function properly, including critical infrastructures of a variety of different fields, to name a few: chemical, commercial facilities, critical manufacturing, energy, healthcare, food and agriculture, transportation, water distribution, nuclear.

The subset of OT responsible for controlling automation within industrial environments is denoted by the term ICS, which stands for industrial control systems. Within this field the terms OT and ICS are frequently used interchangeably.

If for any reason an OT system responsible for controlling one of the previously mentioned infrastructures were to encounter a failure, it could potentially result in massive losses including financial, environmental damage, and straight up loss of human life. These are the main reason why OT, specifically safety and security in this context, play such an important role in seamlessly assisting the normal unfolding of everyday life.

2.1.2 Evolution of OT over the years

Since it has been around, OT has evolved over the years trying to keep up with the fast paced progress of IT technology. Many solutions that have in fact been

developed in the context of IT, have been successively integrated into OT scenarios. Analog mechanical controls, that used to be the backbone of OT, got replaced by embedded digital controls, in order to favor inter-connectivity among devices and the internet, which provides edge nodes with more computing power, effectively decentralizing the decision making process. OT greatly benefits from these capabilities, as they can be leveraged to boost productivity, decision making, safety, and system lifetime.

This approach was inspired by the rise in popularity of the Internet of Things (IoT) paradigm, which allows to make use of the more advanced features available for IT, such as internet connectivity, data collection, data analytics, and security, to improve automation capabilities.

However, although the deployment of smart devices able to connect to the internet greatly enhances the scope of what OT is capable of handling, it also increases the attack surface. In fact, the more functionalities and responsibilities a device is associated with, the more likely it is to expose security issues that could potentially lead to a system malfunctioning or being compromised. Therefore cybersecurity in this field must be addressed with special care, risk and impact parameters should be thoroughly evaluated, and proper measures must be chosen on the basis of these parameters.

2.1.3 Components of an OT network

An OT system relies on control loops, consisting of sensors, actuators and controllers, for performing some regulation of a physical process.

Sensors are responsible for taking real time measurements in order to produce a clear snapshot of the process' status to send to the controller.

Controllers are the actual brain of the regulating process, as it is responsible for interpreting the input received by the sensors and computing values for the control variables to conform the process behaviour to the desired one. This is achieved through a control algorithm which performs calculations based on inputs and other manually set parameters called set points.

While actuators are the physical components, such as valves and motors, that are used to directly interact with and control the actual process behaviour at a physical level.

In most cases there must be a way for personnel to interact with the system in order to monitor its functioning, or to reconfigure set points and algorithms based on the goals that must be reached. This is where human machine interfaces come into play.

Last but not least, there must also be a module capable of providing support for diagnostics and maintenance, in order to prevent or recover from possible failures.

A general overview of the system flow described above is given in Figure 2.1.

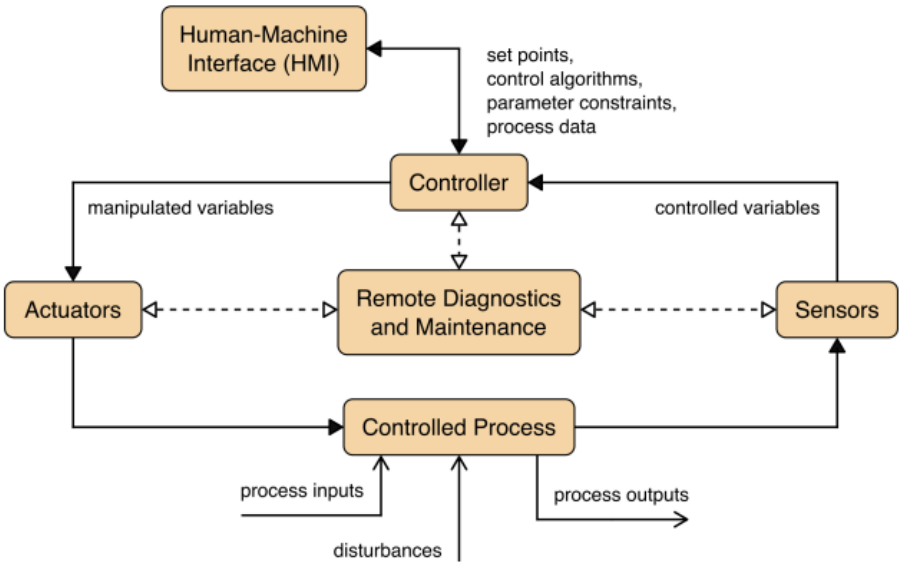

Figure 2.1: Standard OT system control loop taken from the NIST guide for OT [2]

In terms of the actual hardware equipment used to build such a system, the main components that can be found in an OT network are: human machine interfaces (HMI), programmable logic controllers (PLC), remote terminal units (RTU), control servers, historian servers, engineering workstations (EWS), I/O devices and intelligent electronic devices (IED).

PLC

PLCs are the actual controllers of the system. They are non other than industrial computers, built to withstand the harsh environmental conditions that can be found in industrial workplaces, such as extreme temperatures, electrical/mechanical noise, low power quality, and so on.

Similarly to regular PCs, they are composed of a processing unit (CPU), a memory unit, an input/output interface and a communications module to setup a network configuration. Their main characteristics include: being equipped with a real time operating system in order to perform real time computation and decision making for time critical scenarios (such as industrial), ability to operate continuously without much required maintenance for long periods of time, the possibility to be easily reprogrammed without having to rewire any electrical/mechanical components since the control operation is handled via software.

In order to work, a PLC must first be loaded with a program containing the necessary control procedure to be executed cyclically throughout its operation. This

program is responsible for interpreting sensor inputs coming from the I/O interface and sending the calculated output signals to the necessary output modules.

The five most popular and widely supported programming languages for writing PLC code, as defined by the IEC 61131-3 standard [3], are: ladder logic, function block diagram, structured text, instruction list, sequential function chart.

Since PLCs possess a communications module, it is possible to link them together in a network configuration, allowing them to send collected data to other devices for analysis and visualization, and also receive new task details to commit there effort to.

RTU

Like PLCs, also RTUs are responsible for monitoring and controlling field devices within an automated industrial process.

RTUs however, are more sophisticated devices and are mainly used to supervise and control more numerous and complex tasks, possibly distributed over a large geographical area. Because of these characteristics, they are usually linked to a Supervisory Control And Data Acquisition (SCADA) system (which is discussed later in Section2.1.4).

Normally PLCs, with respect to RTUs, are more suitable for local centralized control systems rather than distributed ones, as they have been designed to handle high speed inputs and outputs, which is important for those critical processes where time is of the essence. RTUs, on the other hand have been designed for monitoring and controlling distributed systems, therefore they must be capable of communicating with other control systems, possibly via wireless since they may be spread out across large areas.

Because of their lower complexity, PLCs seem to be the better option when it comes to automating simple centralized processes, such as assembly lines or packaging equipment, as they are cheaper, more efficient, require less maintenance, have lower energy consumption, and so forth. Vice versa, RTUs are more convenient when the required functionalities include: possibility to operate remotely, general monitoring and data exchange capabilities, ability to execute more complex control algorithms, and so on.

HMI

HMI, as mentioned before, stands for human machine interface.

As the name suggests, a HMI is a device responsible for providing a user interface to human personnel for displaying information about the status of a monitored process and devices associated with it, along with the possibility to perform some authorized basic interactions with the system, such as modifying operating speed parameters.

The introduction of these components in industrial environments helped to ease the burden on human operators, who can now simply keep track of production progress and machinery status without having to walk the plant floor to manually assess everything. This results in increased productivity and reduced likely hood of human error, since all the interesting information is automatically gathered and neatly displayed in the form of graphs and charts on the HMI screens.

EWS

An engineering workstation, abbreviated as EWS, is a computer connected to an OT network, that allows authorized personnel to interact with the system, somewhat similarly to what a HMI does.

The difference between the two lies in the fact that a HMI, is often restricted by access control and therefore provides only limited functionalities to the user, such as displaying the status of a monitored process or allowing some basic parameter reconfiguration. While an EWS, on the other hand, is more than a simple graphical interface, as it typically possesses: a wider view of the system as a whole, sensitive design documentation regarding the plant configuration and operation, necessary tools that allow to configure and update the control equipment (PLCs, RTUs, servers, and the like) responsible for driving the overall plant operation.

All of the previously stated resources contribute to make an EWS a higher valued asset within an OT system with respect to a HMI. Therefore they require more attention to cybersecurity even though the typical attack vectors used to compromise them are not much different from the ones of a HMI, since the underlying purpose they serve is the same.

Historian server

In order to keep track of operational data and compute process statistics, it is necessary for OT infrastructures to include specialized servers within their networks.

These server are normally called historians, as they serve the purpose of storing live data received from other devices, typically sensory data gathered by control devices (e.g. temperature values, actuator speeds, energy consumption, and so on), and maintaining the history of the plant's operation. This particularly increases the overall efficiency of the processes controlled by the system, as it is possible to analyse the data collected over a large period of time and use it to produce useful statistics that could reveal flaws and provide hints about what needs to be improved to reduce cost and maximise productivity.

The historian can also be used to keep a log of system events such as user interactions with the system and system alarms in order to provide security auditing features which are fundamental for detecting system anomalies and potential threats.

Control server

A control server is a network component responsible for providing the supervisory and control software necessary for monitoring and coordinating the lower level control devices of an OT infrastructure. It is the glue of the system that allows all other previously stated components to communicate seamlessly by providing the following functionalities:

- **Control and automation**: it manages the automation and control of industrial processes by coordinating and dispatching tasks to the lower level control stations which are handled by PLCs/RTUs.

- **Data collection and storage**: it collects system data from PLCs/RTUs regarding field data obtained from I/O devices, such as temperature and pressure values coming from sensors and actuators, and handles its storage at the historian. This feature is crucial for conducting subsequent analysis on the system's past operation, allowing to detect possible anomalies that could have occurred and supporting business planning and decision making.

- **Real time monitoring, alarms and notifications handling**: this functionality is crucial for maintaining safety at all times. It allows to pickup on possible anomalies and potential threats to the system in real time, notify personnel about the issues that have been encountered and promptly take decisions aimed at returning the system to a safe status before resuming operation.

- **System status visualization and reporting**: provides a general overview of the system's operation by gathering and visually displaying useful information about the running physical processes (charts, diagrams, and so on) and automatically generating detailed reports about the OT infrastructure's status.

- **Security and access control**: it can also be configured to manage security parameters to protect the system against possible external or internal threats. Some of the implemented features could be: setting user privileges, regulating access control, handling data traffic encryption, maintaining user access logs, providing authentication mechanisms, and so forth.

- **Integration with higher level systems**: it is possible for the control server to communicate also with higher level systems, such as enterprise resource planning (ERP) or manufacturing execution system (MES). These systems will be discussed further on in more detail, however it is important to note that they manage production information at higher and more business oriented level, so they are to be considered part of the IT world.

Depending on the chosen architecture and the geographical scope of the system, there could be several control servers working together and grouped in a hierarchy.

IED

The acronym IED stands for Intelligent Electronic Device. This term is used to represent I/O devices that directly integrate control, supervisory, and communication functionalities without necessarily having to rely on a PLC/RTU to handle them, as they are capable of directly interacting with the control server.

The introduction of these smart components can reduce cost and boost efficiency in an OT system, as more computational power is embedded into edge devices, thus promoting independence among system components. This an example of how the IoT paradigm influenced the world OT and industrial automation.

2.1.4 Architecture of an OT network

Depending on the requirements and scope of the system looking to integrate OT into its functioning, there are several available options to choose from in order to select the most suitable architecture for the particular case at hand.

The main factors that help design and discriminate among the different possibilities are:

- **Safety**: relates to how the system detects and reacts to the occurrence of hazardous conditions; most times human oversight proves to be essential for maintaining safety during operation.
- **Control timing requirements**: groups the different time related requirements for process automation, such as high speed, consistency, synchronization, regularity; all of which are of fundamental importance for the safe execution of the assigned process.
- **Geographic distribution**: defines the physical scope of the system, which can range from a small local process to a distributed, large system like an electric power grid.
- **Hierarchy**: describes the level of aggregation of a system consisting of multiple locations.
- **Control complexity**: specifies the level of complexity that must be tackled by the system in order to meet the prescribed goals.
- **Availability**: defines the up-time requirements that have to be met and therefore is influenced by the level of redundancy within the system.

- **Impact of failures**: defines how the system should cope with failures depending on the impact degree caused by the anomaly.

Based on these factors, the architectures that are going to be presented in the following subsections are: SCADA (Supervisory Control And Data Acquisition), DCS (Distributed Control Systems), IIoT (Industrial Intenet of Things). The main focus will be on SCADA since it is the architecture that has been considered in the case study.

SCADA

SCADA stands for Supervisory Control And Data Acquisition.

The goal of SCADA is to collect and display operational data coming from distributed control stations managed by PLCs/RTUs responsible for locally controlling the physical processes that make up an OT system, so that an operator can easily monitor and control the operation of the entire system from a centralized location.

Normally SCADA is used to manage geographically dispersed systems consisting of several field stations covering wide areas, such as oil and natural gas pipelines, water distribution, railways and other types of public transportation, and so on. For these mentioned infrastructures it is of the utmost importance to have a centralized control location from which it is possible to get a global overview of the system, since failures in one field station could compromise the functioning of all the others.

As can be seen in the general SCADA architecture depicted in Figure 2.2, the control centre is configured as a local area network (LAN) consisting of a control server, a HMI, a EWS a data historian and the gateway communication routers necessary to communicate with the field stations.

As mentioned in Section 2.1.3, the control server is the brain of the SCADA system, as it is responsible for all of the supervisory tasks of the system, such as collecting and logging data from field sites, triggering control actions based on occurring events and parameter ranges in field sites, handling alarm procedures, displaying information to HMIs and providing support for analysis and reporting.

Because of the geographic nature of these systems, long range communication mediums must be supported to allow the transmission and reception of data between the field stations and the control centres, and also to support remote access to field sites in order to perform remote diagnostics and repair. Also because of the criticality of the processes that these infrastructures typically control, it is of fundamental importance to have some form of redundancy to ensure the availability of the system at all times.

In Figure 2.2, it is possible to see that four different types of long range communication are supported, such as wide area network (WAN), satellite communication, radio/cellular, and switched telephone/power line.

The field sites consist of sensors and actuators monitored and controlled by PLCs or RTUs which communicate with the control server through their communications module.

Typically field sites also offer remote access capabilities to operators in order to allow them to run system diagnostics and repair operations when needed.

It also possible for a field site to have IEDs which are capable of directly communicating with the control server and have embedded control functionalities, therefore the presence of a RTU/PLC to locally control the process is not stricly required in these cases.

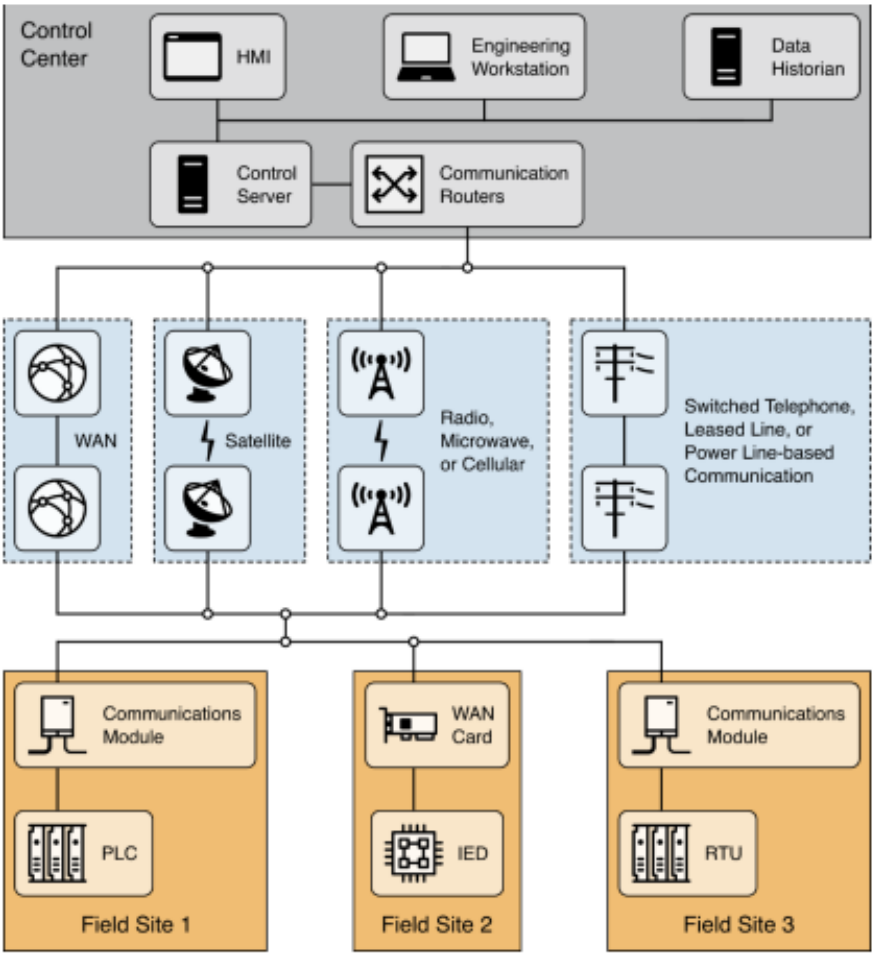

Figure 2.2: Standard SCADA architecture taken from the NIST guide for OT [2]

Depending on its size, the system could employ a number of control servers organized in a hierarchy, in order to ease the load on the central hub and increase

management efficiency by allowing the different nodes to work as independently as possible, as shown in Figure 2.3.

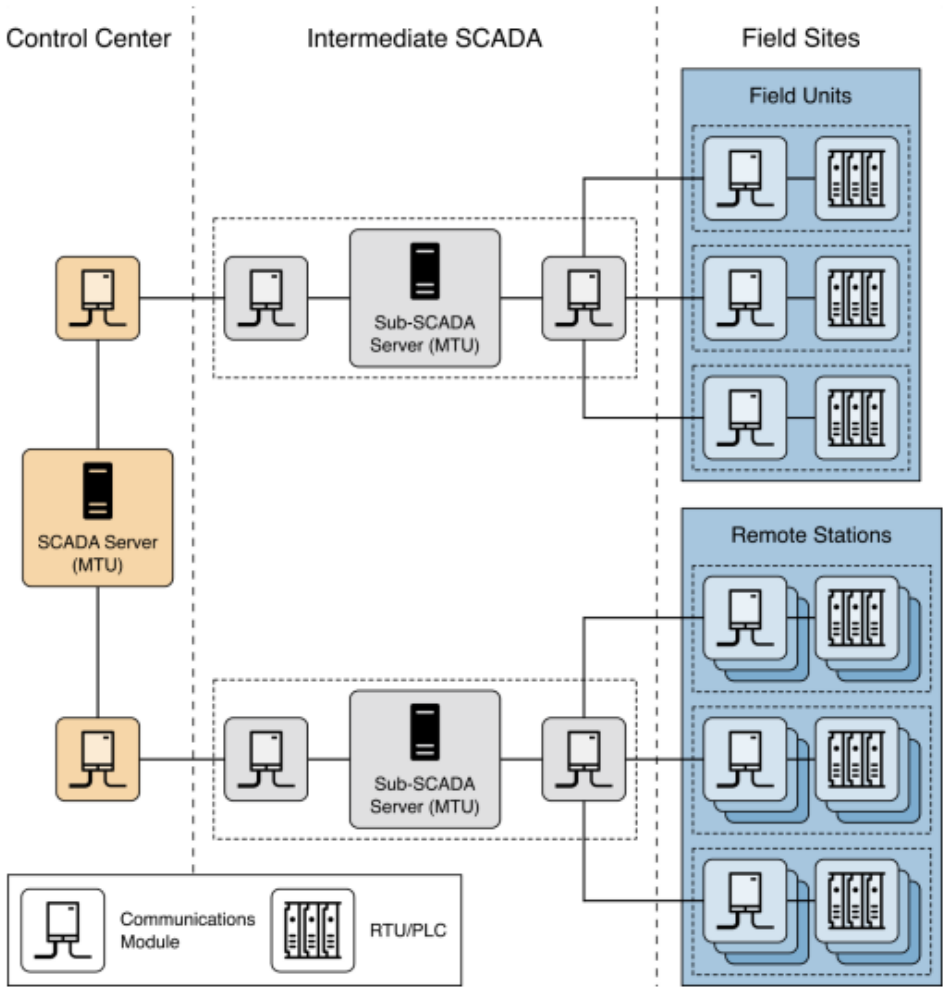

Figure 2.3: SCADA with multiple control servers taken from the NIST guide for OT [2]

DCS

Distributed control systems, abbreviated by the acronym DCS, is a control architecture typically used for controlling a production system that is localized within the same geographical region, as opposed to SCADA. Some examples of where a DCS could be found would be: oil refineries, automotive production, pharmaceutical processing, and so on.

As can be seen in Figure 2.4, a DCS consists of a supervisory level composed of control servers, EWSs, data historians, and a field level composed of controllers (such as PLCs), sensors, actuators, and HMIs.

The supervisory level is responsible for supervising the various distributed field station by selecting set points and collecting data from the controllers. It hosts a historian for storing the collected field data, a control server which communicates directly with the field controllers and monitors and controls their behaviour, an EWS for allowing operators to interact and reconfigure control parameters of the control server, and a general console for displaying information to the plant operators.

The field level is split into several field station, each responsible for controlling a specific process. Each station is locally managed by a controller, which could be a generic PLC but could also be a more capable controller, specifically built and tuned for handling the task it has been assigned to (in Figure 2.4 there are three stations, one is controlled by a PLC, while the other two are controlled by specific controllers). These controllers operate in either feedback or feed-forward control loops, by receiving input signals from sensors and computing output signals to send to the actuators regulating the physical processes.

It is possible for field station devices, such as HMIs, sensors, actuators, remote access computers, to be connected to the corresponding controller through a field bus rather than having to rely on a point to point connection. This type of bus connection allows for more efficient communication, as it avoids routing back every control signal through the controller, however it requires support for specific bus protocols which is discussed later in Section 2.1.5 .

In modern systems, DCS are typically interfaced also with the enterprise network in order to provide a view of the lower level production status to the higher level business operation infrastructure. In Figure 2.4 this corresponds to the top layer which hosts various IT devices such as application servers, printers, workstations, and connects to the management information system (MIS) and enterprise resource planning (ERP) systems, which provide support for business decision making through automation and real time data analysis.

IIoT

Since nowadays IT and OT are converging ever more, there is no reason not to benefit from solutions that have been developed for their counterpart.

The acronym IIoT stands for industrial internet of things, and it refers to the attempt to deploy the classical internet of things paradigm aimed at IT scenarios to OT environments. This consists in introducing smart devices that can harness internet connectivity in order to enhance industrial processes.

The standard architecture for IIoT is the one depicted in Figure 2.5. It is based

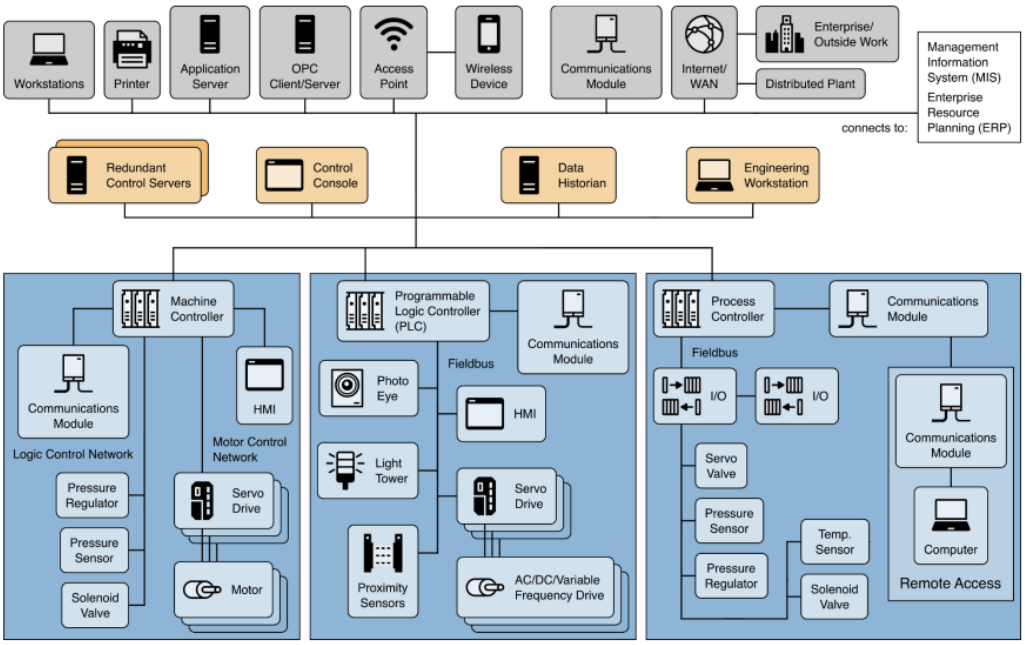

Figure 2.4: Standard DCS architecture example taken from the NIST guide for OT [2]

on a three tier model consisting of an edge tier, a platform tier, and an enterprise tier. This subdivision into tiers allows to separate the processing of data into different levels, from the lower level data generated from the edge devices (sensors, actuators, and so forth), to the higher level data concerning enterprise specific production planning and business decisions. This provides greater modularity and lower coupling within the system, which results in increased efficiency and scalability, therefore allowing to move towards a distributed system that can exploit the cloud to provide enhanced data analytics capabilities.

