



Project Number 101017258

D5.6 Tools for Automated Security Analysis of MRS and for Production of EDDIs (Final Version)

**Version 1.0
5 July 2023
Final**

Public Distribution

FORTH

Project Partners: Aero41, ATB, AVL, Bonn-Rhein-Sieg University, Cyprus Civil Defence, Domaine Kox, FORTH, Fraunhofer IESE, KIOS, KUKA Assembly & Test, Locomotec, Luxsense, The Open Group, Technology Transfer Systems, University of Hull, University of Luxembourg, University of York

Every effort has been made to ensure that all statements and information contained herein are accurate, however the SESAME Project Partners accept no liability for any error or omission in the same.

© 2023 Copyright in this document remains vested in the SESAME Project Partners.

PROJECT PARTNER CONTACT INFORMATION

Aero41 Frédéric Hemmeler Chemin de Mornex 3 1003 Lausanne Switzerland E-mail: frederic.hemmeler@aero41.ch	ATB Sebastian Scholze Wiener Strasse 1 28359 Bremen Germany E-mail: scholze@atb-bremen.de
AVL Martin Weinzerl Hans-List-Platz 1 8020 Graz Austria E-mail: martin.weinzerl@avl.com	Bonn-Rhein-Sieg University Nico Hochgeschwender Grantham-Allee 20 53757 Sankt Augustin Germany E-mail: nico.hochgeschwender@h-brs.de
Cyprus Civil Defence Eftychia Stokkou Cyprus Ministry of Interior 1453 Lefkosia Cyprus E-mail: estokkou@cd.moi.gov.cy	Domaine Kox Corinne Kox 6 Rue des Prés 5561 Remich Luxembourg E-mail: corinne@domainekox.lu
FORTH Sotiris Ioannidis N Plastira Str 100 70013 Heraklion Greece E-mail: sotiris@ics.forth.gr	Fraunhofer IESE Daniel Schneider Fraunhofer-Platz 1 67663 Kaiserslautern Germany E-mail: daniel.schneider@iese.fraunhofer.de
KIOS Maria Michael 1 Panepistimiou Avenue 2109 Aglatzia, Nicosia Cyprus E-mail: mmichael@ucy.ac.cy	KUKA Assembly & Test Michael Laackmann Uhthoffstrasse 1 28757 Bremen Germany E-mail: michael.laackmann@kuka.com
Locomotec Sebastian Blumenthal Bergiusstrasse 15 86199 Augsburg Germany E-mail: blumenthal@locomotec.com	Luxsense Gilles Rock 85-87 Parc d'Activités 8303 Luxembourg Luxembourg E-mail: gilles.rock@luxsense.lu
The Open Group Scott Hansen Rond Point Schuman 6, 5 th Floor 1040 Brussels Belgium E-mail: s.hansen@opengroup.org	Technology Transfer Systems Paolo Pedrazzoli Via Francesco d'Ovidio, 3 20131 Milano Italy E-mail: pedrazzoli@ttsnetwork.com
University of Hull Yiannis Papadopoulos Cottingham Road Hull HU6 7TQ United Kingdom E-mail: y.i.papadopoulos@hull.ac.uk	University of Luxembourg Miguel Olivares Mendez 2 Avenue de l'Université 4365 Esch-sur-Alzette Luxembourg E-mail: miguel.olivaresmendez@uni.lu
University of York Simos Gerasimou & Nicholas Matragkas Deramore Lane York YO10 5GH United Kingdom E-mail: simos.gerasimou@york.ac.uk nicholas.matragkas@york.ac.uk	

DOCUMENT CONTROL

Version	Status	Date
0.1	Initial draft with outline and first content	12 May 2023
0.2	First draft	7 June 2023
0.3	Ready for internal review	21 June 2023
0.9	Updated version from internal reviews	4 July 2023
1.0	Final QA version	5 July 2023