The enterprise tier provides an interface for end users and supports domain specific applications for decision making. It is responsible for receiving data and sending control commands to the underlying tiers.

The platform tier's main job is to act as an intermediate step between the enterprise tier and the edge tier. This is achieved by providing non domain specific services for processing data coming from other tiers, forwarding control commands to the edge tier, and managing devices and assets.

The connectivity between the enterprise tier and the platform tier is managed by the service network, which could take the form of a virtual private network, whose goal is to allow the authorized domain applications to access the functionalities provided by the services hosted by the platform tier.

Last but not least is the edge tier, which is responsible for collecting data from the single devices (sensors, actuators, and the like) which it can communicate with through the proximity network. This data can be either processed at the edge level itself or forwarded to the superior tiers.

The advantage of having an edge tier is that rather than only collecting and forwarding data for future processing, it could be used to directly manage and control the underlying edge devices, therefore effectively decentralizing the computing infrastructure and consequently improving overall speed and efficiency.

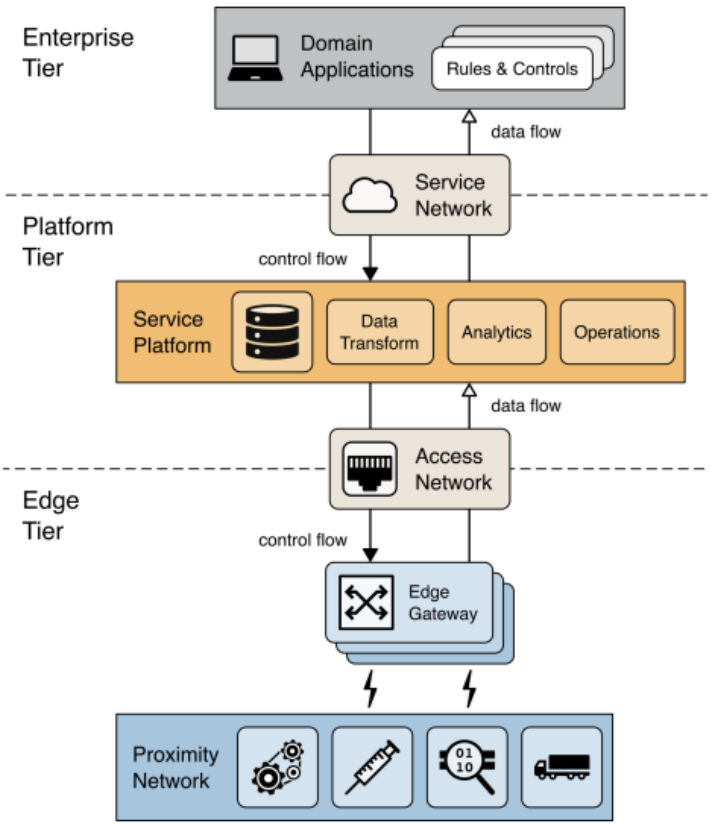

Figure 2.5: Standard IIoT architecture model taken from the NIST guide for OT [2]

2.1.5 Commonly used protocols

When it comes to networking, OT and IT have developed different standards to rely upon over the course of their history, since initially there was never no intention of combining them together. With regards to this, the protocols that have been deployed in OT networks are specifically tailored to properly handle the needs that

OT scenarios typically entail. Therefore they substantially differ from the common protocols used in IT infrastructures.

Some of the most commonly used protocols in the context of industrial automation are: OPC-UA, MODBUS, PROFIBUS/PROFINET.

OPC-UA

OPC-UA stands for Open Platform Communication Unified Architecture.

It consists in an open source, platform independent standard developed by the OPC foundation and used for communication within industrial networks.

A complete dissertation on this protocol is given by [4]. However, for the sake of brevity and simplicity, it can be summarized as a service oriented, application layer protocol commonly deployed in OT environments in order to support horizontal communication between PLCs, but also vertical between PLCs and control servers or data aggregation servers.

OPC-UA supports both a client server communication model, based on either TCP/IP or HTTPS for the transportation of data, and a publisher subscriber communication model, compatible with protocols such as MQTT.

A common place to find this protocol used in client server mode would be within SCADA architectures. In these scenarios, the PLCs act as OPC servers allowing the control servers, which act as clients, to request field data or to configure set points for the controlled processes.

Since its release in 2006, OPC-UA has undergone several updates, some of which include the improvement of security features by adding support for data authentication, confidentiality, and integrity.

MODBUS

As described in [5], MODBUS is an old protocol developed by Modicon in the 1970s for supporting point to point communcation between their PLCs. Since then, due to the popularity its simplicity and popularity, MODBUS has been openly released and has become a standard for PLC communication in the industrial scene.

As for OPC-UA, it is mainly used for communication between PLCs and superviosory computers. It is an application layer protocol that works in a client server fashion, however it presents different specifications based on the underlying transport protocol in use. These specification are the following: Modbus RTU, which uses a simple binary representation for data, including basic integrity features, and operates over serial bus connections in a master slave mode; Modbus TCP/IP, which reaps the benefits of TCP/IP protocol and is typically used when speed and long range communication are of the essence, however as a drawback adds more complexity and overhead to the process.

From a security stand point, seen as it is a much older protocol compared to the previously described OPC-UA, MODBUS does not protect against unauthorized commands.

PROFIBUS/PROFINET

As described in [6], PROFIBUS consists of a serial bus protocol. In OT systems, it is mainly used for the field level communication between PLC controllers and field devices such as sensors and actuators, all connected in bus formation.

According to the system requirements, it is possible to choose between two different specifications of PROFIBUS, being: Profibus DP and Profibus PA. The first stands for decentralized peripherals, and is to be used when speed is of the essence. While the second stands for process automation, and is mainly concerned with maintaining safety of operation in hazardous environments at the cost of transmission speed.

PROFINET, on the other hand, is an ethernet based communication protocol that leverages the advantages of ethernet and TCP/IP protocols, allowing for easy integration of both factory and process automation, including higher-speed applications and more complex setups.

2.2 Security in OT

Cybersecurity over the past decades had never been given the importance it truly deserved among the fields of computer sciences. However, nowadays things are changing and the importance of cybersecurity has grown to everyone's attention.

Especially over the last five years, due to unpleasant events, technology and internet connectivity have become crucial for the regular unfolding of everyday life, impacting all fields, from social to health and work. All of these circumstances served to highlight the shortcomings of the current state of cybersecurity in a variety of different fields.

One of the most impacted fields, lacking the most in terms of cybersecurity, turns out to be non other than OT. This should be extremely worrying given the role it plays in the management of critical processes. If a malicious attacker were to somehow gain control of one of these systems, the damage he could cause could be incalculable.

The main reason behind these security related issues in OT is the fact that for years these systems have been kept isolated from the internet, relying on the so called *air gap* as the primary (and in some cases the only) form of security put in place to defend themselves from threats. This form of security does indeed narrow down the attack surface by eliminating the possibility of external attacks originating from outside the local network infrastructure. However it still is not enough to

guarantee the necessary security standards, as it does not provide protection against insider threats, which in the last couple of years have affected more than 60% of companies globally.

Now that also these type of systems are being connected to the internet, in order to provide support for remote operation, data analytics and remote diagnostics, the main defence mechanism they relied upon, being the *air gap*, is gone. Resulting in the exposure of all the internal security issues to the public internet.

This is a perfect example of how one of the core principles of cybersecurity, being defense in depth, has not been applied properly.

The defense in depth principle specifies how security must be structured on multiple layers so that if one line of defense were to fail there would still be other ones to fallback on.

2.2.1 Security priorities in OT versus IT systems

Even though OT and IT are converging, they still present different characteristics, given the diversity of their nature.

The first is built specifically for the management and control of physical processes, while the second focuses more on providing data oriented functionalities such as communication and analysis.

Since OT and IT priorities differ, also cybersecurity strategies for the two must differ, as they should be tailored to the corresponding environmental properties.

As stated in [2] the main properties around which security must be built for OT scenarios are:

- **Timeliness and performance requirements**: typically OT systems cannot handle time delay and jitter, as most of the processes they control are time critical and therefore require real-time responsiveness in order to enforce deterministic behaviour. From a cybersecurity standpoint this requirement might rule out the application of solutions that could introduce some sort of overhead, such as traffic encryption.

- **Availability requirements**: availability is another important characteristic of OT systems, as they are responsible for running important processes that cannot afford downtime and must therefore be available at all times. This normally entails different forms of redundancy to maintain operation in case of components becoming unavailable. Due to this requirement, cybersecurity solutions must be exhaustively tested before deployment, as it is not possible to interrupt the system's operation every time a security update needs to take place, or when an accident happens, on the contrary to what happens in IT. It is although still possible to perform security updates during the lifetime of the system, but they must happen during previously planned periods when it

is alright to stop the system's operation, therefore they cannot be scheduled frequently.

- **Risk management requirements**: arguably the most important concern of OT is safety as opposed to IT which is more concerned with maintaining confidentiality and integrity of data. Cybersecurity solutions must therefore focus on maintaining safety at all times by implementing measures such as proper access control, least privilege, adequate network segmentation, safety shutdown mechanisms, alarm notification system.
- **Communications**: even though nowadays OT systems are becoming more compatible with common IT protocols such as ethernet/IP, they still rely on different ad-hoc protocols some of which could even be proprietary. These type of protocols did not get the same focus on security improvement that their IT counterparts did, therefore they could present several potentially exploitable vulnerabilities.
- **Managed support**: most modern day IT systems make use of third party software to provide different functionalities (e.g. online payment processing). In OT however, many systems are supported by a single vendor, not allowing for any third party solutions to be used without breaking prior licenses and service agreements.
- **Component lifetime**: in contrast to IT technology, in which components typically last for short periods of time (about 4-5 years) due to the rapid evolution characterizing this branch of technology, components of an OT system are built to last for longer time spans (up to 15 years) allowing for prolonged operation before having to block any controlled processes for hardware upgrades. This point also plays a part in security, as it requires a lot of care and attention in selecting good hardware equipment and ensuring they meet the necessary security standards before including them in the operation cycle.
- **Component location**: as opposed to IT components, it is common to find OT components hosted in remote facilities, which are difficult to reach physically by human personnel due to the nature of the processes they control. Therefore security measures must also address this characteristic by providing alternative ways to react or reach these devices in a secure way in case something goes wrong.

2.2.2 MITRE ATT&CK framework for ICS threats

In addition to the previously highlighted properties, which greatly restrict the ability of OT systems to adopt the same security measures of their IT counterparts,

the main concerns with the current state of security in this field of application stem from the low level of understanding shown by many network administrators when configuring these systems. It is not uncommon in fact to find real world scenarios for which no sort of access control or segmentation has been put in place, allowing anyone with basic access to any machine (e.g. wifi printer) in the network, to reach critical control device without proper authorization. In even worse cases, it has been possible to find ip addresses of PLCs publicly exposed to the internet and accessible through their hosted web portal service by simply reusing the vendor's provided default credentials. It is therefore needless to say that the best way to start improving security in OT environments, would be to educate the companies on the possible risks that could be encountered, and the best practices to adopt in order to mitigate them. Because the chances are that most of these systems would not even require a big financial investment to achieve an acceptable level of security.

In regards to the common attacks affecting OT technology, the MITRE ATT&CK framework for ICS, described in [7], has put a lot of effort into providing a detailed and organized representation of the common tactics and techniques employed by these attacks. Tactics represent the goals that each attack action tries to accomplish within the targeted system. Techniques on the other hand correspond to the actual steps an attacker could use in order to achieve a tactical goal.

Popular examples of tactics could be the following: code execution, privilege escalation, lateral movement, data collection, impair process control, and many others. While techniques could be: exploitation of remote services, network sniffing, unauthorized command message, adversary in the middle, denial of service, and many more. From these examples, it is pretty easy to notice how these two categories correlate between each other. As the code execution tactic code be achieved by the exploitation of remote services technique, while data collection could leverage network sniffing, impair process control could be linked to denial of service, and so on so forth.

Some of these techniques provided by the MITRE ATT&CK framework have been used to model the types of attacks later described in Chapter 4.

2.2.3 Security assessment process

The process of security validation is the step by step procedure that is carried out to frame the overall security status of a system, in order to notify the people of interest about the discovered issues and potential consequences that could happen if not promptly handled.

This procedure is extremely important when it comes to securing any type of system. If not performed accurately it could lead to the exposure of the system to all sorts of threats, some of which could be critical, therefore major care must be

taken when executing this process.

The main steps that constitute the validation process are:

- **Reconnaissance**: regards the process of gathering all of the information about the system (hosts, subnets, protocols, services, vulnerabilities, topology, asset values, and other useful data) that is needed to conduct the following steps. This could be achieved for example through passive/active scanning techniques by making use of popular open source scanning tools such as Nmap or Nessus. This is the most important part of the process, as all of the following steps depend on the information gathered during the this phase, therefore if the produced data weren't accurate enough many attacks could go undetected, causing a cascading effect in the rest of the procedure.

- **Attack modelling and planning**: this point is about constructing a model for representing and enumerating the potential attacks that could be attempted by malicious actors to compromise the system by exploiting discovered vulnerabilities and network configuration.

- **Attack validation**. concerns selecting and executing the appropriate tasks to test the actual exploitability of the potential attack paths, modelled at the previous step, as some may be false positives.

- **Attack risk analysis**: involves the definition of metrics for expressing the risk associated with the discovered potential attacks in a measurable way. The overall risk should take into account metrics describing probabilities of threat occurrences, asset values, impact in case of failures.

- **Mitigation strategy support**: indicates how the system should be modified in order to reduce the risk parameters, calculated at the previous step, so that they comply with defined acceptable value ranges. The proposed modification strategies could be of different forms, such as software vulnerability patching, introduction of firewalls for providing access control and network segmentation, addition of servers for increasing redundancy and ensuring greater availability, termination of all non necessary open ports, and so on.

- **Reporting**: revolves around the production of a human readable report detailing all the information generated in the previous steps.

In order to automate this process, suitable models should be defined for structuring all of the required data so that it can be represented in a standard way, allowing it to be understood and processed automatically by the various modules responsible for managing the previously listed steps.

2.2.4 Famous attacks against OT

Over the last decade the number of attempts to attack OT systems has grown exponentially. The low security measures combined with the great potential impact of a threat occurrence greatly appeal to malicious attackers seeking anything from financial gain to physical disruption fueled by geopolitical motives.

Some attacks against OT systems stood out in particular, as they helped to raise awareness about the gaps and shortcomings in the current state of security for these systems, and how important it is to address these issues given the huge impact that some of these attacks had.

Following will be a brief overview of a two of the most famous attacks that shook the world of OT at its core.

Stuxnet

Stuxnet is widely regarded as one of the most famous examples of successful attacks against SCADA driven infrastructures. It has been the focus of numerous works such as [8], [9], [10], [11] to name a few.

As described by these works, the Stuxnet attack consisted of a worm crafted with the purpose of disrupting the standard functioning of centrifuges responsible for regulating the process of uranium enrichment in a nuclear power plant located in Iran.

The malware was discovered in 2010 and was subsequently classified as an advanced persistent threat, due to its capability of hiding and replicating itself to other connected devices.

The reason it became famous is because it was the first major attack against critical infrastructures, showcasing how cyber attacks could also affect physical systems and deal enormous amounts of damage.

The worm exploited four zero day vulnerabilities and made use of two stolen public key certificates in order to target Windows systems to spread through the network without being detected while meanwhile searching for vulnerable SCADA software (WinCC, Siemens PCS 7, STEP 7) which could be exploited to compromise Siemens S7 PLCs responsible for controlling the centrifuges.

The initial vector used to deliver the payload was likely a compromised USB flash drive, which demonstrated how air-gapped systems could be easily bypassed and how the lack of proper defensive measures could lead to catastrophic consequences.

Due to the high degree of complexity demonstrated by Stuxnet, and to the geopolitical context, its development has been attributed to the USA secret services, highlighting how cybersecurity could play an important role in matters of national interest.

Triton

Another attack that has been widely covered since its appearance is Triton [12].

What made Triton stand out is the fact that it was the first attack to target safety instrumented systems (SIS[1]).

The attack was carried out in the summer of 2017 and the intended victim was a petrochemical plant located in Saudi Arabia.

The perpetrators of the attack were able to gain initial access to the IT network of the company by leveraging easily detectable vulnerabilities. Thanks to this foothold, they were then able to pivot over to the OT network controlling the plant. From there they managed to reach an engineering workstation of the SIS and through social engineering they had an operator download a malicious executable file to the machine. Once executed, the malware would exploit a zero day vulnerability of Triconex controllers (PLCs) responsible for the SIS, allowing to upload arbitrary programs to the memory of these devices.

Fortunately, following an attempt to upload malicious programs to the controllers, for some unknown reason, the SIS system went into shutdown and the presence of the malware was discovered before any real damage could take place. If this had not happened the consequences of the attack could have been catastrophic for human and environmental safety.

The key takeaways from this event regard the security of SIS systems, and also the problems brought on by the convergence of IT and OT systems. In the discussed case in fact, the SIS system was reachable from the OT control network, which in turn was connected to the IT network accessible from the outside. This by no means should have been possible and it represents the main reason why SIS systems should run on physically separated networks. All it took was some common IT vulnerabilities and a likely misconfigured firewall to grant external attackers the power to cause incalculable damage.

2.2.5 Lessons learned

As discussed in Section 2.2.2 and demonstrated by the real world attacks discussed in Section 2.2.4, the most common problems affecting OT scenarios consist in a combination of misconfigurations made by network administrators and low security support for the vulnerable protocols and devices common to this field.

In order to help companies to identify and deal with the security issues exposed by their systems in a complete and deterministic manner, a standard tool for

[1]SIS are systems charged with the task of detecting and preventing the occurrence of malfunctions in the operation of regular OT control systems. They are crucial for ensuring safety in industrial environments and therefore they have low to zero tolerance for failures.

automating the security assessment steps described in Section 2.2.3 would be of great benefit. Such a solution would allow network administrators to reliably protect their systems in the hope that events such as the ones described previously would not happen again.

The following chapters will address the steps that this work took, in the scope of the company project mentioned in the introduction, to get a step closer to realising a fully automated solution for security assessment in OT infrastructures. And in doing so, it will also analyse relevant works available in the literature.

Chapter 3

An automated solution for security validation of OT infrastructures

Still to this day, their is a lot of human effort that goes into the cybersecurity assessment of networked infrastructures, as the complexity of the task makes it difficult to create fully automated solutions. This manual procedure has proven to be costly and time consuming, as it requires good technical expertise and never the less is still subject to errors due to its reliance on the human eye.

Over the years, several tools have been developed to automate certain steps of the security assessment process of a system, such as network vulnerability scanners, and exploit auxiliary modules. Most of them have been developed for IT scenarios but can now find themselves to be useful also in OT scenarios, because of the convergence that is happening between the two. Although the usage of such tools greatly improves the efficiency of the overall assessment process, at the moment there still are not any standard solutions capable of performing a full assessment whilst requiring minimum human expertise and manual intervention, especially in regards to OT scenarios. Hence the need for an all-in-one solution capable of performing a complete and automatic assessment of OT infrastructures from start to finish, resulting in the reduction of overall cost and effort that is required by the manual procedure.

3.1 Context of the project and scope of this work

The global context, within which the project collocates itself, is the creation of a standard solution for automating the process of security validation of OT

infrastructures, therefore addressing the problem highlighted in the introduction of this chapter.

In order to achieve this, the proposed tool must be capable of performing the main security assessment steps described in Section 2.2.3. All of which should happen in a continuous update fashion, allowing to adapt when changes take place.

From a physical stand point the proposed tool is a plug-and-play device that once connected to a network should be able to dynamically assess its security without the need for prior knowledge (black box approach). However, if more detailed results wish to be produced, it could be possible for administrators to provide the tool with some additional information about the network (grey box approach).

The general workflow of the project is given in Figure 3.1.

As can be seen in the figure, the proposed tool relies on five engines in order to carry out a complete security assessment process:

- **Reconnaissance engine**: is the engine responsible for scanning the network and producing a detailed description containing information such as device configurations, network configuration, network topology, exposed vulnerabilities, and access restrictions.

- **Formal verification engine**: is the engine responsible for computing possible attack paths and generating a validation task execution plan based on the information gathered by the reconnaissance engine.

- **Task execution engine**: is the engine responsible for running the validation tasks selected by the formal verification engine, in order to detect which discovered attack paths are actually exploitable and which are false positives. The output of this step is the set of the refined attack paths obtained by eliminating the false positives.

- **Risk assessment engine**: this engine is responsible for ranking the attack paths, obtained by the combined effort of the previous engines, based on risk analysis metrics and formulas.

- **Reporting engine**: is the engine responsible for summarizing all the useful information generated during the whole process and use it to automatically generate a complete report about the overall status of security within the system under evaluation.

With respect to this architecture, this work focuses on the design and development of a prototype for the formal verification engine.

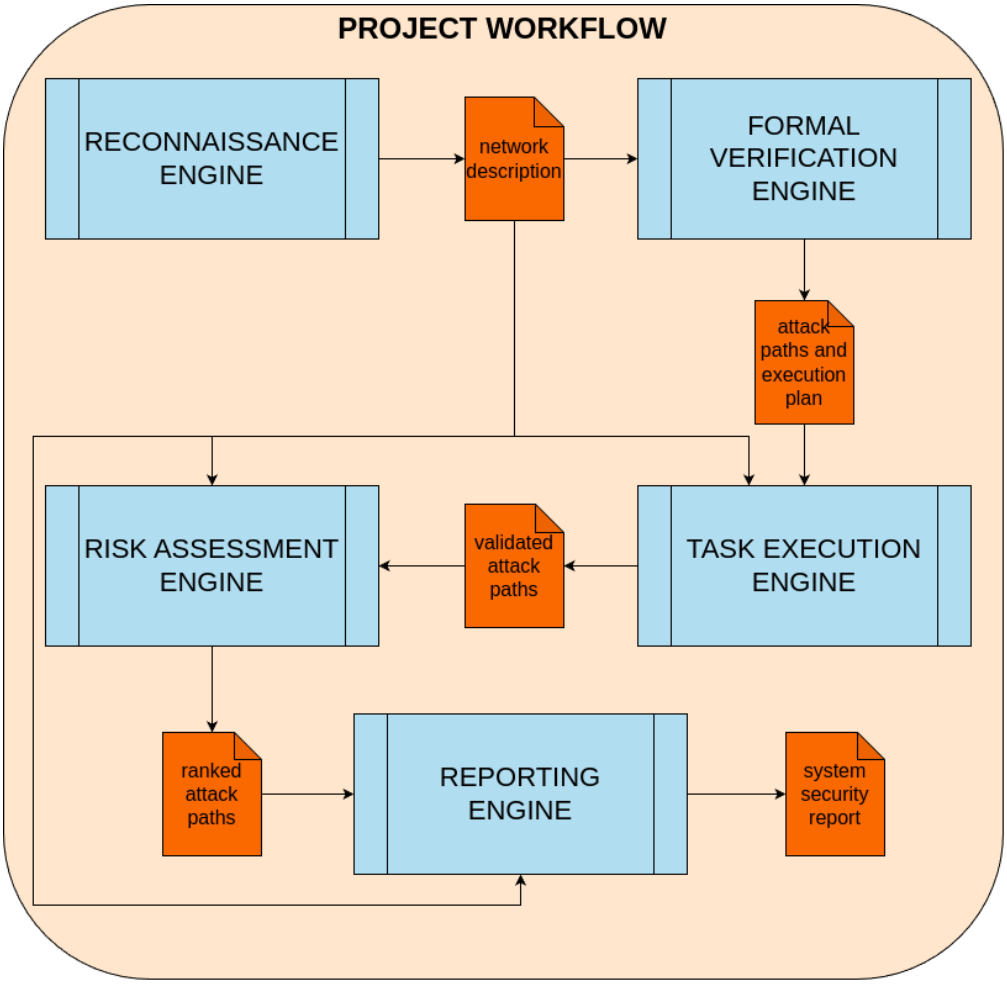

Figure 3.1: Workflow of the full company project

3.2 Need for attack graphs

Typically, in the process of security assessment, the modelling of potential attacks that could compromise the security of a networked system is handled by means of automatic vulnerability scanners, which provide detailed reports containing information regarding the presence of vulnerabilities along with possible mitigation strategies.

These tools however, tend to be mostly host centered, as they provide host specific information in isolation from the global system, therefore they do not capture the relationships that exist between different vulnerabilities, which is a very important point when it comes to attack modelling. In fact, on many occasions it is not possible to simply patch every detected vulnerability on each host due to cost

and operational constraints, therefore it is essential to have a clear understanding of how the identified issues relate to each other when choosing the mitigation strategies to apply. As a matter of fact, most successful attacks against critical assets come from chaining together multiple elementary exploits across multiple hosts in what is called a multi-stage, multi-host attack.

In order to tackle this problem, the concept of attack graph has been introduced as a possible solution for accurately representing in a standardized way the inter-dependencies that exist between host vulnerabilities and other network information (topology, security policy, and the like).

Through attack graphs it is possible to visually represent, in a concise and exhaustive manner, all of the paths an attacker could take to compromise a target system. And because of all the research that has been done in regards to incorporating automation in this field, the use of attack graphs seemed to be the best option for the job.

3.2.1 History of attack graphs

Since its introduction in 1998, the concept of the attack graph has garnered significant attention from researchers, leading to considerable research efforts in developing both the theory and practical applications [13]. As a matter of fact, the literature presents numerous studies and surveys on various attack graph generation tools.

In [14], the authors reviewed and compared several academic and commercial tools for attack graph generation and visualization.

In [15, 16, 17, 18], researchers provided a comprehensive overview of attack graph generation and analysis methodologies by summarizing different approaches to attack modelling, including attack graphs and attack trees.

In their manuscript [19], Hong *et al.* analyzed the state of graphical security models across four phases: generation, representation, evaluation, and modification.

More recently, in this research work [20], the authors examined several unknown vulnerability risk assessments using directed graph models and categorized their security metrics.

In the meantime, in [21] the authors conducted empirical research on over 180 attack graphs and attack trees, analyzing their visual syntax for representing cyber attacks.

Over the years, different types of attack graphs have been proposed, the most common being: *attack trees*, *state graphs*, *exploit dependency graphs*, and *logical attack graphs*.

Attack trees

Attack trees, abbreviated with AT, have been introduced in 1999 by Schneier in [22].

As can be seen in Figure 3.2, attack trees consist of a set of nodes linked in a tree like configuration. The nodes of the tree represent the attacker's actions, with the root node being the main goal of the attack, and the others being the steps the attacker could take in order to reach the root node.

Even though attack trees do not model all of the possible states of a system, their size still relies upon the number of possible events that could happen within the system, making them a poorly scalable option.

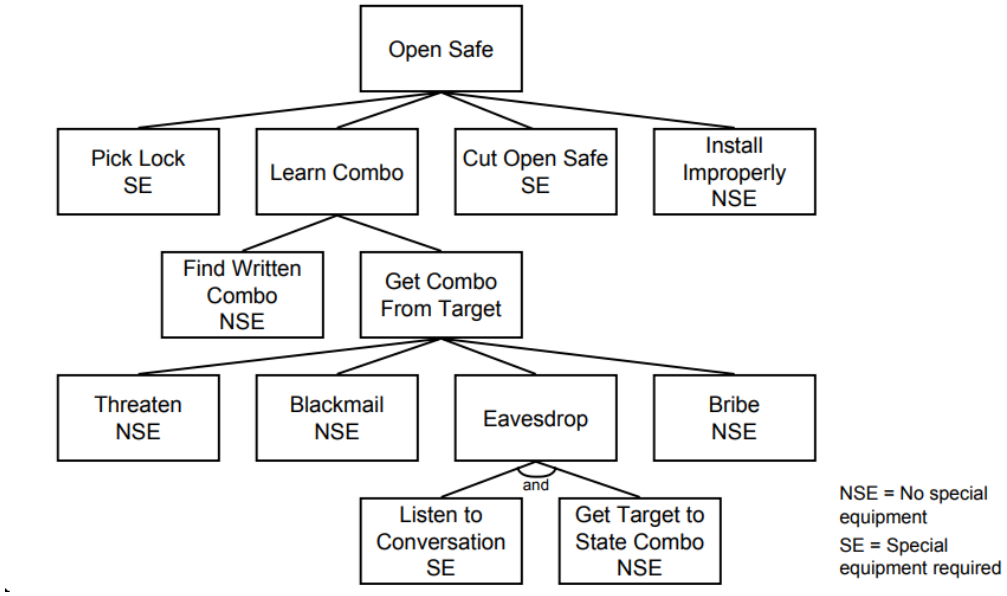

Figure 3.2: Example of attack tree taken from [22]

State graphs

As implied by the "state" term, a state graph models all of the possible states of a networked system, represented as nodes in the graph, along with the all of the possible attacker actions that trigger state transitions within the system, represented as edges in the graph, as can be seen in Figure 3.3.

Based on this concept, Sheyner presented the Attack Graph Toolkit [23] in 2002, an attack graph generation tool based on the NuSMV symbolic model checker. However, the main issue with this solution consisted in poor scalability caused by the state space explosion phenomenon, which characterizes state graphs, as addressed by [24].

Since then, several studies have tried to tackle this problem in order to make state graphs a viable option for modelling real world attack scenarios.

In [25], an assumption was made stating that preconditions of an attack could not be invalidated by the execution of another attack. This reduced the complexity required to generate the graphs from exponential to polynomial.

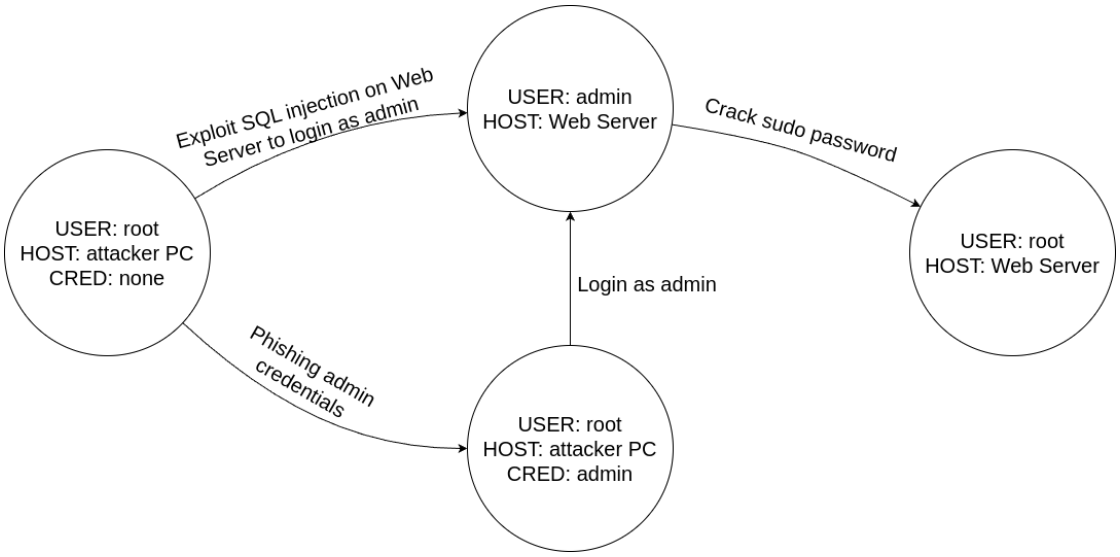

Figure 3.3: Example of state graph

Exploit dependency graphs

Exploit dependency graphs, abbreviated as EDG, were introduced to model attack scenarios as a sequence of exploits.

As showcased by [26], in this type of structure each node represents either an exploit that an attacker can execute, or an exploit precondition, and each edge expresses the dependency that exists among the exploit nodes. In other words, each exploit node depends on its parent nodes. So in order to successfully execute an exploit, it is necessary that at least one path of parent exploit nodes leading to it has been correctly executed.

In Figure 3.4, it is possible to notice that the nodes of the graph representing the actual exploits themselves, are enclosed in circles, e.g. sshd_bof(0,1), corresponding to a buffer overflow exposed by ssh which results in the attacker gaining user privileges on machine 1. While the open nodes are the preconditions required by the exploits, e.g. sshd(0,1), corresponding to the presence of an accessible ssh channel from machine 1 to machine 2.

The size of attack graphs based on EDG is quadratic in the number of exploits,

however the complexity required to enumerate all possible exploit combinations is exponential.

In 2005 [27], presented the TVA tool, which relies on EDGs to produce attack graphs. In order to effectively reduce the complexity and size of the produced graphs, the TVA tool generates all the attack paths starting from a predefined attack goal and working its way backwards. In doing so, only the exploits that are required to reach the prescribed attack goal are considered during the computation, as opposed to the forward dependency graph generation which blindly combines the exploits.

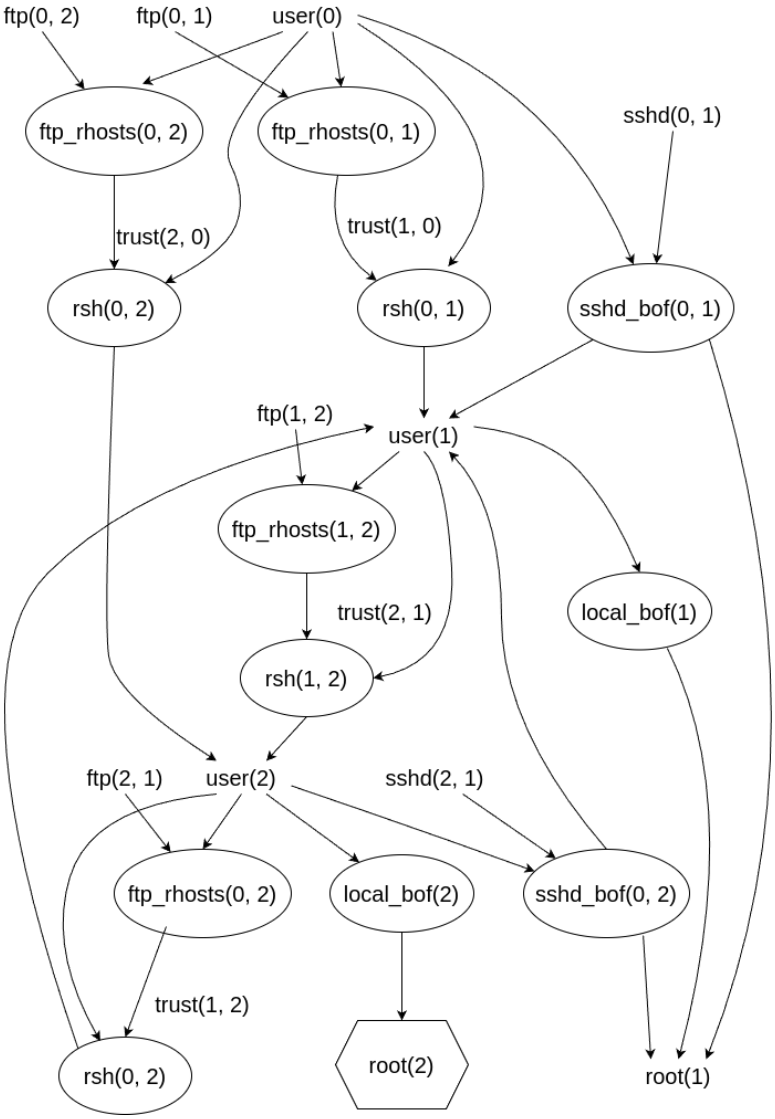

Figure 3.4: Example of exploit dependency graph

Logical attack graphs

Logical attack graphs, LAG in short, were introduced in 2006 by [28] as a means to represent the logical dependencies among attack goals and configurations.

In order to achieve this, LAGs consist of three types of node: primitive facts, derived facts, and rules.

Primitive facts are the nodes responsible for describing network assets and their configuration, so therefore include information about hosts, protocols, services, exposed vulnerabilities, topology, and the like.

Derived facts describe the information obtained via the application of a rule, and can therefore represent attack goals and intermediate steps needed to reach them.

As for rule nodes, they are the link between the different fact nodes, and represent the reasoning logic that drives the construction of the graph. They represent the actions that, given a certain input, described by a set of primitive and/or derived facts, produce a corresponding output, described as a single derived fact.

Attacker actions and exploits are modelled by means of rule nodes. Through these type of nodes it is possible to construct attack graphs that are polynomial in size with respect to the network being analysed.

An example graph is given later on in Figure 3.7.

3.3 MulVAL rule based inference engine

The authors of [29] compare all the most popular solutions for attack graph generation based on the different attack graph types described in Section 3.2.1, and also some others, and report them in Table 3.1. Among the available options, MulVAL is the one that has been selected for this work, as it has the benefits of being open source, easily extendable, provides good enough scalability, and has been referenced by many papers making it one of the most popular options according to the research community.

MulVAL is an open source multi-host multi-stage vulnerability analysis framework, presented by Ou *et al.* in [30] with the goal of modelling the interaction between network configurations and exposed vulnerabilities in order to produce a detailed LAG displaying the main security concerns that system administrators should worry about. At its core it consists of a rule inference engine, that runs in a Prolog environment as shown in Figure 3.5, which takes as input a set of primitive facts describing the target network and computes a set of derived facts through the application of a set of interaction rules. In order to perform this computation, the MulVAL framework leverages Datalog as modelling language for defining the elementary components of a LAG, being primitive facts, derived facts and rules.

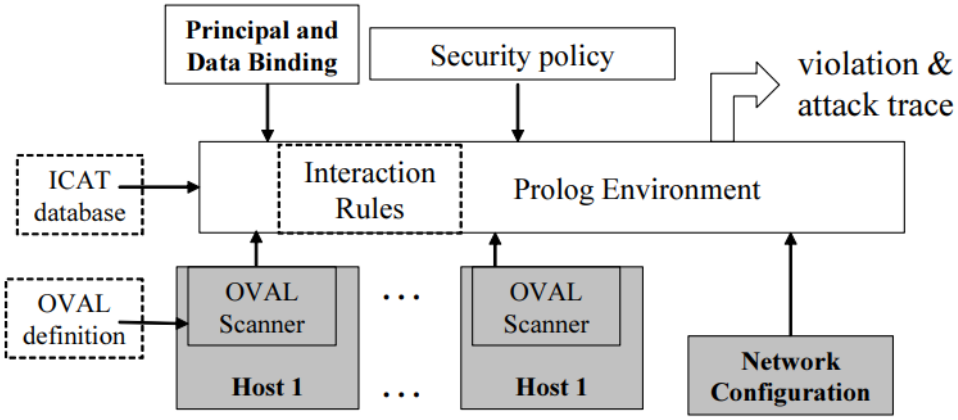

Figure 3.5: MulVAL framework from [30]

3.3.1 Datalog modelling language

Datalog is a declarative logic programming language that is considered a subset of the Prolog programming language. As opposed to Prolog, Datalog relies on a bottom-up approach which turns out to be particularly useful for dealing with deductive databases and applies well to networking scenarios. Datalog consists of facts and rules. Facts are simple atomic statements that hold true and are used to symbolically describe the properties of the system being analysed. Rules on the other hand define how new facts can be derived as a logical consequence from other facts. The syntax used to define facts is the following:

```
parent(mario, luca).
```

This fact specifies a parent relationship and is read as "mario is a parent of luca". As for rules they are defined in the following way:

```
sibling(PersonA, PersonB):-
    parent(Parent, PersonA).
    parent(Parent, PersonB).
```

Which is read as: "if Parent is a parent of PersonA and Parent is a parent of PersonB, then PersonA is a sibling of PersonB".

All properties that start with a lowercase letter are considered constants, while the ones starting with uppercase letters are variables.

In the previous case the rule is defined with uppercase properties because it applies to any two people that have the any same parent.

In MulVAL, Datalog is used to model all of the input described in Section 3.3.2. This input gets loaded into the XSB environment [31], which supports

Name	Developers	Accessible	AG Type	Scalability	Intuitive Level	Year	No. of References
Attack Graph Toolkit	Carnegie Mellon University	Open source	SG	Poor, Exponential	Fair	2005	52
MulVAL	Kansas State University	Open source	LAG	$O(N^2)$ - $O(N^3)$	Good	2005	938
TVA	George Mason University	Not open source, difficult to obtain	EDG	$O(N^3)$	Good	2005	578
SkyBox View	SkyBox Security, Inc.	Commercial Software	Unknown	$O(N^3)$	Good	2005	39
NetSPA	Massachusetts Institute of Technology	Not open source, difficult to obtain	MPAG	O(NLgN)	Fair	2006	493
SeaMonster	Norwegian Univ. of Science and Technology and SINTEF research foundation	Open source	AT	Polynomial	Fair	2008	59
Cauldron	PROINFO Company, George Mason University	Commercial Software	EDG	$O(N^3)$	Good	2011	150
FireMon	FireMon, Massachusetts Institute of Technology	Commercial Software	MPAG	O(NlgN)	Good	2012	16
CySeMoL	Royal Institute of Technology Stockholm, Sweden	Not open source, difficult to obtain	Unknown	Polynomial	Not Provided	2013	196
CyGraph	MITRE	Not open source, difficult to obtain	Unknown	Scales Well	Very Good	2016	94
SAGE	Delft University of Technology, Netherlands, Rochester Institute of Technology, US	Open source	Alert-driven	NA	Good	2021	8
AttacKG	Zhejiang University, National University of Singapore, Northwestern University	Open source	CTI-based	NA	Fair	2021	0

Table 3.1: Summary of various Attack Graph Tools, taken from [29]

tabled execution of Prolog programs. Tabling allows to skip the recomputation of previously computed facts, making the reasoning process of building an attack graph much faster. Moreover with tabled execution, the order in which rules are specified does not affect the end result, therefore providing a declarative style programming paradigm.

3.3.2 MulVAL input

As shown in Figure 3.5, in order to carry out its analysis, MulVAL receives as input information regarding advisories, host configuration, network configuration, principals, interaction rules, and policy, all of which must be expressed in the form of Datalog clauses. The main input concerning information about advisories, host configuration, and network configuration, which accounts for the greater part of the input data, is gathered by means of network and vulnerability scanners running independently on different hosts. MulVAL as of itself presents integrated compatibility with two popular vulnerability scanners, being OVAL [32], and Nessus. In fact built-in parser functionalities have been put in place by the developers in order to automatically convert the output of these two scanners to Datalog primitives.

Advisories

Advisories comprise the vulnerabilities that have been reported from the scanners, which run independently on each host. As can be seen in Figure 3.5, MulVAL is integrated with the OVAL scanner [32]. Vulnerabilities are represented by the following Datalog clauses:

```
vulExist(webServer, 'CAN-2002-0392', httpd).
vulProperty('CAN-2002-0392', remoteExploit, privilegeEscalation).
```

The first one specifies the existence of a certain vulnerability with code "CAN-2002-0392" affecting the https service hosted by the *webServer* machine. While the second one defines the exploitability range and consequence properties of the vulnerability identified by "CAN-2002-0392", which in this case are the possibility to be exploited remotely and the higher privilege obtained when the vulnerability is successfully exploited. These clauses are primitive facts that will be used by MulVAL to derive the final attack graph. Since the default vulnerability scanners do not provide information about the range and consequence properties linked to a given CVE identifier, MulVAL is capable of assigning them automatically by extracting the needed information from the NVD database, which at the time of developement was still known as the ICAT database as can be seen in Figure 3.5.

Host configuration

Host configuration gathers information regarding each host, such as exposed network services, local client programs, set uid executables, file paths privileges, exported network file systems. This data is mapped to the following primitives:

```
networkService(Host, Program, Protocol, Port, Priv).
clientProgram(Host, Program, Priv).
```

```
setuidProgram(Host, Program, Owner).
filePath(H, Owner, Path).
nfsExport(Server, Path, Access, Client).
nfsMountTable(Client, ClientPath, Server, ServerPath).
```

This data combined with the network configuration data provides a complete overview of the system chosen as target of evaluation, therefore the accuracy of this information is crucial for performing multistage attacks.

Network configuration

Network configuration models everything that has to do with how hosts communicate with each other. This includes information about network topology and firewall rules. In order to achieve this in the most simple way possible, MulVAL relies on the concept of host access control list (hacl). Hacl specifies for each host, who is able/allowed to reach it, and through which protocol and port.

```
hacl(SrcHost, DstHost, Protocol, Port).
```

Through hacl clauses it is possible to model in a simple and effective way the otherwise complex rules and conditions that influence the reachability of each host, such as switch and router locations, and firewall rules. Network information can be either provided manually or by some other automatic tool capable of scanning topology and accessibility of a system (OVAL does not provide this information). In the latter case a parser must be put in place to translate the data into the corresponding Datalog clauses.

Principals

This is where the correspondence between user accounts, hosts, and privilege level is specified. These bindings are modelled must be provided manually by the system's administrator, as scanners are not capable of extracting this information.

```
hasAccount(admin, dbServer, root).
```

The example above states that there is an *admin* user account with root privileges on the *dbServer* host. The properties are all lower case because they are constant strings representing an instantiation of the *hasAccount* primitive.

Interaction rules

The interaction rules represent the logical steps through which it is possible to derive new facts from already existing ones. Interaction rules are used by MulVAL to model the generic attack strategies rather than the single vulnerabilities, therefore

they do not need to be updated every time a new bug is discovered. MulVAL uses its own default set of interaction rules in order to produce attack graphs, however it also allows users to provide custom sets of rules in order to tailor the resulting output to their liking. The syntax through which interaction rules are expressed is the one already discussed in Section 3.3.1. The following example is taken from the default set of rules, and shows how a code execution attack can be inferred.

```
execCode(Attacker, Host, Priv) :-
  vulExists(Host, VulID, Program),
  vulProperty(VulID, remoteExploit, privEscalation),
  networkService(Host, Program, Protocol, Port, Priv),
  netAccess(Attacker, Host, Protocol, Port),
  malicious(Attacker).
```

It is possible to see that a malicious attacker can obtain code execution on a host with a certain privilege level (which varies depending on the privilege held by the vulnerable program), if the attacker has network access to a remotely exploitable service hosted by the target host resulting in privilege escalation.

All in all interaction rules embody one of the most important ingredients of the whole analysis process, therefore special care must be put into their selection, especially for non common scenarios such as OT. If they are not designed properly, some attacks might go undetected, leaving potentially critical system assets exposed to malicious actors.

Policy

Policies are made up of *allow* clauses which bind data access to specific users. They follow a whitelist architecture so everything that is not explicitely permitted is forbidden.

```
allow(admin, write, dbUserTables).
```

The previous is an example of a policy which only allows the admin user to write data to the user database. As interaction rules and principals, if there are policies in place, they must be provided by the system administrators.

Attack information

MulVAL also requires information about attacks in order to compute attack graphs. This information includes initial attacker location and privileges within the target of evaluation network, and also the list of attack goals to check for. This data is expressed through the following Datalog clauses.

```
malicious(attacker).
attackerLocated(ews).
attackGoal(dos(attacker, plc)).
```

In the example above the first primitive clause defines a new malicious principal in the system identified by the *attacker* symbol. The *attackerLocated* primitive states that *attacker* has access to the *ews* host. And the *attackGoal* primitive instructs MulVAL to check for a possible denial of service attack launched from the previously defined *attacker* principal targeting the *plc* device.