TABLE OF CONTENTS

1. Introduction.....	1
1.1 Overview.....	1
1.2 Security challenge	1
2. The challenge of Security assessment	2
2.1 Defining the problem.....	2
2.2 State of the art in security assessment	3
2.2.1 Threat modelling and security assessment	3
2.2.2 Security assessment in robotic systems.....	3
2.2.3 Security knowledge repositories	4
3. The SESAME Security Methodology	5
3.1 Processes of the SESAME security methodology	5
3.1.1 System description	5
3.1.2 Identification of vulnerabilities	11
3.1.3 Identification of potential attacks.....	7
3.1.4 Identification of mitigations.....	12
3.1.5 Template Attack Trees.....	13
3.1.6 Generation of attack trees	19
3.1.7 Generation of security EDDIs.....	21
3.2 Safety and security.....	23
4. Tools for Applying Security Assessment and EDDI Production.....	25
4.1.1 System description	25
4.1.2 Identification of vulnerabilities	28
4.1.3 Identification of potential attacks.....	34
4.1.4 Generation of attack trees	39
4.1.5 Generation of security EDDIs.....	44
4.1.6 Runtime security- Intrusion Detection System	48
5. Applying SESAME methodology.....	50
6. Conclusions.....	59
7. References.....	61

TABLE OF FIGURES

Figure 1: SESAME security methodology	6
Figure 2: Discovering potential attacks from known vulnerabilities – from [50].....	12
Figure 3: False situational assessment Template Attack Tree	15
Figure 4: Publish tampered messages Template Attack Tree	16
Figure 5: Template Attack Tree with Lidar physical vulnerabilities	17
Figure 6: Template Attack Tree with Compass physical vulnerabilities	19
Figure 7: Example graph that can be produced utilizing the CanFollow relationship of CAPEC	20
Figure 8: Proposed additions for the TARA package along with their relationships with classes of the FailureLogic and FTA packages	23
Figure 9: Step -1 of system description – SESAME security methodology	26
Figure 10: Step -2 of system description – SESAME security methodology	27
Figure 11: OpenVAS web interface.....	28
Figure 12: RVD Java classes of the custom RVD parser.....	32
Figure 13: CAPEC classes of the custom CAPEC identifier	38
Figure 14: Template Attack Tree with cyber and physical vulnerabilities	43

Figure 15: Snort - example rule 49
Figure 16: Snort example output 49
Figure 17: Combined attack patterns based on the CanFollow and CanPrecede relationships 54
Figure 18: Updated version of the "publish tempered messages" Template Attack Tree 55

EXECUTIVE SUMMARY

This deliverable outlines the final version of the proposed concept and methodology for security assessment within the SESAME project. Addressing the security flaws of multi-robot systems proves to be a complex task due to factors such as increased connectivity, close proximity to humans, and a lack of awareness regarding the risks that robotic systems face.

The document presents how the state-of-the-art techniques, tools, and repositories in conducting security assessments can contribute to the definition of the SESAME security assessment concept and methodology. Furthermore, it reviews existing methodologies employed in security assessment for robotic systems, aiming to identify common patterns.

Moreover, this deliverable describes the techniques and tools that are adopted towards the successful application of the SESAME security assessment methodology. The utilized tools are designed to construct system models capable of integrating security-related information specific to a target system. These models are transformed into ODE-compliant models to facilitate the generation of runtime EDDIs.

The deliverable concludes by presenting the sequential steps of the SESAME security assessment and the corresponding tools and technologies adopted or developed for each step. Finally, an application of the proposed methodology, based on common cyber threats for the three use cases that SESAME security assessment will be integrated, is presented.

LIST OF ABBREVIATIONS

AiTB	Adversary in the Browser
AiTM	Adversary in the Middle
CAPEC	Common Attack Pattern Enumeration and Classification
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
CWE	Common Weakness Enumeration
XSS	Cross-Site Scripting
CPS	Cyber Physical Systems
DoS	Denial-of-Service
EDDI	Executable Digital Dependability Identity
IPS	Intrusion Prevention System
MX	Mail Exchange
NVD	National Vulnerability Database
ODE	Open Dependability Exchange
PUF	Physically Unclonable Function
ROS	Robot Operating System
RVD	Robot Vulnerability Database
SSI	Server Side Include
SACM	Structured Assurance Case Meta-Model
URL	Uniform Resource Locator
UAV	Unmanned Aerial Vehicle

1. INTRODUCTION

1.1 OVERVIEW

The existence of software and hardware vulnerabilities in robotic systems poses a significant risk with potentially severe consequences. Exploiting these vulnerabilities can lead to various harmful outcomes, including financial losses, exposure of sensitive data, erosion of customer trust, damage to critical assets, and even human injuries or fatalities. Given that robotic systems play an active role in numerous industry sectors such as automotive, energy (both traditional and alternative), food, pharmaceuticals, aerospace, and more, all sectors become potential targets for adversaries.