It is very important to specify the attack goals, as they represent the terminating conditions of the recursive graph building algorithm. And also if not carefully selected it is possible for some potentially dangerous attacks to go undetected.

3.3.3 MulVAL architecture

A global overview of the file structure is given in Figure 3.6.

As can be seen, MulVAL as a whole is quite a big project, therefore it is made up of many different modules each working in conjunction with one another, and due to its size and computational nature, multiple programming languages have been adopted across the various modules depending on the role of each one within the overall framework.

Starting from the top layer of the tree and proceeding from left to right, the first found directory is *bin*. The bin directory contains all of the compiled binary executable files corresponding to the source code files contained in the *src* directory.

Next in the list is the *doc* directory, which only contains a readme file detailing the usage of the software.

The first interesting folder to be found is the *kb* folder. This is where all the interaction rule files should be stored for usage. Here in fact it is possible to find also the default rules which are saved in the *interaction_rules.P* file (.P is the extension used for identifying Prolog files).

Moving on there is the *lib* directory, which contains all the external library files required by MulVAL, such as the mysql-java-connector for interfacing with the vulnerability database.

The *src* directory is where all the source code lies. It consists of four subdirectories: analyzer, attack graph, metrics, and adapter.

The *analyzer* subdirectory contains the XSB Prolog files for setting up and running the XSB environment, and actually computing the attack trace.

The *attack graph* subdirectory contains the code for constructing the graph data structure from the attack trace produced by the analyzer. This task has been implemented in C++.

The *metrics* subdirectory contains the Java coded algorithms for performing a metrics based analysis on the graph, taking into account information about vulnerabilities, such as impact, CVSS score, risk probability, and possible others.

The last subdirectory of *src* is the *adapter* subdirectory, which contains all of the Java implemented features for parsing the output of the compatible network

vulnerability scanners and generating the corresponding primitive Datalog clauses to then be passed to MulVAL. Among these features there is also a functionality for retrieving range and consequence properties given a CVE code from a local vulnerability sql database synched with NVD.

Moving back up a level, there is the *testcases* directory, which includes a few test network configurations in Datalog format, ready to be passed to MulVAL as input.

The last directory that is displayed in Figure 3.6 is the *utils* directory. This is where all the utility bash scripts reside, including the *graph_gen.sh* file, which is the main command that should be run in order to launch MulVAL from the command line.

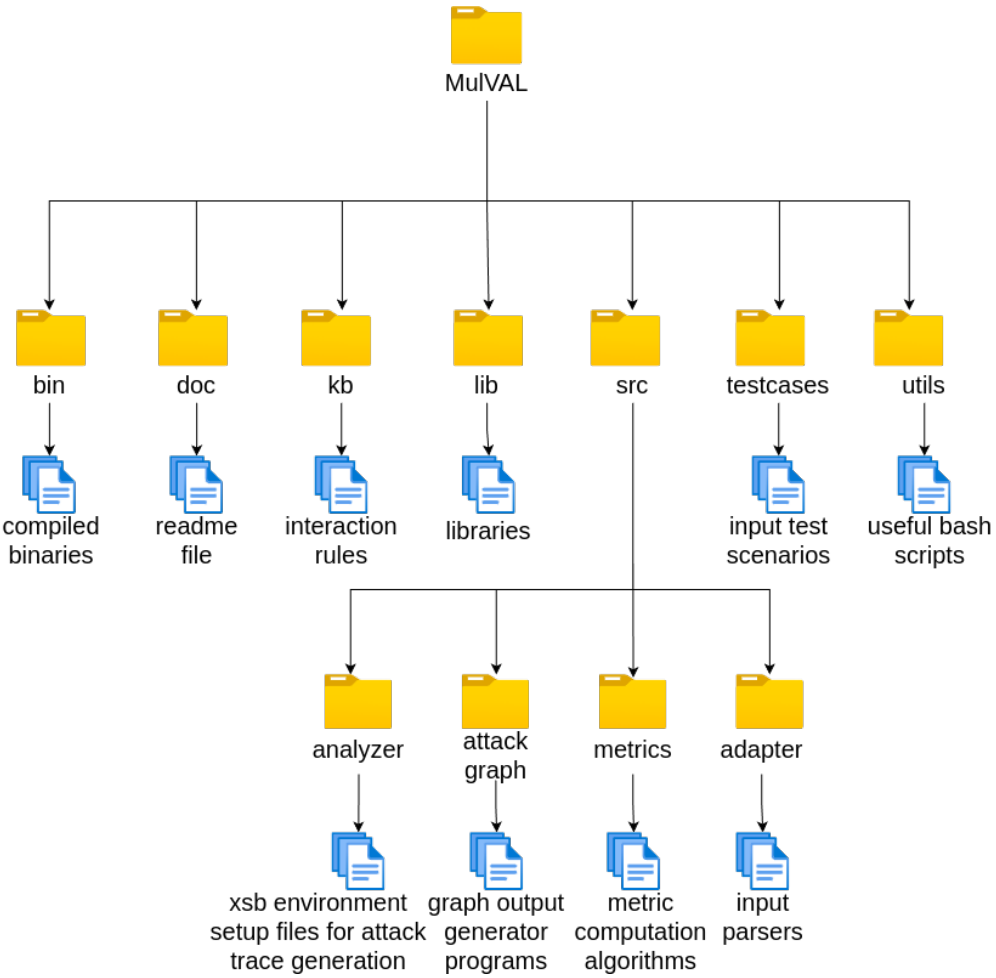

Figure 3.6: MulVAL directory tree

3.3.4 MulVAL usage

MulVAL is designed to be used from the command line. In order to run it needs the XSB and dot commands to be already installed on the system and included in the *$PATH* environment variable.

XSB, as mentioned before, is a compiled dialect of the Prolog programming language that allows to execute Prolog programs with the addition of the so called tabling feature, addressed in Section 3.3.1. This point is very important because without tabling MulVAL would not be scalable to big network scenarios as the attack graph generation would take too long to complete.

On the other hand, dot is a file format extension that is used for textually representing graph structures so that they can be interpreted and rendered by the GraphViz graph visualization software through the dot command.

In order to run properly it is necessary to define an environment variable called *$MULVALROOT*, and to set its value to the absolute path to the root MulVAL folder shown in Figure 3.6.

Once this configuration is out of the way, it is then possible to launch MulVAL from the command line by executing the *graph_gen.sh* bash script in the *utils* folder. This program takes as mandatory argument the file (with a .P extension) containing the Datalog primitives describing all the information discussed in Section 3.3.2.

The set of facts and rules used by MulVAL can be specified through the *-r* option followed by the path to the rule file of choice. If the -r option is not specified, the default interaction rules apply (which are the ones saved in the kb folder shown in Figure 3.6).

For the output it is also possible to specify the *-v* option, which produces a pdf visual representation of the graph through GraphViz.

3.3.5 MulVAL output

After having performed the computation of the attack graph for the given input, MulVAL proceeds to output the result in several different formats. The main ones are txt, xml, csv, dot, pdf, and eps. The first four are textual and the latter are graphical.

In order to textually represent these graphs, information about nodes and their corresponding connections must be stored. For each node of the graph, the following information is defined: node id, node label, node type, and metrics.

- **Node id**: is a unique integer number that identifies the corresponding node within the graph.
- **Node label**: is the text associated with a fact or an interaction rule.

- **Node type**: specifies whether the node is a *LEAF* node, which is a node that has no parents, basically it is a primitive fact node; an *AND* node, which identifies rule nodes because by definition a rule applies when all of its input facts hold true, therefore it represents an and condition; or an *OR* node, which identifies derived facts, as they can be reached through different paths among which it is enough for at least one to hold true.

- **Node metrics**: represent information about the risk associated with vulnerabilities, for example probability coefficients.

An example graph is given in Figure 3.7. The content of each node is reported in Table 3.2, which also shows how the textual files would store the information about each node. By inspecting the graph, it is possible to note that MulVAL utilizes different shaped boxes to represent each node according to its type. Primitive facts, corresponding to *LEAF* nodes, are enclosed by rectangles, interaction rules, corresponding to *AND* nodes, are enclosed by ellipses, and derived facts, corresponding to *OR* nodes, are enclosed by diamonds.

Graph edges, on the other hand, are stored as ordered pairs of node ids (source node followed by destination node), as can be seen in Table 3.3.

For the txt, xml, and dot formats, all the information belonging to nodes and connections are stored in the same file.

For the csv format on the other hand, the output is divided into two files. One stores the previously described information about each node, and is called *VERTICES.CSV*. While the other stores all the edge connections of the graph as ordered pairs of node ids, and is called *ARCS.CSV*.

The pdf and eps formats are used to store the graph in its graphical representation. They are generated from the dot file (which is a textual representation that can be interpreted by GraphViz) by GraphViz when the -v option is specified.

ID	Label	Type	Metrics
1	attackerLocated(laptop1)	LEAF	1.0
2	hacl(laptop1, webServer, https, 443)	LEAF	1.0
3	RULE 6 (Direct network access)	AND	1.0
4	networkService(webServer,webPortal,https,443,admin)	LEAF	1.0
5	vulExists(webServer,9,webPortal,remoteExp,privEsc)	LEAF	1.0
6	netAccess(webServer, https, 443)	OR	1.0
7	RULE 2(remote exploit of a server program)	AND	0.75
8	execCode(webServer,admin)	OR	0.75

Table 3.2: Example of node information

Source Node ID	Destination Node ID
1	3
2	3
3	6
6	7
4	7
5	7
7	8

Table 3.3: Examples of node connections

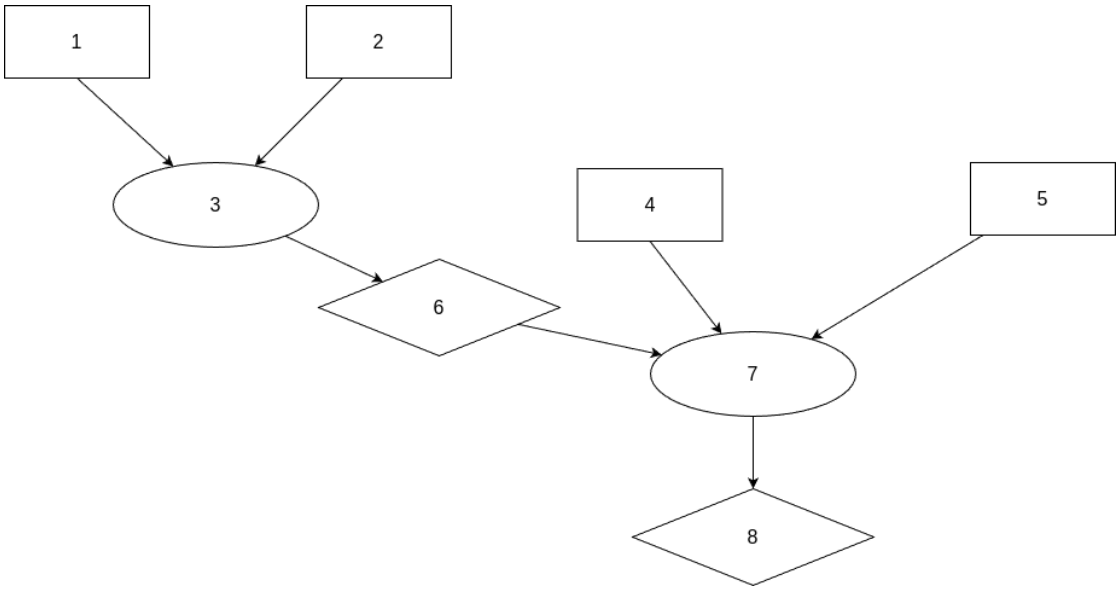

Figure 3.7: MulVAL attack graph example

3.4 Limitations of MulVAL

Even though MulVAL is the best option for effectively computing attack graphs in reasonable time, it still presents some limitations. Many of which have been analysed to some extent by the available literature.

The authors of [33, 34] proposed a set of extended interaction rules for modelling more complex network scenarios.

In [35, 36], the authors attempted to build an automated interaction rule generator which is capable of creating new interaction rules by extracting vulnerability information from the NVD given a CVE code.

In [37, 38, 39], the authors worked on improving the output of MulVAL by refining the output graphs with custom data reduction algorithms.

[40, 41] concentrated on improving MulVAL by adding support for risk analysis through the use of Bayesian models.

[42, 43] proposed a MulVAL based end-to-end framework enhanced by deep reinforcement learning for identifying the optimal attack paths and performing a automatic penetration testing on basic networks.

This work focuses mainly on extending MulVAL to support OT environments, and reducing the complexity of the output attack graphs. Additionally, it also focuses on another limitation that hasn't been analysed much so far, which is the low input compatibility of MulVAL with automated scanning solutions. Each of the previous limitations is explained in the following subsections.

L1 - Low input interoperability

This limitation stems from the fact that MulVAL's input is defined in the form of Datalog facts, as discussed in Section 3.3.2. The problem with this is that most of the information gathered during network reconnaissance must be manually mapped to the corresponding Datalog primitives, with the exception of data related to vulnerabilities obtained from the supported scanners (Nessus, OVAL).

Since in the greater scope of the project the entire security assessment process must be fully automated, it is necessary for MulVAL's input to be generated automatically from the output of the reconnaissance step. In order to face this issue, a standard input model must be defined along with a custom parser for mapping the populated data structure to Datalog clauses.

The methodology that has been used for this is the one discussed in Section 4.2.

L2 - Low network modelling capabilities

This limitation addresses the poor network modelling capabilities of MulVAL's default interaction rule set, especially when applied to OT scenarios.

When it comes to modelling different attack techniques, and consequently all of the linked vulnerabilities, MulVAL's default interaction rule set falls short, as it only allows to represent issues that are linked to hosts and services.

However in the world of OT, the most commonly exposed vulnerabilities are the ones linked to network protocols, specifically serial protocols such as Modbus or even data link protocols such as ARP.

By enriching MulVAL's default rule set with new interaction rules, it becomes possible to generate attack graphs that can detect certain types of attacks that would previously go unnoticed, such as man in the middle attacks, sniffing and spoofing attacks, all while maintaining compatibility with the default rules. In order to support this, it is necessary to add new primitive and derived facts for describing

the network in a more granular way, including information about configured network protocols, subnets and their various types, and more.

The methodology for this step is discussed later in Section 4.4.

L3 - High attack graph complexity

This other limitation regards the complexity in terms of readability and moreover general understandability of attack graphs generated by MulVAL. It is in fact quite easy to notice how quickly the size of the graphs scale with the size of the network. If on top of that new and more granular facts and rules are considered, the size of the graph scales even faster.

This constitutes a problem, as an overly complicated graph does not bring any added value to the assessment, and if a human were to supervise the process it would not be of any use. Additionally it is important to highlight that most of the nodes in the graph correspond to transient network information derived by applying the interaction rules and therefore do not carry useful knowledge about the actual attack actions an attacker would take.

The methodology steps discussed in Section 4.6, describe how all these unneeded nodes can be filtered out resulting in a more compact and simplified graph without the loss of any attack related information.

Chapter 4

Methodology

This chapter presents the main methodology steps that have been taken in this work to design and build the first prototype of the engine for modelling OT networks, deducing possible attacks depending on the vulnerabilities and misconfiguations discovered during the reconnaissance phase (which is not included in the scope of the work as mentioned in the previous chapters), and selecting the list of validation tasks to be performed in order to test the feasibility of the detected attacks (also the execution of the tasks is not included in the scope of the work). The chapter will start from giving a global overview of the design of the engine to make it easier for the reader to understand the general architecture of the tool and the relationship between the various modules. Next it will move on to describe in greater detail the purpose of each module, highlighting some of the technical choices that were made during the actual implementation phase.

4.1 Design of the engine

The goal of the engine is to receive as input the full description of the network that has been selected as target of evaluation, enumerate the possible attack paths that an attacker could decide to undertake in order to compromise the system, and select and schedule the most suitable tasks from a collection in order to test the discovered paths.

As discussed in the Chapter 3, the method that has been chosen for modelling the possible attack scenarios is the generation of logical attack graphs. Through these it is possible to represent the information about network configurations and the relationships between the latter and existing issues, which could range between specific vulnerabilities linked to CVE ids and simple misconfigurations.

The framework that has been selected as the best option for the job is MulVAL, as discussed in the previous chapter.

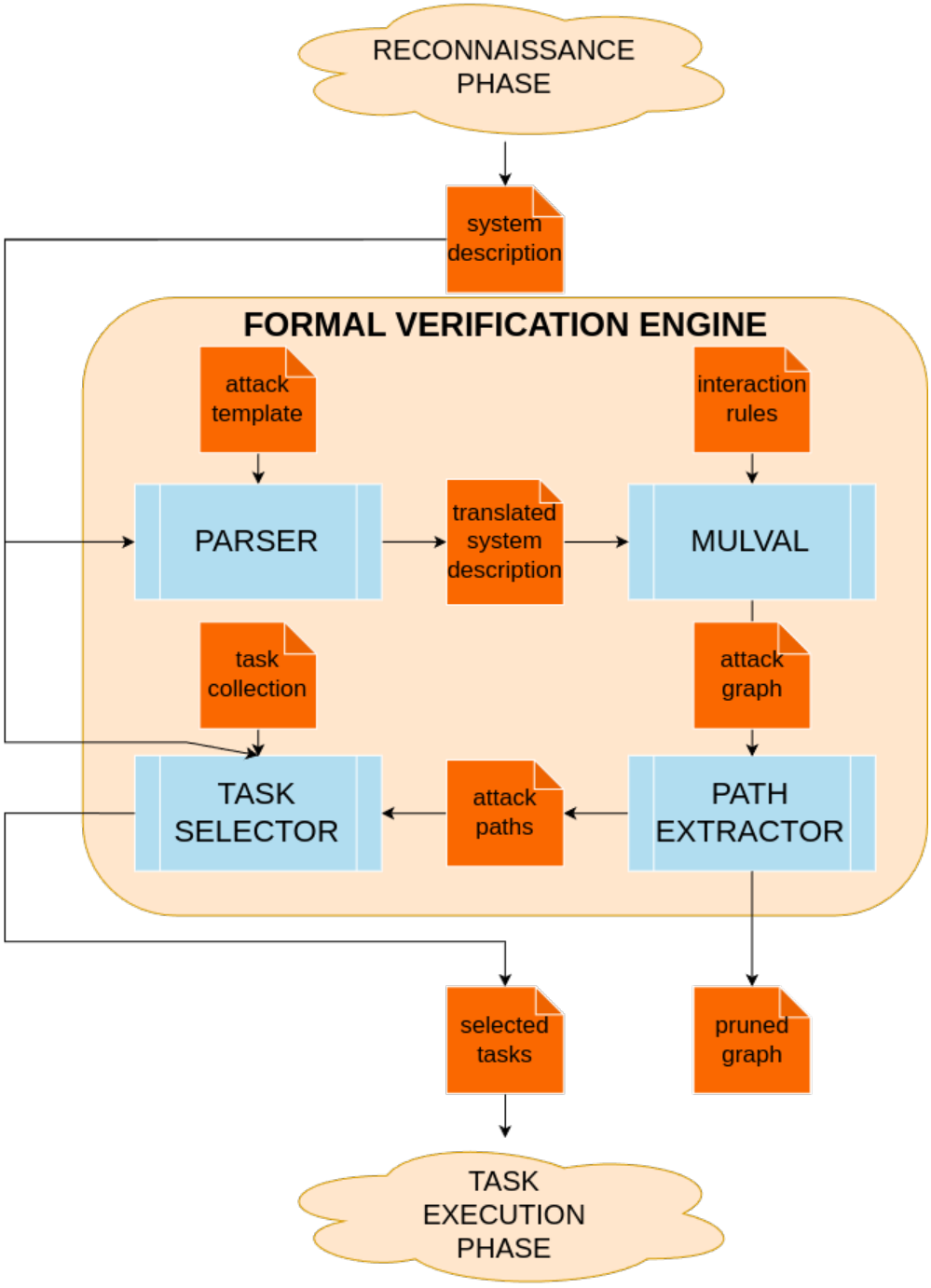

Figure 4.1: General architecture of the proposed engine

In regards to the input, a model has been defined for representing in one place all of the information gathered during reconnaissance in a standard and structured manner so that it could be easily generated automatically in the future and provide a simple way to be accessed by the resources requiring it.

From a design point of view, the engine has been split into four distinct modules:

- **Parser**: this module is responsible for receiving the network description and translating it into the corresponding set of MulVAL primitives represented as Datalog clauses, as discussed in the section about MulVAL in the previous chapter.

- **Extended MulVAL**: this module is the actual attack graph builder, and consists of the chosen open source framework MulVAL. This module takes as input the set of primitives that model the network, generated by the parser module, joined with the information about the attack goals to check against, along with the attacker's supposed location, which are stored in an attack template file. Additionally it can also receive an interaction rule file containing the set of rules that wish to be used to derive generic facts and attacks. This is where the actual extension is applied, by supplying MulVAL with a set of new interaction rules crafted to model OT scenarios and attacks in greater detail, allowing to produce more fine grained outputs and detect a wider range of attacks. From these inputs MulVAL computes the corresponding attack graph and saves it in different formats, such as graphical and textual.

- **Path extractor**: this module receives the textual representation of the attack graph generated by MulVAL, and by applying a pruning algorithm, transforms it from a logical attack graph into an exploit dependency graph. This step is important as ultimately the interesting information that needs to be extracted from the graph is the one concerning vulnerabilities, attacks, and the dependency relationships that exist among them. Therefore all of the information regarding network configuration, which accounts for a large quantity of the graph, is useful initially for computing the attack graph, but afterwards it can be removed as it already served its purpose. From the new reduced graph the module then extracts the single attack paths, consisting of an ordered succession of attacks linked to the corresponding vulnerabilities, and stores them in an appropriate data structure.

- **Task selector**: the task selector module, is the one responsible for interpreting the attack paths produced by the path extractor module, and applying a search algorithm for selecting the appropriate tasks to execute in order to test each path. The selection can be done on the basis of parameters such as vulnerable service versions, vulnerable protocols, CVE codes, exploit types, and so on.

The output will be a file containing a data structure representing all the different tests that have been selected for each discovered attack path.

Figure 4.1 visually represents the architecture described above. It is easy to see how the previously stated modules, which are coloured in light blue, link together, and also what are the required inputs and produced outputs, which correspond to the orange files in the image.

4.2 Definition of a network description model

One of the first and most important steps to attend to when trying to automate almost any kind of process, is the definition of standard models for representing the input data in a complete and structured way. This is particularly important because it represents the blueprint of the data that all of the different modules require throughout their operation. It should therefore describe the common knowledge base in an easily accessible way, allowing the modules to perform queries and cross reference data in an efficient manner.

When designing the model it is paramount to keep in mind that the goal is to allow for data to be generated, represented, and accessed automatically, therefore it must be structured in a machine readable format. This step of the design addresses the *L1 - Low input interoperability* discussed in Section 3.4 by standardizing the input that will later be parsed and sent to MulVAL.

When it comes to modelling network configurations and its corresponding security related characteristics, the first step should be to identify the main data of interest, which in this case corresponds to information regarding hosts, protocols, topology and vulnerabilities.

The relationships among elements of these categories dictate whether certain attacks could be carried out successfully or not. In fact, the presence of vulnerabilities is not enough to conclude that a system can be exploited, as said vulnerabilities might be isolated from the hypothetical attacker.

The proposed model follows a hierarchical structure consisting of a root object containing an array of subnets, an array of vulnerabilities and an array of data flows.

The format that has been used to represent this information is JSON, as it is a popular standard for modelling hierarchical relationships in the form of strings and allows for easy parsing due to the wide support offered by many programming languages. However this is only a matter of preference, the most important part is the actual data structure, which is independent from the underlying modelling language of use.

An example data structure is provided in Section A.1 of the appendix. In the future more information could be added to the model if needed but for now only

the essential data has been considered, as this work is an initial prototype.

Subnet information

Each subnet contains the general information about itself, such as subnet name, subnet type, which could correspond to *ipSubnet*, *bus*, *physical*, *wireless* (in the example referred to in Section A.1 they are all *ipSubnet*), subnet ip address, subnet mask, the array of layer 2 protocols in use, and last but not least the array of host objects which are part of it.