It is therefore crucial to prioritize the security of robotic systems. However, this responsibility should not fall solely on the shoulders of robot designers and operators. Standards creators, software developers, robot vendors, and security experts also play a vital role. The objective of all these roles is to make the process of exploiting robot vulnerabilities challenging and resource-intensive, ensuring that the overall security of robotic systems is strengthened.

1.2 SECURITY CHALLENGE

Modern robotic systems face a unique set of threats due to their evolving characteristics. These systems have become integral parts of our daily lives, integrated into various applications such as cars, appliances, surveillance platforms, medical equipment, and more, often operating in close proximity to humans. However, many of these systems lack built-in security mechanisms against malicious threats. Moreover, they require connectivity to the external world for monitoring and maintenance purposes, thereby introducing new attack surfaces through APIs. In addition, administrators of such systems often lack awareness of the emerging risks, as the traditional industrial robot environment was previously closed and considered trustworthy. As a result, conducting security assessments for robotic systems has become an essential yet challenging task.

The rest of the deliverable is structured as follows. In the Challenge of Security Assessment section, the definition of the problem of security threats in robotic systems is described, listing the main reasons why such systems become attack targets. Moreover, the state-of-the-art an abstract of techniques of conducting security assessment are presented. Different kinds of attacks, protection mechanisms and the most common robot specific attacks are mentioned. The threat modelling process and different threat modelling models are described. Works found in the literature that present security assessment approaches on robotic systems are referenced. The last part of this section includes security knowledge repositories that are used in the proposed methodology.

Section 3 presents the rationale for each of the steps of the SESAME security methodology. Section 4 incorporates the tools and technologies that were adopted or developed towards the individual goal of each of the steps of the SESAME security methodology. Furthermore, in section 5 an application of the proposed methodology is presented to show case its applicability. Finally, we present our concluding remarks in section 6.

2. THE CHALLENGE OF SECURITY ASSESSMENT

2.1 DEFINING THE PROBLEM

As robotic systems become more integrated into our daily lives, there is a growing concern about cybersecurity. Robots used in areas such as autonomous driving, surveillance, surgery, home assistance, and industrial automation can be vulnerable to cyber-attacks, which could have serious real-world consequences [1].

The problem encompasses another dimension related to the Robot Operating System (ROS), which serves as a standardized middle-ware for robotics. It enables the formation of diverse clusters of robots by facilitating communication among the robots within the cluster [2]. While the widespread acceptance of ROS can be attributed to its notable advantages, including an engaged community and the ability to reuse code, it also brings to light certain drawbacks such as concerns regarding network security, authorization, and resource permissions. The work in [3] presents several vulnerabilities, including communications in plain-text and unprotected TCP ports.

Successor to ROS, ROS2 incorporates important security related improvements. It offers secure communication through the integration of the Data Distribution Service (DDS) standard, implements a more robust access control system called "ROS2 Security", separates configuration files from the core codebase easing secure customization, and introduces proper dependency management. Moreover, ROS2 community actively monitors and addresses security issues, releasing updates and patches promptly. While ROS2 addresses many common security concerns, it is essential to follow security best practices and consider the broader security aspects, as far as developing and deploying robotic systems is concerned.

The necessity of evaluating the security of robotic systems is emphasized in reference [4], which presents several observations made by the authors regarding industrial robots. The first observation highlights the growing interconnectedness of robotic systems, leading to the expansion of potential attack points. Previously, industrial robots operated in isolated environments under strict control. However, with their integration into information and communication technology (ICT) ecosystems, they are now connected to external networks, including the Internet. This connectivity of industrial robots serves purposes such as control, monitoring, and maintenance, and is even incorporated into ISO standards for the integration of robot systems [5]. Furthermore, there is a trend towards developing robot application programming interfaces (APIs) that provide endpoints for user-defined requests, enabling control of the robots. Additionally, robots can be managed and supervised using portable devices like smartphones [6].

Furthermore, a prevailing tendency is observed in the adoption of safety mechanisms, where programs and libraries are being prioritized over hardware-based solutions employed in the past. This shift in implementation introduces a heightened vulnerability to potential security incidents. Compounded with the emergence of next-generation industrial robots designed to work in close proximity to humans, the scope of security attacks on robotic systems expands significantly, posing a direct threat to human safety.

Another notable observation relates to the inadequate recognition of risks faced by robotic systems. In reference [4], the authors conducted a survey that revealed

concerning findings. Some of the key results indicated that: i) a significant portion of the survey respondents (60%) modify default safety measures, thereby introducing limitations; ii) access control measures are not implemented for robots and robot-controllers among 28% of the respondents; and iii) a substantial majority (76%) do not utilize security assessment as a means of enhancing security.

The historical practice of providing services through industrial robots in closed and trusted environments appears to have led robot manufacturers to overlook essential security mechanisms [7].

The security challenge intensifies in distributed multi-robot systems (MRSs). In such configurations, if one robot is targeted in an attack, it has the potential to impact other robots or even the entire system. The compromised robot can act as a malicious entity, commonly referred to as a "bad bot," carrying out automated tasks according to the adversary's intentions, initiating attacks on other components of the system, including robots and robot-controllers. An illustrative incident described in reference [8] involves 100 drones crashing into a building during a light show in Chongqing, China. The root of the problem originated from the control system's mainframe [9].

The evaluation of security in robotic systems, which involves the identification, assessment, and mitigation of security risks to ensure compliance with system security requirements, appears to be essential. This process encompasses the recognition of valuable assets and potential vulnerabilities, the identification of threats capable of compromising those assets, and the exploration of protective measures while considering the calculated level of risk.

2.2 STATE OF THE ART IN SECURITY ASSESSMENT

2.2.1 Threat modelling and security assessment

Ensuring security in robotic systems requires a holistic approach that considers the overall system design. It is crucial to address security concerns early in the system design process. Threat modelling plays a vital role in identifying, communicating, and understanding potential threats to the system, enabling the definition of countermeasures to mitigate their effects. Threat modelling involves investigating the system from an adversary's perspective, determining what needs to be protected and from whom. The process includes steps such as system description, architecture dataflow, identification of trust boundaries, threat analysis, and determination of countermeasures. Different threat modelling methods, such as STRIDE, PASTA, LINDDUN, and CVSS, offer distinct approaches and perspectives in assessing and addressing security risks. Several open-source threat modelling tools, including Cairis, Microsoft Threat Modelling Tool, OWASP Threat Dragon, Threagile, and Tutamantic, facilitate the implementation of threat modelling processes.

2.2.2 Security assessment in robotic systems

Robots are becoming increasingly prevalent in various aspects of daily life, such as transportation, surveillance systems, home assistance, and medical services. However, the integration of different sensors, actuators, interfaces, and information processing in robots introduces new vulnerabilities that can be exploited, leading to economic damage and safety issues. Several works have explored the security analysis of robotic systems to identify cyber-attacks and their impacts. For example, one study [3] employed a

cyber-physical honeypot using ROS (Robot Operating System) to discover vulnerabilities and exploits, while another [10] demonstrated hacking a modern automobile and compromising its digital dash, door locks, brakes, and engine control components. Attacks on unmanned aerial vehicles (UAVs) were also investigated [40] revealing the impact of denial-of-service attacks on UAV cameras and network latency. Furthermore, a model was introduced to represent the performance of multi-robot systems [12], highlighting the potential for denial-of-service attacks to compromise cloud-robotic platforms.

In terms of security assessment methodologies, researchers have focused on evaluating the security of specific robotic systems. One study conducted a security assessment of Pepper [13], a social robot, through automated and manual phases involving port scanning, vulnerability scanning, traffic analysis, and brute force attacks. Flaws were identified that could enable credentials spoofing, data theft, and hacking of connected devices. A similar assessment was performed on the Franka Emika Panda robot [7], uncovering vulnerabilities in its web application that could affect human safety and the manufacturing process. Additionally, an analysis of security issues in cyber-physical systems (CPS) [14] highlighted the importance of securing sensors, transmission, and application layers, with suggested solutions including Physically Unclonable Functions (PUF) for unique identification and identity-based encryption for privacy protection. Finally, an experimental security analysis of an industrial robot controller [4] identified potential attacker goals, access points, and capabilities, along with specific attack classes, demonstrating the feasibility of attacks on a reference robot.