The information about the layer 2 protocols is useful for modelling possible man in the middle attacks caused by inherent protocol weaknesses.

Host information

The information regarding the single hosts connected to the network is modelled to include the following data: ip addresses, host names, operating systems, open ports, running services (local or remote), privilege levels, device name, device vendor.

As can be seen in Section A.1, this information resides inside each host object contained in the hosts array of each subnet. All of this info will be useful for task selection and task execution phases, since many exploits rely on this data in order to be searched or executed.

Topology and access information

In real world case scenarios, attackers usually don't have access to most of the devices connected to the target network, and therefore must be able to work with the few access points they can interact with. This requires them to infiltrate the network one step at a time, by first compromising the direct entry points and then pivoting from one machine to the next by exploiting newly exposed vulnerabilities that previously weren't reachable due to topology and access restrictions.

If an attacker knew the topology prior to launching an attack, he could pin point critical assets and plan out the optimal path to reach his goal without having to go through blind attempts.

Knowledge about topology is therefore fundamental for modelling multistage attacks, and must be properly represented by the chosen model.

In this work, information about topology has been modelled through an access control list which is represented as a *hacl* array in each host object, as can be seen in Section A.1. This list specifies for the corresponding host, which hosts can communicate with it, through which protocol and on what port.

For simplicity, it is assumed that all hosts within the same subnet can reach each other on any port, this is the reason why all the *hacl* arrays are empty, as all connections are implicitly modelled in this way. In the future this ought to be

changed, since it is not always guaranteed that all hosts within the same subnet can communicate with each other freely.

Data flows

Also data communication sessions could be added to the model in order to represent the open channels established between network devices and data in transit. This would be useful for modelling vulnerabilities and misconfigurations of application layer protocols, such as unencrypted traffic.

Data flows are stored in the *data_flows* array, which contains a list of objects representing each data flow through a flow name, a source host, a destination host, an application layer protocol used for communication, and the corresponding port.

Vulnerability information

The model must provide a way for representing the potential vulnerabilities discovered during the reconnaissance phase. These vulnerabilities could be of different types depending on the corresponding resource they expose.

For the purpose of this work, vulnerabilities can be linked to a host's service, a layer 2 protocol or a data flow.

They are stored in an array of objects denoted as *vulnerabilities*, and for each of them several properties are defined, such as: *id*, which is a unique identifier within the model; *cve* code, which specifies the global identifier that has been assigned by MITRE in their Common Vulnerability Enumeration framework; *type*, which specifies the type of target the vulnerability exposes (*service*, *l2_protocol*, *data_flow*); the corresponding target, which is represented as a set of properties that vary depending on the type of vulnerability (*service* type implies *host* and *service* properties, *l2_protocol* type implies *l2_protocol* and *subnet* properties, and *data_flow* type implies *data_flow* and *vul* properties); *range*, which specifies whether the vulnerability can be exploited locally, remotely, or adjacently from within the same subnet (*localExploit*, *remoteExploit*, *adjacent*), and consequence, which specifies the result of successful exploitation (*sniffing*, *privEscalation*, *impersonateDst*, *accessControlBypass*, and so on).

4.3 Description of the parser module

The parser module addresses the *L1 - Low input interoperability* described in Section 3.4. It is responsible for transforming the input description model into a set of Datalog clauses representing primitive facts used by MulVAL to generate attack graphs. The existence of this module is important in the context of automation, as it allows to decouple the input format of MulVAL from the input description model

which can easily be generated automatically from scanning tools. The MulVAL primitives could be part of the default set, defined in the default interaction rules file, as well as custom sets, defined in user supplied interaction rules which extend the default ones.

In the context of this work the parser is implemented as a Python script based off of six functions, each responsible for extracting specific data and mapping it to the corresponding primitives.

Following is a brief description of each of the six parsing functions, the primitives however will be discussed in greater detail in Section 4.4 about extending MulVAL.

- ***parse_subnets***: which iterates through the hosts array of each subnet and generates MulVAL primitives of the form:

  ```
  located(Hostname, Subnet, Type).
  ```

  The output falls under the subnet information category described in the previous section, as it provides correspondence between host and subnet of belonging along with the type of the subnet.

  In the case of this work, the output could also be considered to fall under the topology and access information category, seen as for simplicity it is assumed that hosts within the same subnet can freely communicate with each other.

- ***parse_l2_protocols***: this function extracts the layer 2 protocols in use in each subnet, from the l2_protocols array field, and for each one it generates a MulVAL primitive of the form:

  ```
  existingProtocol(Subnet, Protocol).
  ```

  These primitives constitute part of the subnet information category, as they are specific to each subnet rather than to each host. It is important to have this information included separately as some of the attack tactics and techniques in ICS (that can be found in MITRE's ATT&CK framework) may depend on vulnerabilities related to these protocols, especially since some of them have been specifically crafted for ICS scenarios, e.g. the profibus protocol.

- ***parse_hacl***: maps each host's access control list to a primitive of the form:

  ```
  hacl(SrcHost, DstHost, Prot, Port).
  ```

  This falls under the topology and access information, as through these primitives it is possible to describe the reachability conditions of each host which depends on the presence of physical/virtual connections (two hosts could be directly connected through a bus, or indirectly through a switch/router, or furthermore virtually connected through a vpn) and of access restriction rules,

which normally take the form of firewall rules put in place to limit the accessibility of each host to the bare minimum whilst maintaining correct network operation (example application of the least privilege principle of security).

- ***parse_data_flows***: this function is responsible for parsing the active communication channels established between hosts in the network for the exchange of data. For each channel it generates a set of two primitives of the form:

```
dataFlow(SrcHost, DstHost, FlowName).
flowBind(FlowName, Prot, Port).
```

  The data is extracted from the data_flows array field and therefore falls under the data flows category.

- ***parse_services***: extracts information about running services from each host and depending on whether they are local or remote (specified by the *type* field for each service in the services array field of each host), generates one of the two following primitives:

```
localService(Host, Service, Priv).
remoteService(Host, Service, Prot, Port, Priv).
```

  Seen as each active service is specific to each host, this information falls under the host information category.

- ***parse_vulnerabilities***: is responsible for mapping the information regarding vulnerabilities (found in the vulnerabilities array of objects field in the model) to the corresponding primitives, which are different depending on the type of the vulnerability (defined by the *type* property in each object in the vulnerability array of the model and described in the previous chapter).

```
vulLinkProtocol(LinkID,VulID,Protocol,Range,Consequence).
vulE2EProtocol(Host,VulID,Protocol,Port,Range,Consequence).
vulData(Data,VulID,VulType,Consequence).
vulExists(Host, VulID, Service).
vulProperty(VulID, Range, Consequence).
```

  The last two primitives refer to host vulnerabilities, they are split into two separate clauses because they are default primitives.

It is possible to see that the all of the needed network information described in Section 4.2 section has been encompassed by the defined parsing functions. However, since the model is likely to be extended in the future, also the parser will have to be extended accordingly.

4.4 Extending MulVAL interaction rules

In this section the *L2 - Low network modelling capabilities* limitation of MulVAL presented in Section 3.4 is addressed.

The default interaction rule set provided by MulVAL only allows to model certain types of network configurations and attacks. Specifically only vulnerabilities linked to hosts can be modelled and the types of attacks that can be detected are code execution, denial of service, and privilege escalation (which is a subset of code execution).

As for network configurations, all connections between hosts are expressed through *hacl* clauses. It therefore is not possible to model important topological information such as network segmentation, subnet connection type, adopted network protocols, and so forth.

These two points prove to be especially important for modelling OT scenarios. This low modelling capability clearly is not good enough for producing accurate results. As a consequence of this, many potentially critical attacks could go undetected and the use of MulVAL would lose most of its significance. A solution is therefore required in order to ensure a good coverage of the possible attack scenarios, especially when dealing with OT infrastructures. As the main vulnerabilities that can be found in these environments are in fact linked to the poor security of the deployed protocols, and to how the different devices are connected together.

The methodology presented in this work consists in extending the default interaction rules with a new set of custom rules and facts in order to map different kinds of attacks that are common to OT. For the scope of this work, the spectrum of attacks techniques that have been added to the default ones has been restricted to account for spoofing, traffic sniffing, and man in the middle (which is a subset of spoofing).

The rules and facts that have been used are the ones provided by [34], with the addition of a few customized ones. In [34] in fact the authors developed these rules with the specific goal of modelling network vulnerabilities linked to protocols and also different types of communication such as wireless and bus communication (which is common in industrial environments).

4.4.1 Physical topology and network communication

In order to better represent the typical layered structure of computer networks, the following facts have been added:

```
located(Host, Subnet, Type).
existingProtocol(Subnet, Protocol).
relay(RelayHost, SrcHost, DstHost, Prot, Port).
isMaster(Host, BusId).
isSlave(Host, BusId).
isAP(AP, WirelessRange, DstZone, Prot, SecurityConf).
dataFlow(SrcHost, DstHost, FlowName).
flowBind(FlowName, Prot, Port).
l2Connection(Dev1, Dev2, LinkId, Prot, Type).
```

Listing 1: MulVAL facts for topology and communication taken from [34]

All of them are primitive facts except for the last one which is derived.

The located clause allows to define subnets of different types through the *Type* parameter. The supported types are: *bus*, for serial bus communication; *ipSubnet*, for hosts within the same IP subnet; *physical*, for hosts located within the same wireless physical range.

The *existingProtocol* primitive allows to define layer 2 network protocols and bind them to the subnets where they're deployed.

The *relay* primitive specifies the presence of hosts configured to act as relay points for the communication between other hosts.

The *isSlave* and *isMaster* primitives are used for serial bus configurations, since they adopt a master-slave architecture.

The *isAP* primitive is used for representing wireless access points, bind them to the protocol they use to establish connections (e.g. WEP, WPA2, and the like), and whether it has secured access or it allows open access.

dataFlow and *flowBind* are used to model application level data flowing across the network from a source host to a destination host and bind the communication to a specific port and protocol.

The *l2Connection* derived fact represents layer 2 connectivity [1] among hosts. It is obtained by combining the previous primitives through different variations of the following interaction rule.

[1]The term layer 2 refers to the layered OSI architecture for network communication. Specifically to the second layer of the OSI model which is the data link layer used for routing data between hosts in the same local area network.

```
interaction_rule(
  (l2Connection(Dev1,Dev2,LinkId,Prot,ipSubnet):-
      located(Dev1,LinkId,ipSubnet),
      located(Dev2,LinkId,ipSubnet),
      existingProtocol(LinkId,Prot)),
  rule_desc('Ethernet link', 1.0)).
```

Listing 2: Layer 2 connection rule taken from [34]

The rule shown above deduces that there's a layer 2 connection between two devices (*Dev1* and *Dev2*) that are located within the same IP subnet using protocol *Prot*.

Another important thing to point out is that the use of *hacl* clauses is maintained for host access control and also for compatibility with the default rules. For simplicity the assumption that all hosts located within the same subnet can freely communicate with each other was made, which holds true in most network scenarios, in order to eliminate the need to specify all possible combinations of hosts with *hacl* clauses. This assumption is enforced through the following rule.

```
interaction_rule(
  (hacl(X,Y,_,_):-
    located(X,S,_),
    located(Y,S,_)),
  rule_desc('Two hosts in the same subnet can reach each other', 1.0)).
```

Listing 3: Custom rule for connection within same subnet

This rule states that any two devices X and Y located within the same subnet can access each other on any port and any protocol.

4.4.2 Host configuration

When it comes to host configuration mainly includes information about the different services running on each host. These services can be of three different types: local, remote, or login; and they are modelled by means of the following primitives.

```
localService(Host, Prog, User).
networkService(Host, Prog, Prot, Port, User).
isLoginService(Prog).
```

Listing 4: Host configuration primitives taken from [34]

The *User* property indicates the privilege level under which the service is being run. This information is especially important for code execution and privilege escalation attacks.

The *isLoginService* specifies that the program denoted by *Prog* allows users to login (e.g. ssh, ftp, and the like).

4.4.3 Principal access

Principal access refers to how a human actor, called principal, can access system assets, and also which assets he's allowed to access.

The extension rules provided in [34], differentiate between host access, network access, and data access.

Host access

Host access regards the ability of principals to login to hosts. The main fact used to represent this is the following.

```
localAccess(Principal, Host, User)
```

Listing 5: Access to host fact taken from [34]

This specifies that a certain principal can login to a host as a certain user. This fact is a primitive clause but at the same time it is also a derived clause, as it can be the consequence of other interaction rules.

```
interaction_rule(
  (localAccess(attacker,Host,User):-
    execCode(Host,User)),
  rule_desc('Principal can execute any codes', 1.0)).
```

```
interaction_rule(
  (localAccess(Principal,HostB, User):-
    hasAccount(Principal,HostB,User),
    networkService(HostB,Prog,Prot,Port,LoginServiceUser),
    netAccess(Principal,HostA,HostB,Prot,Port),
    hacl(HostA, HostB, Prot, Port),
    isLoginService(Prog)),
  rule_desc('A principal that has local access to a host (HostA)
            can use it to gain access via the network to a remote
            host (HostB) by using a network login service and a
            previously obtained account', 1.0)).
```

Listing 6: Local access rules taken from [34]

As it is possible to see, the first rule shows that local access can be obtained by an attacker who achieved code execution, while the second one shows that it can also be obtained remotely through a login service for which the principal owns a user account.

Network access

Network access regards the ability of a principal to access and interact with network resources such as communication channels and network services exposed by hosts.

Network access is in turn subdivided into link layer access and end-to-end access depending on whether a principal can directly access a host through the link layer or whether he can use an end-to-end protocol.

Link layer access is represented by the following derived clause.

```
l2Access(Principal, SrcHost, DstHost, Prot, LinkID, Type).
```

Listing 7: Layer 2 access derived fact taken from [34]

And it is obtained from the following interaction rule.

```
interaction_rule(
  (l2Access(Principal, SrcHost, DstHost, Prot, LinkID, Type):-
    localAccess(Principal, SrcHost, User),
    l2Connection(SrcHost, DstHost, LinkID, Prot, Type)),
  rule_desc('Access to a host through the link layer', 1.0)).
```

Listing 8: Layer 2 access rule taken from [34]

As shown in the code above, a principal can access a destination host from a source host if these he already has local access to the source host and if there is layer 2 connection between the two.

As for end-to-end access, the following derived clause is used.

```
netAccess(Principal, SrcHost, DstHost, Prot, Port).
```

Listing 9: Net access derived fact taken from [34]

The corresponding interaction rule is the following.

```
interaction_rule(
  (netAccess(Principal, SrcHost, DstHost, Prot, Port):-
    localAccess(Principal, SrcHost, SrcUser),
    hacl(SrcHost, DstHost, Prot, Port)),
  rule_desc('Access to a host through an end-to-end protocol', 1.0)).
```

Listing 10: Net access rule taken from [34]

A principal who has local access to a source host has network access to a destination host only if that the destination host is inside the source's access control list.

Data access

Data access regards the ability of a principal to access data in the network. This data could either be at rest, if stored on a host, or in motion, if transiting as traffic through the network.

In the case of data at rest the following default MulVAL primitive is used.

```
accessFile(Principal, Host, AccessPerm, Path).
```

Listing 11: Default MulVAL file access primitive fact

This primitive specifies that *Principal* can access a file at *Path* on *Host* with *AccessPerm* (read, write, execute).

Data in motion however is not supported by MulVAL's default rules. Therefore the following facts and rules have been added by [34] in order to allow this feature to be modelled.

```
accessDataFlow(Principal, FlowName, AccessPerm).

interaction_rule(
  (accessDataFlow(Principal, FlowName, view):-
    l2Connection(HostA, HostB, WirelessRange, Prot, wireless),
    located(SideHost, WirelessRange, physical),
    localAccess(Principal, SideHost, admin),
    dataFlow(HostA, HostB, FlowName, Direction)),
  rule_desc('ATTACKER_ACTION Access to data in motion', 1.0)).

interaction_rule(
  (accessDataFlow(Principal, FlowName, view):-
    l2Connection(HostA, RelayHost, WirelessRange, Prot, wireless),
    located(SideHost, WirelessRange, physical),
    localAccess(Principal, SideHost, admin),
    dataFlow(HostA, FlowName),
    relay(RelayHost, FlowName)),
  rule_desc('ATTACKER_ACTION Access to data in motion', 1.0)).

interaction_rule(
  (accessDataFlow(Principal, FlowName, view):-
    located(SideHost, Link, ipSubnet),
    located(HostA, Link, ipSubnet),
    located(HostB, Link, ipSubnet),
    localAccess(Principal, SideHost, admin),
    dataFlow(HostA, HostB, FlowName),
    vulFlow(FlowName, unencrypted, sniffing)),
  rule_desc('ATTACKER_ACTION Access to data in motion', 1.0)).
```

Listing 12: Data flow access rules taken from [34]

The first rule shows how a flow of data between two hosts connected through the same wireless link can be viewed by a principal that has local access as admin to a third host also connected to the same link.

The second rule is similar to the first one, with the difference that the communication between the source and destination hosts passes through a relay as they are not connected to the same wireless link. In this case a principal can access the data flowing between the source and the relay host if he's connected to the same wireless link.

The third rule is a custom rule that has been added in order to model access to data flowing in an ip subnet, if that data is not encrypted.

4.4.4 Vulnerability modelling

The whole point of using an extended rule set was to allow different types of vulnerabilities to be modelled in MulVAL, rather than only host related ones.

In fact, in [34] the authors differentiate between host related vulnerabilities, network related vulnerabilities, and data related vulnerabilities.

Host vulnerabilities

In order to model host related vulnerabilities, which consist in vulnerabilities linked to programs running on hosts, the default primitives have been maintained.

```
vulExists(Host, VulID, Program).
vulProperty(VulID, Range, Consequence).
```

Listing 13: Default MulVAL primitives for host vulnerabilities

These two primitives allow to bind vulnerable programs to the hosts they're running on, and also assign range (*local*, *remote*, *adjacent*) and consequence (*dos*, *privEscalation*) properties to them.

Network vulnerabilities

Network vulnerabilities consist in vulnerabilities that affect protocols that are deployed in the network, therefore they are not linked to any specific hosts.

In order to express this new type of vulnerability, the following primitives have been defined.

```
vulLinkProtocol(LinkID, VulID, Protocol, Range, Consequence).
vulE2EProtocol(SrcHost, DstHost, VulID, Protocol, Port, Range, Consequence).
```

Listing 14: Network vulnerability primitives taken from [34]

The first one relates to link layer protocols (e.g. ethernet, arp, and the like), which are the ones defined through the previously discussed *existingProtocol* primitive.

The second one, on the other hand, relates to end-to-end protocols, and therefore affects protocols used for host to host communication, hence the need for source, destination, and port properties to be specified. It is also possible to define a single host if the vulnerability is not affected by the direction of the communication.

Data vulnerabilities

Also data can present vulnerabilities. For example it might be unencrypted, allowing malicious actors who have access to the network to sniff it. Or it could also be unauthenticated, in which case attackers could hijack it and completely modify it without anyone noticing.

In order to express vulnerable data, the following clauses are used.

```
vulData(Data, VulID, VulType, Consequence).
vulFlow(FlowName, VulID, Consequence).
```

Listing 15: Data vulnerability facts taken from [34]

The first one is a primitive fact, and it is use to model intrinsically vulnerable data. The *VulType* property expresses whether the data is unencrypted or unsigned, while the consequence represents what could be the effect of this vulnerability, such as eavesdropping (*eavesdropping*), or data falsification (*dataFalsification*).

vulFlow on the other hand is a derived fact used to represent a vulnerability that affects the whole process of data communication. Just because the raw data might be unencrypted, it does not necessary mean that it can be sniffed by an attacker, as the underlying protocol used for transferring it could include some sort of builtin encryption.

```
interaction_rule(
  (vulFlow(FlowName, unencrypted, eavesdropping):-
    vulLinkProtocol(SrcHost, DstHost, VulID, Protocol, Range, eavesdropping),
    vulData(FlowName, VulID, unencrypted, Consequence),
    dataFlow(SrcHost, DstHost, FlowName),
    flowBind(FlowName, Protocol, _)),
  rule_desc('Eavesdropping on an unencrypted link', 1.0)).
```

Listing 16: Vulnerable data flow rule taken from [34]

As can be seen in the interaction rule above, eavesdropping on an unencrypted link is possible only if the data is unencrypted and the link protocol is vulnerable to eavesdropping.

Other variations of the interaction rule above have been used, but for the sake of brevity they haven't been reported in this work.

4.4.5 Attack modelling

When it comes to modelling attacks, as explained previously, MulVAL by default only considers code execution, privilege escalation, and denial of service.

Thanks to the newly added rules, it is now possible to model network specific attacks. At the moment, only spoofing, man-in-the-middle, eavesdropping, and access control bypass attacks have been added, however modelling new types of attacks is straightforward now that the network description capabilities have been enhanced.

The interaction rules below correspond to the previously mentioned attacks, however only one variation for each rule has been reported for the sake of brevity.

```
interaction_rule(
  (execCode(Host, Perm) :-
  vulExists(Host, VulID, Software, remoteExploit, privEscalation),
  networkService(Host, Software, Protocol, Port, Perm),
  netAccess(Host, Protocol, Port)),
  rule_desc('ATTACKER_ACTION remote exploit of a server program', 1.0)).
```

Listing 17: Default MulVAL code execution rule

The rule above is one of the default rules that correspond to code execution, which consists in an attacker gaining control of a system with some level of privilege.

In this case the reported variant of this rule represent code execution obtained by exploiting a remote network service hosted on a machine to which the attacker has net access.

Privilege escalation is a subset of this rule, as through code execution it is also possible, but not certain, that an attacker can elevate his privileges in order to gain a higher level of control over the compromised machine.

```
interaction_rule(
  (dos(Principal, DstHost):-
    networkService(DstHost, Prog, Prot, Port, NetworkServiceUser),
    hacl(SrcHost, DstHost, Prot, Port),
    vulExists(DstHost, VulID, Prog, remoteExploit, dos),
    netAccess(Principal, SrcHost, DstHost, Prot, Port),
    malicious(Principal)),
  rule_desc('ATTACKER_ACTION Network based DoS', 1.0)).
```

Listing 18: Denial of service rule taken from [34]

The rule above corresponds to a network based denial of service attack. By

leveraging this type of attack, an attacker can effectively block the correct operation of network connected machines running potentially critical tasks.

In the syntax of the rule it is possible to notice in fact that, in order for the attack to take place, a malicious principal must have net access to a host running a network service vulnerable to DoS attacks.

In the full set of rules there are other variants for modelling also host based DoS attacks, and also bus based DoS attacks.

```
interaction_rule(
  (spoofLinkHost(Principal, ImpHost, FoolHost, AttackerHost, trafficTheft):-
    vulLinkProtocol(FoolHost, ImpHost, VulID, Prot, adjacent, impersonateDst),
    l2Access(Principal, AttackerHost, ImpHost, Prot, Zone, ipSubnet)),
  rule_desc('ARP spoofing in the same subnet', 1.0)).
```

Listing 19: Link layer spoofing rule taken from [34]

The *spoofLinkHost* rule is used to represents spoofing attacks, which consist in malicious impersonation of a host participating in a communication by an attacker located on another host.

The rule reported above specifically refers to link layer spoofing attacks, as it only applies to hosts connected to the same layer 2 network link, which must be vulnerable to *impersonateDst*.

```
interaction_rule(
  (mitmLink(Principal, SrcHost, DstHost, SpoofingHost):-
    spoofLinkHost(Principal, SrcHost, DstHost, SpoofingHost, trafficTheft),
    spoofLinkHost(Principal, DstHost, SrcHost, SpoofingHost, trafficTheft)),
  rule_desc('ATTACKER_ACTION MITM attack in the link layer', 1.0)).
```

Listing 20: Link layer man in the middle rule taken from [34]

Man-in-the-middle attacks are a subset of spoofing attacks. In these type of attacks in fact, the attacker is able to hijack a communication between two hosts by spoofing in both directions. This results in the attacker being capable of manipulating the intercepted traffic to his liking by posing as both the sender and the receiver of the communication.

The reported rule refers to man-in-the-middle attacks at the link layer, therefore it only works for hosts sharing the same link connection.