Overall, these works emphasize the need for robust security measures in robotic systems to address the vulnerabilities arising from their integration of various technologies and ensure the safety and reliability of these systems in our daily lives.

2.2.3 Security knowledge repositories

There are many repositories, lists, and directories that enclose information about vulnerabilities, weaknesses, bugs, etc. All these security knowledge repositories are vital resources for the SESAME security assessment process.

The security knowledge repositories mentioned in the text are vital resources for understanding and addressing vulnerabilities, weaknesses, bugs, and attack patterns. The Common Vulnerabilities and Exposures (CVE) [15] list provides identifiers for computer security flaws and vulnerabilities, allowing for easy recognition and communication. The National Vulnerability Database (NVD) [16] supplements CVE by offering additional information such as severity scores, countermeasures, and affected software configurations. The Common Weakness Enumeration (CWE) [17] provides a comprehensive list of weaknesses in software and hardware, including detailed descriptions and relationships with other weaknesses. The Common Attack Pattern Enumeration and Classification (CAPEC) [18] serves as a hierarchical classification and dictionary of known attack patterns, facilitating the understanding of how system weaknesses can be exploited.

Additionally, the Robot Vulnerability Database (RVD) specifically focuses on vulnerabilities and bugs related to robots' software and hardware. It offers a centralized repository for categorizing and recording these flaws, providing researchers and practitioners with valuable information to assess and mitigate robot-related security

issues. The Robot Vulnerability Scoring System (RVSS) is used to rate the vulnerabilities included in the RVD, aiding in the prioritization and management of robot security concerns. Overall, these repositories play a crucial role in enhancing security practices by providing valuable insights and resources for vulnerability detection, assessment, and mitigation.

3. THE SESAME SECURITY METHODOLOGY

3.1 PROCESSES OF THE SESAME SECURITY METHODOLOGY

Threat modelling process plays a crucial role in defining the security design and selecting appropriate security technologies for a system, considering its specific security requirements. The security assessment conducted within the context of SESAME is heavily influenced by the threat modelling process and adopts its fundamental principles. The SESAME security assessment follows a well-structured set of steps that often overlap with the systematic process of threat modelling, which has clearly defined steps based on the chosen model. The methodology diagram in Figure 1 illustrates these steps, including their inputs, outputs, external resources, and processes.

While the high-level steps outlined in the following sections provide a general approach, the key to effectively applying the proposed methodology to individual MRSs with unique requirements lies in providing a detailed description of the specific system under consideration. Factors such as the importance of assets and the delineation of trust boundaries containing these assets help to capture the distinct security requirements of each system. Additionally, the identification of vulnerabilities and their combinations contribute to each system being a unique use case, resulting in varying outputs from the SESAME security methodology, such as potential attack scenarios and corresponding mitigations.

3.1.1 System description

The initial stage of the SESAME security assessment involves identifying the target system. It is important to gather a comprehensive description of key aspects such as the system's objectives, its components, and its architecture. This information allows for a thorough understanding of the system's functionalities and the potential threats that may arise.

To achieve this, system administrators are required to provide a detailed system description and respond to a series of questions. The collected information serves as a foundation for identifying both cyber and physical vulnerabilities within the system, as well as potential attacks that could exploit these vulnerabilities. The desired information is organized into categories (Purpose, Components, Architecture, and Scope) and further elaborated upon in the subsequent sub-sections.