In the full rule set that was used in this work, there are also other variations of this attack, such as end-to-end man-in-the-middle, which is the same concept but applied to application layer communication.

```
interaction_rule(
  (accessControlBypass(SrcHost, DstHost, Prot, Port):-
    located(SrcHost, SubnetA, ipSubnet),
    located(DstHost, SubnetB, ipSubnet),
    located(Gateway, SubnetA, ipSubnet),
    located(Gateway, SubnetB, ipSubnet),
    vulExists(Gateway, VulID, _, remoteExploit, accessControlBypass)),
  rule_desc('ATTACKER_ACTION Access control bypass', 1.0)).

interaction_rule(
  (hacl(SrcHost, DstHost, Prot, Port):-
    accessControlBypass(SrcHost, DstHost, Prot, Port)),
  rule_desc('Two hosts can reach each other if access restrictions
            have been bypassed', 1.0)).
```

Listing 21: Custom access control bypass rule

The access control bypass above was not part of the rule set provided by [34]. It has been added in order to model vulnerabilities of gateway firewalls, allowing attackers to bypass network access restrictions if successfully exploited.

The two rules above state that if a gateway between two subnets is vulnerable to access control bypass, then it is possible for two hosts to reach each other even if they are located in different subnets.

4.5 Attack template

Other than its main input, which should provide a full description of the network in need of evaluation, MulVAL also requires the attacker's initial location and attack goals to be explicitly specified. The main reason for this is that without the attacker's supposed location, MulVAL would not know where to start looking for attack paths, while the attack goals represent the termination conditions of the attack graph generation algorithm, so without them MulVAL would not know when to stop executing.

Since the engine must be integrated into a fully automated pipeline process, it should not be possible to have the user manually insert the attack goals to check for before every execution.

In order to face this problem, an attack template has been created. In this template (template.P) all of the main attack goals are specified by default, and every time the engine runs they get loaded automatically into an file (input.P), to which the input generated by the parser module (tmp.P) is then appended, as shown in Figure 4.2.

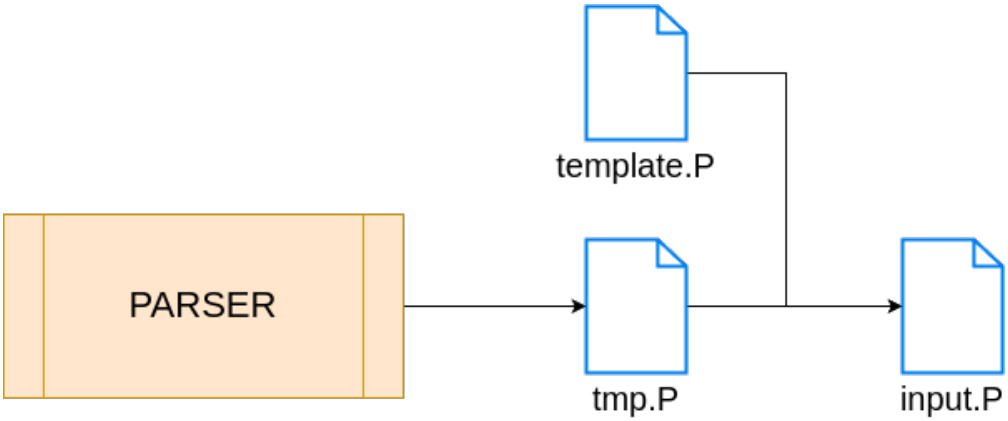

Figure 4.2: MulVAL input file generation schema

For the purpose of this work, the template file contains the following MulVAL clauses.

```
malicious(attacker).
attackerLocated(Host).
attackGoal(dos(attacker, _)).
attackGoal(execCode(_, _)).
attackGoal(mitmLink(attacker, _, _, _)).
attackGoal(accessDataFlow(attacker, _, _)).
```

The first line specifies the existence of a malicious attacker.

The second line defines the attacker's initial location, which must be changed in order to test for different access points.

The other lines correspond to the attack goals that must be considered. In this case they correspond to denial of service, code execution, man in the middle, and traffic sniffing (*accessDataFlow*). The underscores are wild cards used to represent any host.

4.6 Pruning the attack graph

This section addresses the *L3 - High attack graph complexity* limitation discussed in Section 3.4. This limitation concerns the fact that the size of the attack graphs generated by MulVAL scales very quickly with the size of the network, the number of considered vulnerabilities, and the number of facts and interaction rules used to model the system. As a direct consequence of this behaviour, also the complexity and readability of the graph rapidly increase, making it impossible for human operators to understand and for automated programs to navigate.

In order to solve this issue, this work proposes a custom graph pruning algorithm based on depth first search. This step of the process is included in the path extractor module of Figure 4.1.

The goal of the algorithm is to reduce the overall size and complexity of the graph as much as possible, by eliminating all nodes carrying unnecessary information. Before doing this however, it is important to define what kind of information must be kept and what can be done away with. Since the global objective of the verification engine is to produce a list of possible attack paths and corresponding tasks to be executed, the only useful information to maintain is the one regarding vulnerabilities and exploits. Hence, all the other nodes representing network configuration, which constitute the majority of the output graphs, can be safely removed, as they already served their purpose when generating the attack graph and no longer carry any utility when it comes to modelling attack paths.

After having defined which nodes can be kept and which can be deleted, a solution must be put in place in order to differentiate automatically between them. For what concerns vulnerabilities, it is pretty simple as all the related rules start with the string *vul*, which can therefore be used as a filtering criteria for selecting these types of node. As for exploit nodes, since there is no clear way to identify them, an *ATTACKER_ACTION* label has been added to the corresponding interaction rule descriptions, as can be seen for example in some of the rules shown in Section 4.4.5.

The algorithm itself, as mentioned previously, is based on the classic depth first search (DFS [2]) algorithm for graphs.

More precisely, it runs a DFS process starting from every attack goal node previously discovered by MulVAL. Which happen to be all the nodes of the graph that don't have any children and therefore correspond to path termination nodes.

Each instance of the DFS recursively explores the graph backwards starting from the attack goal that has been assigned to it. At every step of the process, it checks whether the current node is a vulnerability or an attacker action. If the considered node is neither a vulnerability nor an attacker action, the search continues to the next node and nothing else is done at this step. Otherwise, the behaviour of the process depends on whether the node happens to be a vulnerability or an attacker action. In case of vulnerability, the node is simply copied over to the resulting pruned graph, if not already present. While in case of attacker action, the node's

[2]The depth first search algorithm, in graph theory, is a recursive algorithm built for the exploration of graph structures. Given a starting node, the recursive procedure explores all the possible paths originating from it. The term depth first is used because the exploration process goes all the way to the end of a path at first and then backtracks to try all the other alternative routes it did not take before. The complexity of DFS is proportional to the number of vertices and edges of the graph to which it is applied.

child is copied over to the resulting pruned graph, also if not already present. This is because the actual attacker action is represented by the derived fact obtained by applying the rule labelled with the *ATTACKER_ACTION* string, as opposed to the rule node itself.

The final result, obtained once all the DFS processes have finished running, will be an attack graph containing only the nodes corresponding to attacker actions and to the vulnerabilities that are linked to them. In other words an exploit dependency graph. Example results are provided and discussed in section Section 5.2.

4.7 Extraction of attack paths from the graph

After having pruned the attack graph, it is necessary to separately extract all the possible different attack paths, in order to automatically plan out the sequence of attacks to test for in the future.

To achieve this, a data structure for representing attack paths has been defined. This structure contains for each path the sequence of exploits required to reach the final goal, and the identifiers of the vulnerabilities on which the exploits depend on.

As for the input data model described in Section 4.2 and showcased in Section A.1, also for the attack paths the JSON notation has been adopted, therefore maintaining consistency between the data representations used by the different modules.

In Section A.2, an example of extracted attack paths has been reported. From this example it is possible to see how each path is identified by a unique path id (e.g. *path_1*), and consists of an ordered array of objects. Each one of these objects corresponds to an exploit, and therefore contains: the identifier of the node in the original MulVAL graph, which is the same as the one used in the pruned graph; the exploit name, which is the text label found on the corresponding node of the graph; and an array containing the identifiers of all the vulnerabilities that the exploit depends on, which are the same identifiers used to uniquely define the vulnerabilities in the input model (*id* field of the *vulnerabilities* array reported in Section A.1).

The order in which the exploit objects are defined in each path is very important, as each exploit depends on the previous ones. Therefore if at least one of the exploits in any given path were to fail, all of the following ones cannot be executed.

Another important point to note is that the vulnerability identifiers allow to cross reference the input data model, making it simple to obtain all relevant information regarding the specific vulnerabilities, such as target host, protocol, range, consequence, and so on.

In order to extract the attack paths from the pruned graph, another DFS based algorithm has been developed. As in the pruning case, a separate DFS search

process is run starting from every goal node (nodes with no children). Each process recursively traverses the graph backwards, while keeping track of the current path by means of a path array. If an exploit node is reached, a new entry corresponding to it is added to the path array. If instead a vulnerability node is reached, its identifier gets added to the vulnerability array of the exploit it relates to.

Every time the recursion procedure reaches its termination condition (which happens when the current node has no parent nodes) a new path is obtained. Consequently the current contents of the path array are reversed (since the DFS search constructs the paths in reverse order starting from the final goal nodes) and saved in the JSON formatted structure described above.

4.8 Definition of a task description model

Referring to the engine architecture design shown previously in Figure 4.1, the step following path extraction is task selection.

This part of the overall security assessment processes consists in selecting the sequence of automated tasks (which are non other than automated exploits) to run in order to test the actual exploitability of the considered systems.

Before jumping straight into the task selection module, a task description model is defined in order to properly represent each task according to its properties. As for all the other data structures that have been addressed throughout this work, also the task description model adopts the JSON formatting notation.

In Section A.3, it is possible to see the data structure that has been used to define represent the task collection that has been used in this work. As can be seen, the task collection consists of a JSON array containing a list of objects, each corresponding to a task descriptor. Each task descriptor includes the following properties:

- ***id***: the unique integer identifier.
- ***qualified_name***: the name denoting the task.
- ***criticality***: gives a measure of the potential impact of the exploit tested by the task (high, medium, low).
- ***category***: an array containing all the labels related to the type of exploit tested by the task (*rce*, *privEscalation*, *dos*, and so forth).
- ***range***: specifies whether the exploit related to the task is local, remote, or adjacent.
- ***vulnerabilities***: array containing all the specific CVE codes linked to the task (it is an array because it is possible for a single task to exploit multiple vulnerabilities).

- ***preconditions***: array which contains restrictions on the use of the task, such as required user interaction, required user authentication, and the like. In this work this field is left empty, but it will turn out useful for modelling more complicated tasks in the future.

- ***input***: an array of objects used to define the input required by the task in order to run. Normally exploits require information such as target ip addresses, target ports, subnet ip addresses, files, and more. As can be seen in Section A.3, each input object has a type field, which defines what kind of input the object refers to (host, subnet, file). And a list of parameters (*param_1*, *param_2*, and so on) which specify the specific type of information required by the exploit (*address*, *port*, *path*, and the like).

  For example the *ignition-breaker-rce* task requires the ip address and port of the target host in order to be launched, while the eavesdropping exploit requires the ip address of the targeted subnet and the protocol that wants to be sniffed.

- ***output***: defines the output format of the task. For all of the tasks considered in this work the output is of the form of a *Passed* or *Not passed* string, indicating whether the exploit was executed successfully.

4.9 Task selection

In order to select the sequence of tasks that need to be executed to test the actual exploitability of the system, a simple task search algorithm has been developed as part of the task selector module.

As shown in the general architecture of the image Figure 4.1, the inputs required by the task selector in order to work are: the task collection (discussed in Section 4.8 and given in Section A.3), the computed attack paths (discussed in Section 4.7 and given in Section A.2), and the system description (discussed in Section 4.2 and given in A.1).

The task selector algorithm then loads the contents of these data structures containing the discovered attack paths, and iterates over them. For each vulnerability identifier linked to the exploit nodes of each path, the algorithm performs cross referencing with the system description data, in order to extract the consequence property related to the considered vulnerability. The value of the extracted consequence is then used to filter the task collection by checking whether it is included within the category array of each task.

The selected tasks are then saved in an appropriate data structure, of which an example is shown in Section A.4.

As can be seen in the provided example, the deployed data structure contains an object representing each attack path corresponding to the attack paths found by the path extractor.

Inside each path object, it is possible to find other objects corresponding to the single exploit nodes of the path. These exploit objects are identified by the value of the *exploit* field of the corresponding nodes in the attack path structure.

Each exploit object in turn contains two arrays. One is the *vul_ids* array, which keeps track of the vulnerabilities related to the exploit, and is the same array of the attack path structure. While the other is the *task_ids* array, which stores the list of identifiers of all the possible tasks that could be executed in order to test the specific exploit.

At the moment the task selection process filters the task collection only on the basis of the category of the tasks. However in the future this could be improved to allow the users to set custom filtering conditions on any of the task descriptor properties.

4.10 Overall pipeline

In order to run the whole verification engine and properly coordinate the operation of all the modules shown in Figure 4.1, that have been presented in the previous sections, a custom bash script has been developed and placed in the *utils* folder shown in Figure 3.6 and described in Section 3.3.3. The script goes by the name of *pipeline.sh*.

The formal verification procedure can then be executed by simply running the *pipeline.sh* command and passing the location of a network description file (Section A.1) as input parameter, therefore acting as black box and abstracting all the inner workings of the different modules. This allows for easy integration of the engine presented in this work within the process pipeline of the bigger project.

The script itself is responsible for separately executing the four main modules in the correct order, and linking the outputs of each with the inputs of the next.

1. The first module to be executed is the parser Section 4.3, which receives the description model of the target network and produces as output the corresponding Datalog primitives. This output gets concatenated to the attack information present in the attack template file Section 4.5 in order to create the MulVAL input file (input.P)

2. The second module to be executed is the extended MulVAL, discussed in Section 4.4, which receives the output of the previous step (input.P) and the extended interaction rules file as input, and generates all the different formatted attack graph files as output (Section 3.3.5).

3. The third module to be executed is the path extractor, which receives as input the *VERTICES.CSV* and *ARCS.CSV* files, generated at the previous step, and produces as output the corresponding pruned graph (Section 4.6 and the file containing the extracted attack paths (Section 4.7).

4. The last module to be executed is the task selector (Section 4.9), which receives as input the location of the task collection file (Section 4.8), the location of the attack paths file computed at the previous step, and the initial network description, and produces as output data structure containing the task execution plan for each attack path (Section 4.9).

The final output of the engine therefore consists of the list of tasks planned for execution, which will be processed by the task execution engine in the future, and the pruned attack graph, which could prove itself useful for human operators who wish to monitor the correct operation of the engine.

Now that the whole methodology has been presented, it is possible to see how the overall formal verification engine proposed in this work simply requires a generic network description in order to perform its assessment duties. And thanks to the defined input data model, it can be easily integrated with any sort of network scanner, customized or commercial, as all that must be done is express the gathered information according to the simple standard that has been presented in Section 3.3.2.

Chapter 5

Results and discussions

This chapter analyzes the main results that were obtained from validating the engine's operation when applied to a test network configuration.

Due to time related constraints, and to the focus of the work being mainly on methodology; the engine has been tested for two different cases of a single network configuration, in order to provide a simple proof of concept of the correct functioning of the proposed solution.

This chapter is therefore divided into two main sections. The first section, contains a complete description of the manually constructed network configuration that has been used as a testing environment. While the second one, illustrates the two considered test cases, corresponding to the different initial access points of the hypothetical attacker, and analyzes the results obtained for each one.

5.1 Test network configuration

In order to test the formal verification engine, a network configuration was manually setup to act as a test environment. This test environment is the one depicted in Figure 5.1. As can be deduced from the image, the test case scenario proposed in this work corresponds to that of a typical OT network infrastructure, seen as it hosts devices commonly found in this branch of technology (Section 2.1.3). Among these devices it is in fact possible to notice the presence of PLCs, HMIs, EWS, SCADA control server, and many I/O devices corresponding to sensors and actuators for controlling physical processes. The network architecture on the other hand also corresponds to that of a typical OT architecture, being SCADA (Section 2.1.4), as suggested by the presence of the SCADA server.

The data model reported in Section A.1 corresponds to the description of the network shown in Figure 5.1, including information about vulnerabilities, topology, and host configuration.

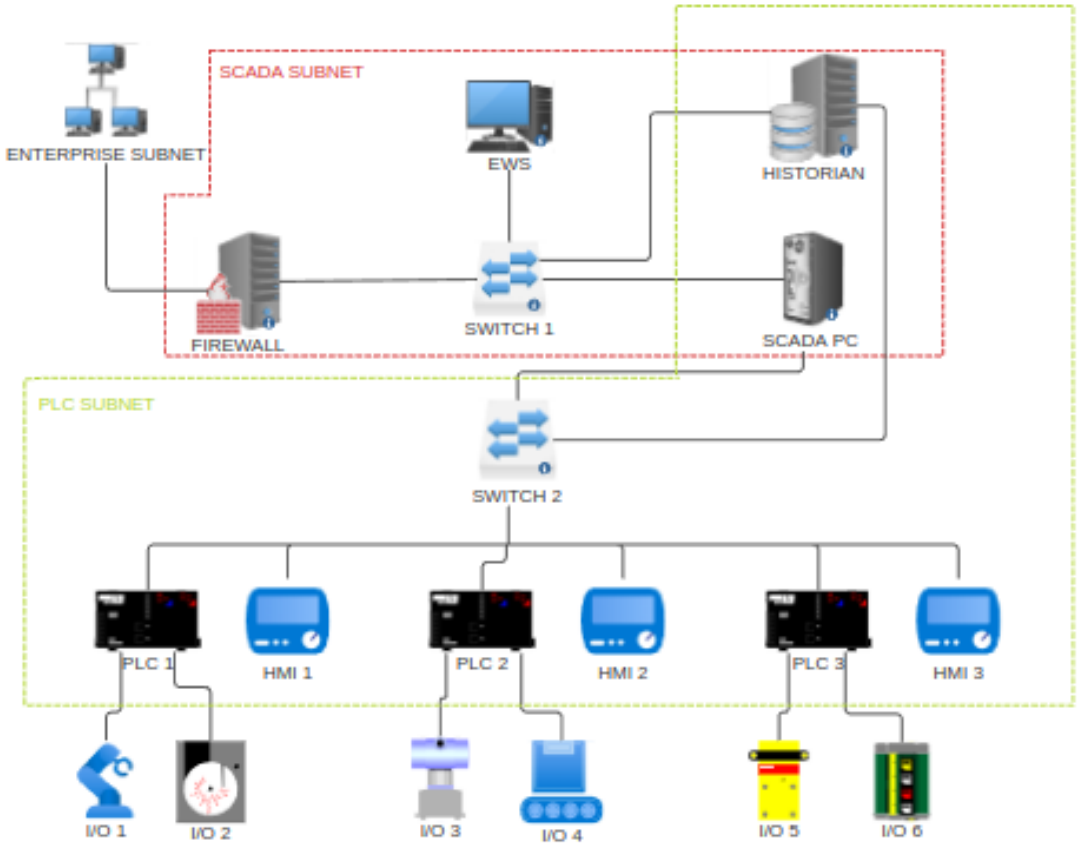

Figure 5.1: Test network configuration

As can be seen in the network schema, the network as a whole is subdivided into three different subnets, being: the ENTERPRISE subnet, the SCADA subnet, and the PLC subnet.

ENTERPRISE subnet

Nowadays, as already mentioned in Section 2.1.2, OT and IT are rapidly converging under the influence of the IoT paradigm. Due to this convergence, it is no longer uncommon to find IT systems interconnected with OT infrastructures, working together in order to enhance productivity and provide remote operation and monitoring capabilities to physically controlled processes.

Keeping this in mind, it is possible to see how the specific network scenario, addressed in this work, was built to include this IT-OT relationship. To this point in fact, the ENTERPRISE subnet, depicted in the top left of Figure 5.1, represents

the IT side of a company infrastructure responsible for managing some sort of industrial production line.

Since the test scenarios is mainly concerned with assessing the security of the OT part of the network, the architecture of the ENTERPRISE subnet has been represented as single node consisting of local area network of normal company PCs.

In order to limit the access to the OT infrastructure, a FIREWALL has been deployed to act as a gateway between the ENTERPRISE subnet and the SCADA subnet.

For more detailed information it is better to consult the JSON description model in Section A.1. It is in fact possible to see that the ENTERPRISE subnet, denoted as *enterpriseLan* in the model, consists of a *companyPC*, which is a normal Windows 10 desktop PC, and of the *fortim* firewall produced by Fortinet, which runs a remote service for filtering incoming traffic called *customPacketFilter* on port 9001. This service however presents a remotely exploitable vulnerability, with code "CVE-2022-40684", which allows attackers to bypass access restrictions and reach the SCADA subnet even without proper authorization.

SCADA subnet

The SCADA subnet, denoted as *scadaLan* in the JSON model, corresponds to the supervisory network. This part of the network, as explained in Section 2.1.4, is responsible for managing and coordinating the underlying control network (PLC subnet), by collecting, analysing and displaying plant data, computing useful statistics, and allowing operators to configure set points, and schedule automated production procedures.

As shown in the network schema, the SCADA subnet is composed by all the devices connected to SWITCH 1, which are: the FIREWALL, the SCADA PC, the EWS, and the HISTORIAN.

The FIREWALL has already been described in the ENTERPRISE subnet section. It has been included also in the SCADA subnet because it acts as a gateway, therefore it is equipped with two different network interface cards.

The SCADA PC, presented in Section 2.1.3, is the heart of the whole OT network, as it is the device responsible for managing all the main control features listed earlier. As can be seen in the JSON description, it consists of a normal HP desktop PC hosting an Inductive Automation Ignition web portal (denoted as *ignitionPortal* in the JSON model), which provides a graphical web interface that can be accessed through the https protocol on port 443 by authorized actors. By accessing this web service, plant operators can easily interact and work with the control functionalities offered by the system. The "Inductive Automation Ignition" portal however presents a vulnerability ("CVE-2023-39476"), which allows unauthenticated attackers to obtain remote code execution on the host machine by

exploiting an insecure deserialization bug inside of a required Java class.

Seen as the SCADA PC must be capable of interacting with the devices of the PLC subnet, it must also be equipped with two network interface card in order to communicate on both subnets.

The EWS, presented in Section 2.1.3, is the engineering workstation. This device simply is a normal HP PC that plant operators use to access the services offered by the SCADA PC and by the HISTORIAN.

The HISTORIAN, presented in Section 2.1.3, is a HP server used for collecting, storing, and displaying real-time operational data received from the PLCs of the PLC subnet, allowing plant statistics to be computed over large periods of time.

Since the HISTORIAN receives its real-time data from the PLCs, it must also be equipped with a second network interface card for communicating with the PLC subnet.

The main advantage of having this device as a separate component, consists in reducing the traffic load on the SCADA PC, especially since real-time data generated by field devices, such as sensors, accounts for a large part of the network load.

In order to perform its duties, the HISTORIAN hosts three different services which are: *grafana*, *mosquittoBroker*, and *influxDB*.

Grafana is a powerful open source platform for providing advanced data analytics and visualization features. It is mainly used by authorized employees as a support for business decision making.

As can be seen in the *scadaLan* subnet of the JSON model, Grafana can be accessed through a web interface on port 3000.

Both the *influxDB* and *mosquittoBroker* services will be discussed in the following section, due to the fact that they can only be reached by devices of the PLC subnet, as opposed to Grafana, which can be reached only from the SCADA subnet.

PLC subnet

The PLC subnet, denoted as *plcLan* in the JSON model, corresponds to the control network. This part of the OT network, as explained in Section 2.1.4, is responsible for ensuring that the physical processes, that the production plant depends on, function correctly.

These physical processes are controlled by PLCs, which, as described in Section 2.1.3, consist of basic computers capable of receiving input signals from the field sensors and calculating corresponding output signals to send to the field actuators. Each PLC is responsible for controlling the I/O devices of the field station it has been assigned to. These devices are non other than the previously mentioned sensors and actuators that constitute the physical processes. As shown in the schema, they are individually connected to their master PLC as peripheral devices,

rather than to the PLC subnet itself.

In order to allow the plant operators who walk the floor to monitor the operation of each process, each PLC is equipped with a HMI, which, as explained in Section 2.1.3, consists in a graphical interface offering basic monitoring and configuration functionalities.

As can be seen in the network schema, the PLC subnet includes all the devices connected to SWITCH 2, which are: the three PLCs, the three HMIs, the SCADA PC, and the HISTORIAN.

From the JSON model it is possible to see that the PLCs are Allen-Bradley PLCs, sold by Rockwell Automation, and they all run a service called *opcuaServer* on port 4840. This specific service allows the PLCs to act as servers by sending their operational data on demand to requesting client devices. This data is sent through the OPC-UA protocol, which is a typical protocol used in OT and has been already described in Section 2.1.5.

The *opcuaServer* service however presents a denial of service vulnerability (CVE-2022-25888), which allows attackers, who can reach it, to send an unlimited number of huge chunks of data, due to a bug in the *opcua* package, and therefore block the operation of the device.

The HMIs are Allen-Bradley Panel View devices configured also to be accessed remotely through ssh on port 22.

In regards to the role of the SCADA PC within the PLC subnet, it simply acts as a client that pulls data and pushes configuration changes to the PLCs by means of the *opcuaServer* service.

As for the HISTORIAN, the previous section already explained that it hosts two specific services for communicating within the PLC subnet.

The first one, as can be seen in the JSON model regarding the *plcLan* subnet, is an instance of *influxDB* running on port 8086. Influx DB is an open source database developed specifically for storing real-time data. In this case it will store the operational data received from the PLCs, which will then be analysed and displayed by Grafana in order to support business decision making.

The other service hosted by the HISTORIAN is the *mosquittoBroker* service. This service is based on the MQTT (Message Queue Telemetry Transport) protocol, which is a light weight publisher-subscriber protocol that has recently become very popular in the IoT field of application.

The MQTT protocol relies on a message broker, which is an intermediate entity that listens for data coming from publishers and notifies subscribers when new information becomes available. In the case at hand the *mosquittoBroker* service acts as the broker, which receives the data published by the PLCs, denoted as *plcData* in the data flows array of the JSON model, and relays it to the *influxDB* database, which acts as the MQTT subscriber. In this scenario however, the MQTT protocol presents a misconfiguration (CWE-319), as reported in the JSON model,

which allows attackers located within the same network to access the data being sent over it, seen as it hasn't been configured to provide any sort of encryption.

Last but not least, in regards to the link layer protocols, the PLC subnet deployes the ethernet protocol for layer 2 communication, and the ARP (Address Resolution Protocol) protocol for resolving link layer addresses. The ethernet protocol in this scenarios presents the same misconfiguration of the MQTT protocol, which consists in null encryption. While the ARP protocol is affected by a vulnerability (CVE-1999-0667) that allows attackers connected to the same link to spoof ARP responses and therefore impersonate other hosts on the network, leading to possible man in the middle scenarios.

5.2 Analysis of obtained results

The verification engine has been tested considering two different initial access points for an attack. The first case supposes that the hypothetical attacker has local access to the EWS and therefore is already located inside the SCADA subnet. For the second case, on the other hand, the attacker is supposedly located in the ENTERPRISE subnet and has local access to a *companyPC* (as denoted in the JSON model reported in Section A.1).

In order to reduce the size of the attack graphs and of the discovered attack paths, the engine has been set to search for the following attack goals: man in the middle between PLC 1 and HMI 1, dos on PLC 1, and traffic sniffing on *plcData1*, which corresponds to the data flow going from PLC 1 to HISTORIAN.

This restriction has been imposed because otherwise the results would include all the possible combinations of these attacks on the other PLCs, since they all expose the same vulnerabilities. In which case, the final resulting graphs and extracted attack paths would prove to be too large to display, and would not contribute to adding any meaningful information to what has already been produced.

5.2.1 Case of attacker located in the SCADA subnet

This is the simplest case of the two that have been considered. In this scenario in fact the attacker, as mentioned previously, is already inside the SCADA subnet and has local access to the EWS. This case is less likely to occur in the real world with respect to the second case, as it would require the attacker to have physical access to the network. However the eventuality of it occurring should not be completely discarded, as it would not be the first time an intruder gained physical access to restricted zones by impersonating authorized personnel.

After having set the attacker location to be EWS in the template file described in Section 4.5, and having defined the attack goals to the ones mentioned above, the engine produced the attack graph shown in Figure 5.2. This graph corresponds

to the final attack graph obtained after the pruning step discussed in Section 4.6. The content of each node of the graph is reported in Table 5.1 in favor of understandability.

For the sake of completeness also the full graph before pruning is shown in Figure 5.3. This has been included for the sole purpose of showing the difference in size and complexity between the pruned and non pruned version, therefore the content of the nodes has been removed for better representation.

As can be deduced from the pruned graph, in order to reach the prescribed goals, the attacker located at the EWS must first exploit the vulnerability of the *ignitionPortal* service (node 1) to obtain code execution on the SCADA PC as admin (node 7).

After having successfully compromised the SCADA PC, he is then able to reach the devices located in the PLC subnet, he can therefore leverage the access level he just obtained to launch three different attacks.

The first option being a dos attack on PLC 1 (node 9), by exploiting the vulnerability of the *opcuaService* (node 5).

The second being a man in the middle attack (node 10), achieved by exploiting the *impersonateDst* vulnerability exposed by the link layer ARP protocol (node 6) which allows the attacker, located at the SCADA PC, to manipulate the communication between PLC 1 and HMI 1 by impersonating each of them.

And the last being a traffic sniffing attack (node 8), through which the attacker can access the operational data flowing from PLC 1 to the HISTORIAN by exploiting the fact that nor ethernet (node 3), nor MQTT (node 4), nor the PLC itself (node 2) encrypt the sensitive data before transfer.

In regards to the extracted attack paths and the selected tasks, only the ones corresponding to the case of the attacker located in the ENTERPRISE subnet have been reported in Section A.2, and Section A.4. Since, as will be explained in the following section, the paths corresponding to the SCADA subnet case constitute a subset of the paths corresponding to the ENTERPRISE subnet case.

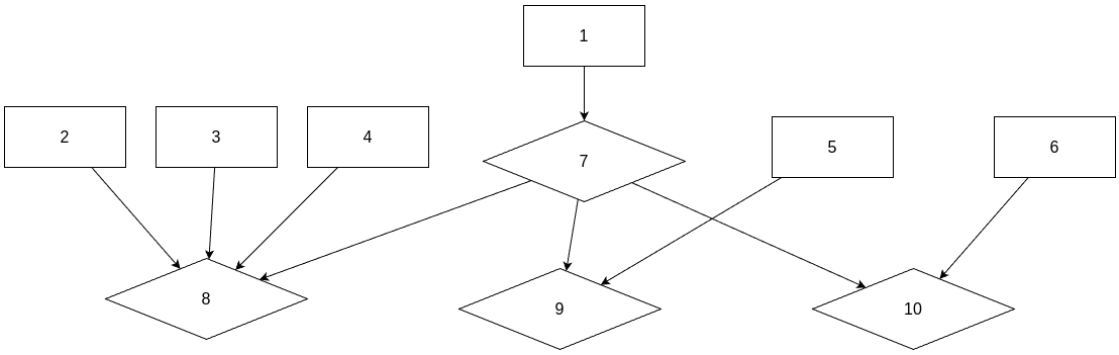

Figure 5.2: Pruned attack graph for case of attacker located in SCADA subnet

Node ID	Node label
1	vulExists(scadaPC,9,ignitionPortal,remoteExploit,privEscalation)
2	vulData(plc1Data,10,unencrypted,sniffing)
3	vulLinkProtocol(plcLan,5,ethernet,adjacent,eavesdropping)
4	vulE2EProtocol(plc1,historian,6,mqtt,1883,adjacent,eavesdropping)
5	vulExists(plc1,1,opcuaServer,remoteExploit,dos)
6	vulLinkProtocol(plcLan,4,arp,adjacent,impersonateDst)
7	execCode(scadaPC,admin)
8	accessDataFlow(attacker,plc1Data,view)
9	dos(attacker,plc1)
10	mitmLink(attacker,hmi1,plc1,scadaPC)

Table 5.1: Node labels corresponding to Figure 5.2

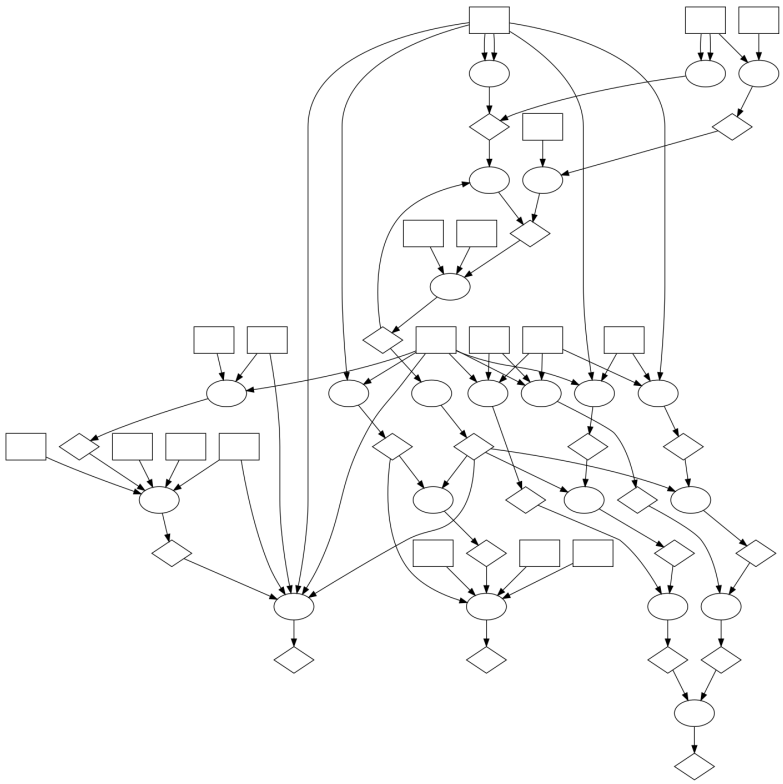

Figure 5.3: Complete attack graph for attacker located in SCADA subnet

5.2.2 Case of attacker located in the ENTERPRISE subnet

This case supposes the attacker to be located in the ENTERPRISE subnet and to have access to a *companyPC*. As opposed to the last case, this scenario is a lot more likely to happen, as it doesn't require physical access to the plant premises. The attack could coincide either with an insider attack performed by a disgruntled employee, or with an external attack performed by a malicious actor who managed to gain access to the IT system through some preliminary attack (e.g. phishing email).

After having changed the initial attacker location from EWS to *companyPC* in the attack template file, the engine produced the attack graph shown in Figure 5.4, for which Table 5.2 reports the content. This graph corresponds to the pruned version of the attack graph. Also in this case, as in the previous, the complete graph, stripped of node content, has been given in Figure 5.5 in order to provide a means of comparison with the pruned version.

By closely inspecting the pruned graph, it is possible to notice that it is exactly the same as the one discussed in the previous case with the addition of three extra nodes (node 1, node 3, and node 4) in the top part.

This is because the ENTERPRISE subnet, where the attacker is initially located, is separated from the SCADA subnet by means of a FIREWALL. Therefore in order to perform the same attacks of the previously discussed scenario, the attacker must first find a way to access the SCADA subnet. As shown by the graph, this can be done by exploiting the access control bypass vulnerability (node 1) exposed by the *customPacketFilter* service running on the *fortim* firewall. The successful exploitation of this vulnerability leads to the attacker being able to bypass traffic restrictions and reach the devices of the SCADA subnet from the *companyPC* he already has access to. This can be seen in node 3, which shows how the *companyPC* can now reach the SCADA PC of the SCADA subnet.

From this point onwards the attack scenario is exactly the same as the one of the previous case. The attacker in fact can now leverage his access to the SCADA subnet to exploit the code execution vulnerability of the SCADA PC to compromise the machine, and consequently use it as a base for launching the other three described attacks targeting the PLC subnet.

In regards to the extracted attack paths in Section A.2 and the corresponding selected tasks in Section A.4, it is possible to notice, in accordance with the results obtained by the graph, how all three of them start by exploiting the access control vulnerability and then move on to gaining privileged access to the SCADA PC by exploiting the code execution vulnerability affecting the Ignition web portal. Only after having reached this point they split into three different paths, one for each of the possible attacks that can be carried out in the PLC subnet.

By returning the attention to the pruned graph however, it is possible to notice

the presence of a particular node (node 4) corresponding to an access control bypass that allows the SCADA PC to reach itself. This is a problem that has been encountered during the generation of attack graphs by MulVAL, and it manifests itself in some of the interaction rules including a source and destination host property (e.g. *accessControlBypass*, *mitmLink*, and the like). In such cases, MulVAL doesn't enforce the condition that the considered hosts must be different, therefore this might results in the generation of meaningless nodes like the one that has just been discussed.

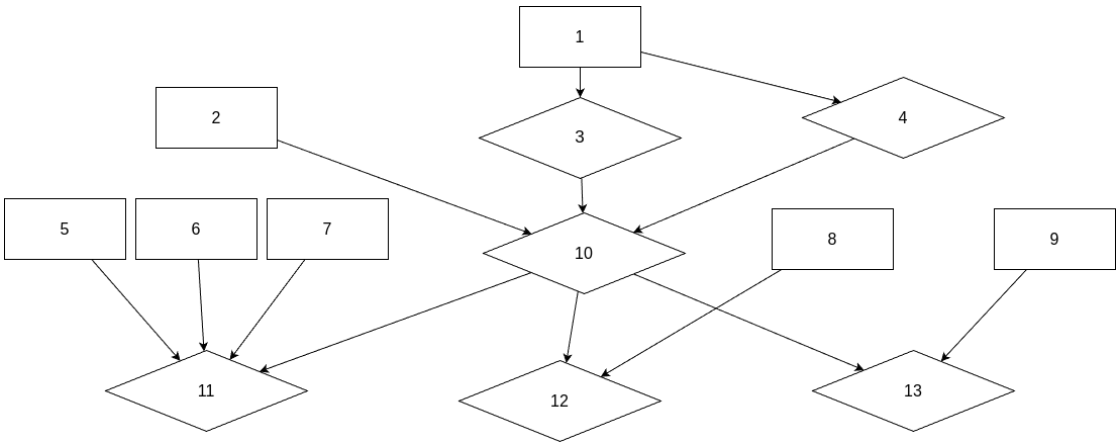

Figure 5.4: Pruned attack graph for case of attacker located in ENTERPRISE subnet

Node ID	Node label
1	vulExists(fortim,13,customPacketFilter,remoteExploit,accessControlBypass)
2	vulExists(scadaPC,9,ignitionPortal,remoteExploit,privEscalation)
3	accessControlBypass(companyPC,scadaPC,https,443)
4	accessControlBypass(scadaPC,scadaPC,https,443)
5	vulData(plc1Data,10,unencrypted,sniffing)
6	vulLinkProtocol(plcLan,5,ethernet,adjacent,eavesdropping)
7	vulE2EProtocol(plc1,historian,6,mqtt,1883,adjacent,eavesdropping)
8	vulExists(plc1,1,opcuaServer,remoteExploit,dos)
9	vulLinkProtocol(plcLan,4,arp,adjacent,impersonateDst)
10	execCode(scadaPC,admin)
11	accessDataFlow(attacker,plc1Data,view)
12	dos(attacker,plc1)
13	mitmLink(attacker,hmi1,plc1,scadaPC)

Table 5.2: Node labels corresponding to Figure 5.4

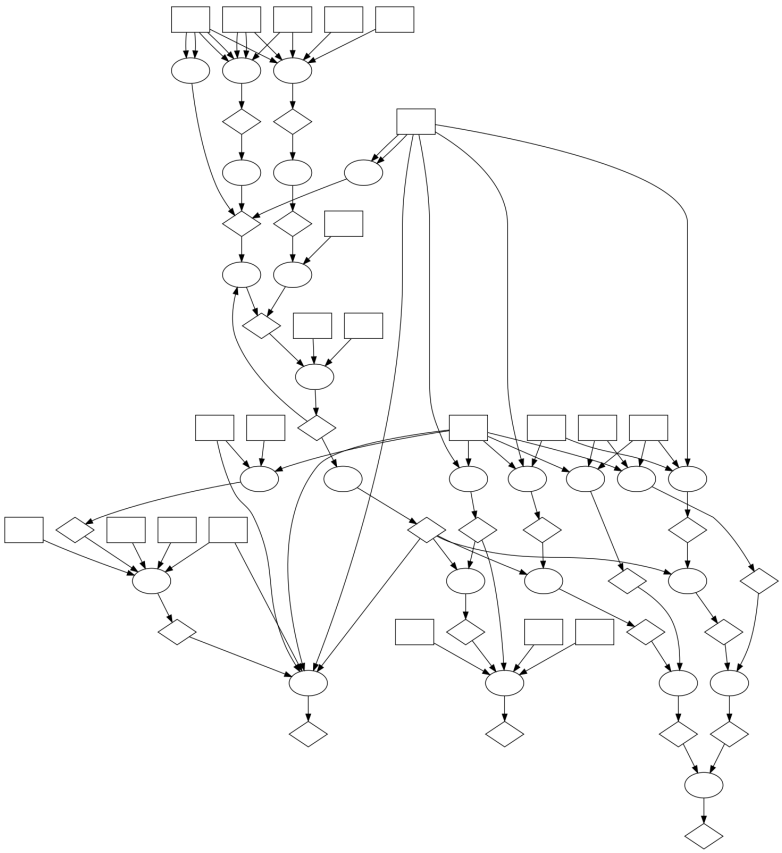

Figure 5.5: Complete attack graph for attacker located in ENTERPRISE subnet

Chapter 6

Conclusions and future work

This work attempts to address the growing concerns with the status of cybersecurity within OT infrastructures by proposing an automated solution, based on formal verification techniques, in order to lend support to the security assessment process.

It is to be said however, that the prototype engine presented in this work is by no means to be considered a stand alone solution. On the contrary, it has been designed to be easily integrated within the overall company project, as part of the automated security assessment process pipeline. To this point in fact, a standard input description model for representing network information within the project has been defined.

Within the context of the project, the main objective that this work tries to accomplish is the automatic generation of an execution plan, consisting of ordered sequences of validation tasks to be run in order to test the security of the system in need of evaluation.

In order to achieve its prescribed goal, the presented prototype engine leverages the MulVAL framework for attack graph generation, as the core part of the project. It then proceeds to build upon it by adding the following features: new network and attack modelling capabilities, most of which presented by [34], with the exception of a few customized ones; parsing functionalities for favoring compatibility with the defined standard data models; and a custom pruning algorithm for reducing attack graph complexity and increasing readability. From the results produced by the enhanced MulVAL framework, the attack paths are then extracted and subsequently the sequences of validation tasks are selected thanks to the path extractor, and task selector module described in Chapter 4.

As showcased in Chapter 5, the engine prototype is capable of correctly identifying possible attack paths, and correspondingly generating task execution plans. However, this has only been tested for two cases of a single OT network configuration that has been manually constructed to include five main types of attack modelled in the extended MulVAL framework. Going forward, it would therefore

be of great benefit to add support for more types of attack, aiming at covering the tactics and techniques of the MITRE ATT&CK framework for ICS [7] as much as possible.

Also the attack graph pruning algorithm could definitely undergo some improvement, such as adding more filtering conditions in order to remove useless nodes like the one mentioned in the example discussed in Section 5.2.2, or being parallelized in order to run more efficiently.

Since the number of possible attack paths scales with the size of the considered network, it could be worth considering to take inspiration from [44] and leverage deep reinforcement learning algorithms in order to extract the optimal options, discarding the rest.

In regards to the task selection module, the current implementation searches for validation tasks solely based on the type of attack (e.g. code execution, dos, and so on). This could be bettered by adding different filtering criteria, such as CVE code, criticality, or given attack preconditions.

All in all the solution proposed in this work represents a first concrete step towards the development of an automated tool for security assessment of OT networks, which hopefully in the future could be used a standard in the industrial scene.

Bibliography

[1] Giulio Sunder, Alberto Salvatore Colletto, Sara Raimondi, Cataldo Basile, Alessio Viticchié, and Alessandro Aliberti. «Enhancing OT Threat Modelling: An Effective Rule-Based Approach for Attack Graph Generation». In: *ICSC: Intelligent Cybersecurity Conference*. 2024 (cit. on p. 2).

[2] Keith Stouffer, Michael Pease, C Tang, Timothy Zimmerman, Victoria Pillitteri, and Suzanne Lightman. «Guide to operational technology (ot) security». In: *National Institute of Standards and Technology: Gaithersburg, MD, USA* (2022) (cit. on pp. 3, 6, 12, 13, 15, 16, 19).

[3] Michael Tiegelkamp and Karl-Heinz John. *IEC 61131-3: Programming industrial automation systems*. Vol. 166. Springer, 2010 (cit. on p. 7).

[4] Wolfgang Mahnke, Stefan-Helmut Leitner, and Matthias Damm. *OPC unified architecture*. Springer Science & Business Media, 2009 (cit. on p. 17).

[5] George Thomas. «Introduction to the modbus protocol». In: *The Extension* 9.4 (2008), pp. 1–4 (cit. on p. 17).

[6] Igor Belai and Peter Drahoš. «The industrial communication systems Profibus and PROFInet». In: *Applied Natural Sciences* 1 (2009), pp. 329–336 (cit. on p. 18).

[7] Otis Alexander, Misha Belisle, and Jacob Steele. «MITRE ATT&CK for industrial control systems: Design and philosophy». In: *The MITRE Corporation: Bedford, MA, USA* 29 (2020) (cit. on pp. 21, 85).

[8] Marie Baezner and Patrice Robin. *Stuxnet*. Tech. rep. ETH Zurich, 2017 (cit. on p. 23).

[9] Sean Collins and Stephen McCombie. «Stuxnet: the emergence of a new cyber weapon and its implications». In: *Journal of Policing, Intelligence and Counter Terrorism* 7.1 (2012), pp. 80–91 (cit. on p. 23).

[10] James P Farwell and Rafal Rohozinski. «Stuxnet and the future of cyber war». In: *Survival* 53.1 (2011), pp. 23–40 (cit. on p. 23).

[11] Aleksandr Matrosov, Eugene Rodionov, David Harley, and Juraj Malcho. «Stuxnet under the microscope». In: *ESET LLC (September 2010)* 6 (2010) (cit. on p. 23).

[12] Alessandro Di Pinto, Younes Dragoni, and Andrea Carcano. «TRITON: The first ICS cyber attack on safety instrument systems». In: *Proc. Black Hat USA* 2018 (2018), pp. 1–26 (cit. on p. 24).

[13] Cynthia Phillips and Laura Painton Swiler. «A graph-based system for network-vulnerability analysis». In: *Proceedings of the 1998 workshop on New security paradigms*. 1998, pp. 71–79 (cit. on p. 29).

[14] Shengwei Yi, Yong Peng, Qi Xiong, Ting Wang, Zhonghua Dai, Haihui Gao, Junfeng Xu, Jiteng Wang, and Lijuan Xu. «Overview on attack graph generation and visualization technology». In: *2013 International Conference on Anti-Counterfeiting, Security and Identification (ASID)*. IEEE. 2013, pp. 1–6 (cit. on p. 29).

[15] Mridul Sankar Barik, Anirban Sengupta, and Chandan Mazumdar. «Attack graph generation and analysis techniques». In: *Defence science journal* 66.6 (2016), p. 559 (cit. on p. 29).

[16] S Haque, M Keffeler, and T Atkison. «An evolutionary approach of attack graphs and attack trees: A survey of attack modeling». In: *Proceedings of the International Conference on Security and Management (SAM)*. The Steering Committee of The World Congress in Computer Science, Computer ... 2017, pp. 224–229 (cit. on p. 29).

[17] Urvashi Garg, Geeta Sikka, and Lalit K Awasthi. «A systematic review of attack graph generation and analysis techniques». In: *Computer and Cyber Security* (2018), pp. 115–146 (cit. on p. 29).

[18] Richard Paul Lippmann, Kyle William Ingols, et al. «An annotated review of past papers on attack graphs». In: (2005) (cit. on p. 29).

[19] Jin B Hong, Dong Seong Kim, Chun-Jen Chung, and Dijiang Huang. «A survey on the usability and practical applications of graphical security models». In: *Computer Science Review* 26 (2017), pp. 1–16 (cit. on p. 29).

[20] Wenhao He, Hongjiao Li, and Jinguo Li. «Unknown vulnerability risk assessment based on directed graph models: a survey». In: *IEEE Access* 7 (2019), pp. 168201–168225 (cit. on p. 29).

[21] Harjinder Singh Lallie, Kurt Debattista, and Jay Bal. «A review of attack graph and attack tree visual syntax in cyber security». In: *Computer Science Review* 35 (2020), p. 100219 (cit. on p. 29).

[22] Bruce Schneier. «Attack trees». In: *Dr. Dobb's journal* 24.12 (1999), pp. 21–29 (cit. on p. 30).

[23] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J.M. Wing. «Automated generation and analysis of attack graphs». In: *Proceedings 2002 IEEE Symposium on Security and Privacy*. 2002, pp. 273–284. DOI: 10.1109/SECPRI.2002.1004377 (cit. on p. 30).

[24] Laura P Swiler, Cynthia Phillips, and Timothy Gaylor. *A graph-based network-vulnerability analysis system*. Tech. rep. Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 1998 (cit. on p. 30).

[25] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. «Scalable, graph-based network vulnerability analysis». In: *Proceedings of the 9th ACM Conference on Computer and Communications Security*. 2002, pp. 217–224 (cit. on p. 31).

[26] Steven Noel, Sushil Jajodia, Brian O'Berry, and Michael Jacobs. «Efficient minimum-cost network hardening via exploit dependency graphs». In: *19th Annual Computer Security Applications Conference, 2003. Proceedings.* IEEE. 2003, pp. 86–95 (cit. on p. 31).

[27] Sushil Jajodia, Steven Noel, and Brian O'berry. «Topological analysis of network attack vulnerability». In: *Managing Cyber Threats: Issues, Approaches, and Challenges* (2005), pp. 247–266 (cit. on p. 32).

[28] Xinming Ou, Wayne F Boyer, and Miles A McQueen. «A scalable approach to attack graph generation». In: *Proceedings of the 13th ACM conference on Computer and communications security*. 2006, pp. 336–345 (cit. on p. 33).

[29] David Tayouri, Nick Baum, Asaf Shabtai, and Rami Puzis. «A Survey of MulVAL Extensions and Their Attack Scenarios Coverage». In: *IEEE Access* 11 (2023), pp. 27974–27991. DOI: 10.1109/ACCESS.2023.3257721 (cit. on pp. 33, 35).

[30] Xinming Ou, Sudhakar Govindavajhala, Andrew W Appel, et al. «MulVAL: A logic-based network security analyzer.» In: *USENIX security symposium*. Vol. 8. Baltimore, MD. 2005, pp. 113–128 (cit. on pp. 33, 34).

[31] Prasad Rao, Konstantinos Sagonas, Terrance Swift, David S Warren, and Juliana Freire. «XSB: A system for efficiently computing well-founded semantics». In: *Logic Programming And Nonmonotonic Reasoning: 4th International Conference, LPNMR'97 Dagstuhl Castle, Germany, July 28–31, 1997 Proceedings 4*. Springer. 1997, pp. 430–440 (cit. on p. 34).

[32] Matthew Wojcik, Tiffany Bergeron, Todd Wittbold, and Robert Roberge. *Introduction to OVAL: A new language to determine the presence of software vulnerabilities*. 2003 (cit. on p. 36).

[33] Jaime C. Acosta, Edgar Padilla, and John Homer. «Augmenting attack graphs to represent data link and network layer vulnerabilities». In: *MILCOM 2016 - 2016 IEEE Military Communications Conference*. 2016, pp. 1010–1015. DOI: 10.1109/MILCOM.2016.7795462 (cit. on p. 43).

[34] Orly Stan, Ron Bitton, Michal Ezrets, Moran Dadon, Masaki Inokuchi, Yoshinobu Ohta, Tomohiko Yagyu, Yuval Elovici, and Asaf Shabtai. «Extending Attack Graphs to Represent Cyber-Attacks in Communication Protocols and Modern IT Networks». In: *IEEE Transactions on Dependable and Secure Computing* 19.3 (2022), pp. 1936–1954. DOI: 10.1109/TDSC.2020.3041999 (cit. on pp. 43, 54–65, 84).

[35] James Tan Wee Jing, Lim Wee Yong, Dinil Mon Divakaran, and Vrizlynn L. L. Thing. «Augmenting MulVAL with automated extraction of vulnerabilities descriptions». In: *TENCON 2017 - 2017 IEEE Region 10 Conference*. 2017, pp. 476–481. DOI: 10.1109/TENCON.2017.8227911 (cit. on p. 43).

[36] Hodaya Binyamini, Ron Bitton, Masaki Inokuchi, Tomohiko Yagyu, Yuval Elovici, and Asaf Shabtai. «An automated, end-to-end framework for modeling attacks from vulnerability descriptions». In: *arXiv preprint arXiv:2008.04377* (2020) (cit. on p. 43).

[37] John Homer, Ashok Varikuti, Xinming Ou, and Miles A McQueen. «Improving attack graph visualization through data reduction and attack grouping». In: *Visualization for Computer Security: 5th International Workshop, VizSec 2008, Cambridge, MA, USA, September 15, 2008. Proceedings*. Springer. 2008, pp. 68–79 (cit. on p. 44).

[38] Mehdi Yousefi, Nhamo Mtetwa, Yan Zhang, and Huaglory Tianfield. «A novel approach for analysis of attack graph». In: *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*. IEEE. 2017, pp. 7–12 (cit. on p. 44).

[39] Steven Noel and Sushil Jajodia. «Managing attack graph complexity through visual hierarchical aggregation». In: *Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security*. VizSEC/DMSEC '04. Washington DC, USA: Association for Computing Machinery, 2004, pp. 109–118. ISBN: 1581139748. DOI: 10.1145/1029208.1029225. URL: https://doi.org/10.1145/1029208.1029225 (cit. on p. 44).

[40] Chundong Wang, Kongbo Li, Yunkun Tian, and Xiaonan He. «Network risk assessment based on improved mulVAL framework and HMM». In: *Security and Privacy in New Computing Environments: Second EAI International Conference, SPNCE 2019, Tianjin, China, April 13–14, 2019, Proceedings 2*. Springer. 2019, pp. 298–307 (cit. on p. 44).

[41] Jaka Sembiring, Mufti Ramadhan, Yudi S Gondokaryono, and Arry A Arman. «Network security risk analysis using improved MulVAL Bayesian attack graphs». In: *International Journal on Electrical Engineering and Informatics* 7.4 (2015), p. 735 (cit. on p. 44).

[42] Zhenguo HU. «Automated penetration testing using deep reinforcement learning». In: (2021) (cit. on p. 44).

[43] Ismael Jabr, Yanal Salman, Motasem Shqair, and Amjad Hawash. «Simulated Penetration Testing And Attack Automation Using Deep Reinforcement Learning». In: (2022) (cit. on p. 44).

[44] Zhenguo Hu, Razvan Beuran, and Yasuo Tan. «Automated Penetration Testing Using Deep Reinforcement Learning». In: *2020 IEEE European Symposium on Security and Privacy Workshops (EuroSPW)*. 2020, pp. 2–10. DOI: 10.1109/EuroSPW51379.2020.00010 (cit. on p. 85).

Appendix A

Supplementary material

This appendix provides supplementary material that is referenced throughout the dissertation. It includes some relevant data models that have been used during the development phase of the project, accompanied by a brief description, as the key features will be addressed and explained in more detail in the specific chapters referencing them.

A.1 JSON model of the tested network environment

This section provides the JSON data structure modelling one of the two network scenarios for which the proposed tool has been tested.

It is important to note that, for the purpose of this work, the following file was created manually, as at the time of development, the module of the tool responsible for performing automatic network reconnaissance was under construction. However, in the feature this model is expected to be generated automatically.

The network configuration that this file refers to is described in Section 5.1, and corresponds to the case scenario with the attacker placed in the company network which is separated from the internal network by a firewall enforcing access control policies.

```
{
    "subnets": [
        {
            "name": "enterpriseLan",
            "system.network.subnet.address": "10.8.3.0",
            "system.network.subnet.mask": "255.255.255.0",
            "type": "ipSubnet",
            "l2_protocols": [
```

```
            "ethernet",
            "arp"
        ],
        "hosts": [
            {
                "hostname": "fortim",
                "diana.network.host.address": "10.8.3.171",
                "device": "Fortinet Fortigate 600D",
                "vendor": "Fortinet",
                "state": "active",
                "os": "linux",
                "services": [
                    {
                        "name": "customPacketFilter",
                        "type": "remote",
                        "version": "_",
                        "port": 9001,
                        "prot": "_",
                        "state": "open",
                        "priv": "admin"
                    }
                ],
                "hacl": []
            },
            {
                "hostname": "companyPC",
                "diana.network.host.address": "10.8.3.172",
                "device": "HP compaq pro 6300",
                "vendor": "HP",
                "state": "active",
                "os": "windows 10",
                "services": [],
                "hacl": []
            }
        ]
    },
    {
        "name": "plcLan",
        "type": "ipSubnet",
        "system.network.subnet.address": "10.8.4.0",
        "system.network.subnet.mask": "255.255.255.0",
        "l2_protocols": [
            "arp",
            "ethernet"
        ],
```

```
"hosts": [
    {
        "hostname": "plc1",
        "diana.network.host.address": "10.8.4.171",
        "device": "Allen-Bradley Micro 870",
        "vendor": "Rockwell Automation",
        "state": "active",
        "os": "linux",
        "services": [
            {
                "name": "opcuaServer",
                "type": "remote",
                "version": "_",
                "port": 4840,
                "prot": "opcua",
                "state": "open",
                "priv": "admin"
            }
        ],
        "hacl": []
    },
    {
        "hostname": "plc2",
        "diana.network.host.address": "10.8.4.172",
        "device": "Allen-Bradley Micro 870",
        "vendor": "Rockwell Automation",
        "state": "active",
        "os": "linux",
        "services": [
            {
                "name": "opcuaServer",
                "type": "remote",
                "version": "_",
                "port": 4840,
                "prot": "opcua",
                "state": "open",
                "priv": "admin"
            }
        ],
        "hacl": []
    },
    {
        "hostname": "plc3",
        "diana.network.host.address": "10.8.4.173",
        "device": "Allen-Bradley Micro 870",
```

```
        "vendor": "Rockwell Automation",
        "state": "active",
        "os": "linux",
        "services": [
            {
                "name": "opcuaServer",
                "type": "remote",
                "version": "_",
                "port": 4840,
                "prot": "opcua",
                "state": "open",
                "priv": "admin"
            }
        ],
        "hacl": []
    },
    {
        "hostname": "hmi1",
        "diana.network.host.address": "10.8.4.174",
        "device": "Allen-Bradley Panel View",
        "vendor": "Rockwell Automation",
        "state": "active",
        "os": "linux",
        "services": [
            {
                "name": "sshd",
                "type": "remote",
                "version": "OpenSSH_6.6.1p1",
                "port": 22,
                "prot": "ssh",
                "state": "open",
                "priv": "admin"
            }
        ],
        "hacl": []
    },
    {
        "hostname": "hmi2",
        "diana.network.host.address": "10.8.4.175",
        "device": "Allen-Bradley Panel View",
        "vendor": "Rockwell Automation",
        "state": "active",
        "os": "linux",
        "services": [
            {
```

```
                "name": "sshd",
                "type": "remote",
                "version": "OpenSSH_6.6.1p1",
                "port": 22,
                "prot": "ssh",
                "state": "open",
                "priv": "admin"
            }
        ],
        "hacl": []
    },
    {
        "hostname": "hmi3",
        "diana.network.host.address": "10.8.4.176",
        "device": "Allen-Bradley Panel View",
        "vendor": "Rockwell Automation",
        "state": "active",
        "os": "linux",
        "services": [
            {
                "name": "sshd",
                "type": "remote",
                "version": "OpenSSH_6.6.1p1",
                "port": 22,
                "prot": "ssh",
                "state": "open",
                "priv": "admin"
            }
        ],
        "hacl": []
    },
    {
        "hostname": "scadaPC",
        "diana.network.host.address": "10.8.4.177",
        "device": "HP Workstation Z230",
        "vendor": "HP",
        "state": "active",
        "os": "linux",
        "services": [],
        "hacl": []
    },
    {
        "hostname": "historian",
        "diana.network.host.address": "10.8.4.178",
        "device": "HPE ProLiant DL380",
```

```
                "vendor": "HP",
                "state": "active",
                "os": "linux",
                "services": [
                    {
                        "name": "influxDB",
                        "type": "remote",
                        "version": "_",
                        "port": 8086,
                        "prot": "http",
                        "state": "open",
                        "priv": "admin"
                    },
                    {
                        "name": "mosquittoBroker",
                        "type": "remote",
                        "version": "_",
                        "port": 1883,
                        "prot": "mqtt",
                        "state": "open",
                        "priv": "admin"
                    }
                ],
                "hacl": []
            }
        ]
    },
    {
        "name": "scadaLan",
        "system.network.subnet.address": "10.8.5.0",
        "system.network.subnet.mask": "255.255.255.0",
        "type": "ipSubnet",
        "l2_protocols": [
            "ethernet",
            "arp"
        ],
        "hosts": [
            {
                "hostname": "fortim",
                "diana.network.host.address": "10.8.5.171",
                "device": "Fortinet Fortigate 600D",
                "vendor": "Fortinet",
                "state": "active",
                "os": "linux",
                "services": [],
```

```
        "hacl": []
    },
    {
        "hostname": "scadaPC",
        "diana.network.host.address": "10.8.5.172",
        "device": "HP Workstation Z230",
        "vendor": "HP",
        "state": "active",
        "os": "linux",
        "services": [
            {
                "name": "ignitionPortal",
                "type": "remote",
                "version": "_",
                "port": 443,
                "prot": "https",
                "state": "open",
                "priv": "admin"
            }
        ],
        "hacl": []
    },
    {
        "hostname": "ews",
        "diana.network.host.address": "10.8.5.173",
        "device": "HP compaq pro 6300",
        "vendor": "HP",
        "state": "active",
        "os": "windows 10",
        "services": [],
        "hacl": []
    },
    {
        "hostname": "historian",
        "diana.network.host.address": "10.8.5.174",
        "device": "HPE ProLiant DL380",
        "vendor": "HP",
        "state": "active",
        "os": "linux",
        "services": [
            {
                "name": "grafana",
                "type": "remote",
                "version": "_",
                "port": 3000,
```

```
                        "prot": "https",
                        "state": "open",
                        "priv": "admin"
                    }
                ],
                "hacl": []
            }
        ]
    }
],
"data_flows": [
    {
        "flow_name": "plc1Data",
        "src_host": "plc1",
        "dst_host": "historian",
        "prot": "mqtt",
        "port": 1883
    },
    {
        "flow_name": "plc2Data",
        "src_host": "plc2",
        "dst_host": "historian",
        "prot": "mqtt",
        "port": 1883
    },
    {
        "flow_name": "plc3Data",
        "src_host": "plc3",
        "dst_host": "historian",
        "prot": "mqtt",
        "port": 1883
    }
],
"vulnerabilities": [
    {
        "id": 1,
        "cve": "CVE-2022-25888",
        "type": "service",
        "host": "plc1",
        "service": "opcuaServer",
        "range": "remoteExploit",
        "consequence": "dos"
    },
    {
        "id": 2,
```

```
        "cve": "CVE-2022-25888",
        "type": "service",
        "host": "plc2",
        "service": "opcuaServer",
        "range": "remoteExploit",
        "consequence": "dos"
    },
    {
        "id": 3,
        "cve": "CVE-2022-25888",
        "type": "service",
        "host": "plc3",
        "service": "opcuaServer",
        "range": "remoteExploit",
        "consequence": "dos"
    },
    {
        "id": 4,
        "cve": "CVE-1999-0667",
        "type": "l2_protocol",
        "subnet": "plcLan",
        "l2_protocol": "arp",
        "range": "adjacent",
        "consequence": "impersonateDst"
    },
    {
        "id": 5,
        "cve": "CWE-319",
        "type": "l2_protocol",
        "subnet": "plcLan",
        "l2_protocol": "ethernet",
        "range": "adjacent",
        "consequence": "eavesdropping"
    },
    {
        "id": 6,
        "cve": "CWE-319",
        "type": "e2e_protocol",
        "src_host": "plc1",
        "dst_host": "historian",
        "e2e_protocol": "mqtt",
        "port": 1883,
        "range": "adjacent",
        "consequence": "eavesdropping"
    },
```

```
{
    "id": 7,
    "cve": "CWE-319",
    "type": "e2e_protocol",
    "src_host": "plc2",
    "dst_host": "historian",
    "e2e_protocol": "mqtt",
    "port": 1883,
    "range": "adjacent",
    "consequence": "eavesdropping"
},
{
    "id": 8,
    "cve": "CWE-319",
    "type": "e2e_protocol",
    "src_host": "plc3",
    "dst_host": "historian",
    "e2e_protocol": "mqtt",
    "port": 1883,
    "range": "adjacent",
    "consequence": "eavesdropping"
},
{
    "id": 9,
    "cve": "CVE-2023-39476",
    "type": "service",
    "host": "scadaPC",
    "service": "ignitionPortal",
    "range": "remoteExploit",
    "consequence": "privEscalation"
},
{
    "id": 10,
    "cve": "CWE-319",
    "type": "data_flow",
    "data_flow": "plc1Data",
    "vul": "unencrypted",
    "range": "",
    "consequence": "sniffing"
},
{
    "id": 11,
    "cve": "CWE-319",
    "type": "data_flow",
    "data_flow": "plc2Data",
```

```
            "vul": "unencrypted",
            "range": "",
            "consequence": "sniffing"
        },
        {
            "id": 12,
            "cve": "CWE-319",
            "type": "data_flow",
            "data_flow": "plc3Data",
            "vul": "unencrypted",
            "range": "",
            "consequence": "sniffing"
        },
        {
            "id": 13,
            "cve": "CVE-2022-40684",
            "type": "service",
            "host": "fortim",
            "service": "customPacketFilter",
            "range": "remoteExploit",
            "consequence": "accessControlBypass"
        }
    ]
}
```


A.2 JSON model of attack paths

This section provides an example of the JSON data structure used to model the attack paths extracted by the algorithm discussed in Section 4.7.

The data structure below showcases three of the possible attack paths discovered for the network scenario presented in Section 5.1, specifically for the case of the attacker located in the company network, which is analysed in Section 5.2.2.

For the sake of brevity, only three attack paths have been reported, as all the other discovered paths correspond to the same attack sequences but targeting different hosts.


```
{
    "path_1": [
        {
            "id": 26,
            "exploit": "accessControlBypass",
            "vul_ids": [13]
        },
        {
```

```
            "id": 16,
            "exploit": "execCode",
            "vul_ids": [9]
        },
        {
            "id": 1,
            "exploit": "accessDataFlow",
            "vul_ids": [10, 5, 6]
        }
    ],
    "path_2": [
        {
            "id": 26,
            "exploit": "accessControlBypass",
            "vul_ids": [13]
        },
        {
            "id": 16,
            "exploit": "execCode",
            "vul_ids": [9]
        },
        {
            "id": 62,
            "exploit": "dos",
            "vul_ids": [1]
        }
    ],
    "path_3": [
        {
            "id": 33,
            "exploit": "accessControlBypass",
            "vul_ids": [13]
        },
        {
            "id": 16,
            "exploit": "execCode",
            "vul_ids": [9]
        },
        {
            "id": 91,
            "exploit": "mitmLink",
            "vul_ids": [4]
        }
    ]
}
```

A.3 JSON model of the task collection

In this section the JSON model used to represent the task collection addressed in Section 4.8 (which describes the schema of the following structure) is provided.

For the scope of this work, only five tasks have been modelled in order to give a proof of concept of how the task selection process happens. These five tasks correspond to the main attacks illustrated in the attack modelling Section 4.4.5, which are: code execution, denial of service, man in the middle, access control bypass, and traffic sniffing.


```
[
    {
        "id": 1,
        "qualifiedName": "ignition-breaker-rce",
        "criticality": "High",
        "category": ["rce", "privEscalation"],
        "range": "remoteExploit",
        "vulnerabilities": ["CVE-2023-39476"],
        "description": "This task attemps to exploit a vulnerability that
                        affects Ignition web portal software for SCADA systems
                        (version < 4.3.1), that allows to obtain remote code
                        execution with root privileges on host",
        "preconditions": [],
        "input": [
            {
                "type": "host",
                "param_1": "address",
                "param_2": "port"
            }
        ],
        "output": "string('Passed', 'Not passed')"
    },
    {
        "id": 2,
        "qualifiedName": "opc-ua-dos",
        "criticality": "High",
        "category": ["dos"],
        "range": "remoteExploit",
        "vulnerabilities": ["CVE-2022-25888"],
        "description": "This task attemps to exploit a vulnerability that
                        affects the opcua package, allowing attackers to cause
                        denial of service on hosts running an opcua server",
        "preconditions": [],
        "input": [
            {
```

```
            "type": "host",
            "param_1": "address",
            "param_2": "port"
        }
    ],
    "output": "string('Passed', 'Not passed')"
},
{
    "id": 3,
    "qualifiedName": "arp-spoofing",
    "criticality": "High",
    "category": ["mitm", "impersonateDst", "impersonateSrc", "spoofing"],
    "range": "adjacent",
    "vulnerabilities": ["CVE-1999-0667"],
    "description": "This task attemps to exploit a vulnerability that
                    affects layer 2 arp protocol, allowing an attacker
                    to poison the ARP tables of hosts connected to the
                    same link and impersonate other hosts",
    "preconditions": [],
    "input": [
        {
            "type": "host",
            "param_1": "address",
            "param_2": "port"
        },
        {
            "type": "host",
            "param_1": "address",
            "param_2": "port"
        }
    ],
    "output": "string('Passed', 'Not passed')"
},
{
    "id": 4,
    "qualifiedName": "access-control-bypass",
    "criticality": "High",
    "category": ["accessControlBypass"],
    "range": "remoteExploit",
    "vulnerabilities": ["CVE-2022-40684"],
    "description": "This task attemps to exploit a vulnerability that
                    affects FortiProxy firewalls allowing malicious
                    attackers to bypass access restrictions",
    "preconditions": [],
    "input": [
```

```
            {
                "type": "host",
                "param_1": "address",
                "param_2": "port"
            }
        ],
        "output": "string('Passed', 'Not passed')"
    },
    {
        "id": 5,
        "qualifiedName": "eavesdropping",
        "criticality": "Medium",
        "category": ["accessDataFlow", "sniffing", "eavesdropping"],
        "range": "remoteExploit",
        "vulnerabilities": ["CWE-319"],
        "description": "This task attemps to intercept unencrpyted traffic
                        flowing through the network due to protocol
                        misconfigurations",
        "preconditions": [],
        "input": [
            {
                "type": "subnet",
                "param_1": "address",
                "param_2": "protocol"
            }
        ],
        "output": "string('Passed', 'Not passed')"
    }
]
```

A.4 JSON model of selected task plans

This section provides an example of the JSON data structure used for the output of the task selector module, for which the schema is discussed in Section 4.9. The three paths shown below correspond to the attack paths listed in Section A.2.

```
{
    "path_1": {
        "accessControlBypass": {
            "task_ids": [4],
            "vul_ids": [13]
        },
        "execCode": {
            "task_ids": [1],
```

```
            "vul_ids": [9]
        },
        "accessDataFlow": {
            "task_ids": [5],
            "vul_ids": [10, 5, 6]
        }
    },
    "path_2": {
        "accessControlBypass": {
            "task_ids": [4],
            "vul_ids": [13]
        },
        "execCode": {
            "task_ids": [1],
            "vul_ids": [9]
        },
        "dos": {
            "task_ids": [2],
            "vul_ids": [1]
        }
    },
    "path_3": {
        "accessControlBypass": {
            "task_ids": [4],
            "vul_ids": [13]
        },
        "execCode": {
            "task_ids": [1],
            "vul_ids": [9]
        },
        "mitmLink": {
            "task_ids": [3],
            "vul_ids": [4]
        }
    }
}
```