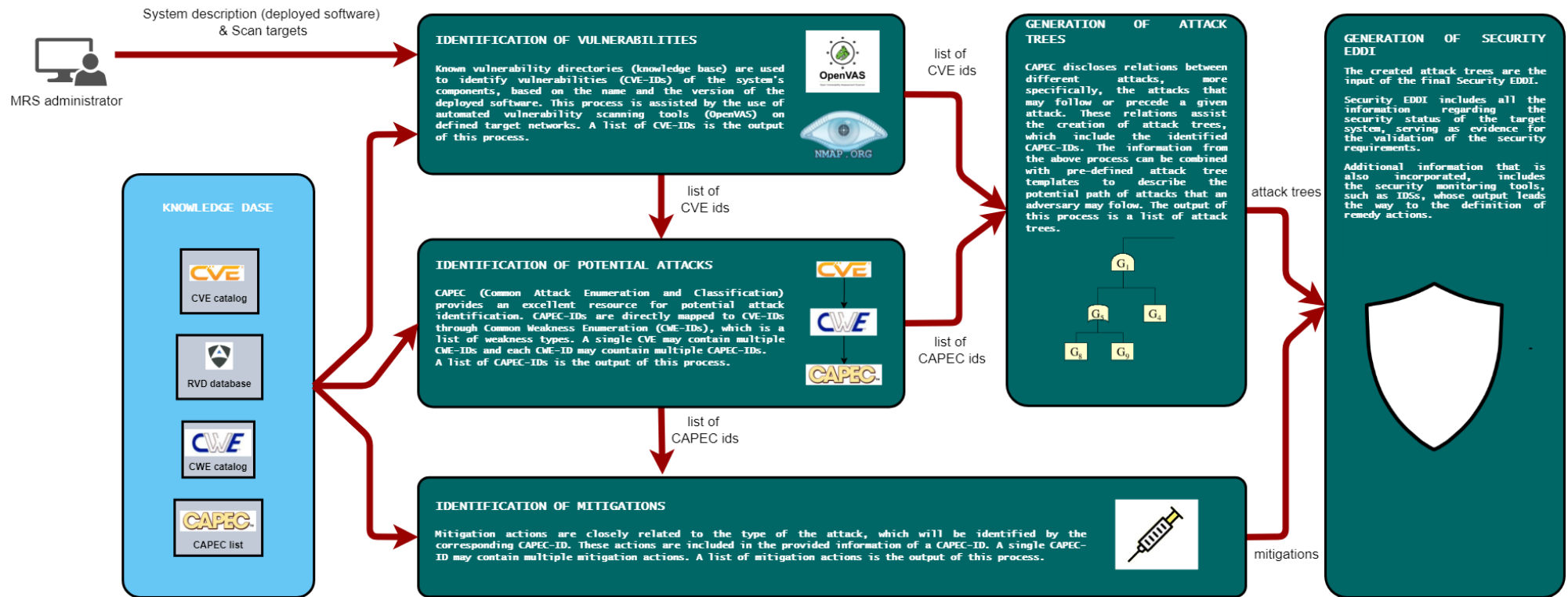


Figure 1: SESAME security methodology

3.1.1.1 Purpose

Having a clear understanding of the purpose and usage of the system is crucial for assessing its security. This information helps determine the criticality of potential attacks and the overall security level of the system under specific attack scenarios. The following questions fall under this category:

1. What is the primary function of the system? Does it perform a single task or multiple tasks?
2. Is the system critical to the organization's operations?
3. Does the system serve a specific business goal?
4. What would be the impact of system unavailability?
5. Are there any compliance requirements associated with the system?
6. Who are the users of the system and what roles do they have?
7. Are there any known system vulnerabilities?
8. Which parts of the system are considered the most critical?

Answers to these questions give insights to the security assessment regarding the system's significance and its potential risks.

If a SESAME safety methodology has been followed, introduced and described in D4.5 “Safety Analysis Concept and Methodology for EDDI development (Final Version)”, some of the requested information -- such as the safety-critical effects of system unavailability -- will already be available and stored in EDDIs, which can be used to inform this assessment.

3.1.1.2 Components

Identifying each system component is essential as vulnerabilities within these components can pose risks to the overall system. Any vulnerability in a component can potentially expose attack surfaces and compromise the system's security. These components can include software, hardware, networking infrastructure, and databases. The process of identifying potential cyber-attacks differs slightly from that of physical attacks, so the desired information is gathered in two distinct ways, as described below.

Cyber vulnerabilities: When gathering information about cyber vulnerabilities in each system component, the desired information includes the following details:

1. *Component Host.* The host of the component refers to the system or infrastructure where the component is deployed. Understanding the host is important because if a specific host is compromised due to an attack, it can impact the availability and security of all the components hosted on it. Moreover, a vulnerability of a not so crucial component becomes more critical, if the vulnerable component is cohosted with a component of great importance.

2. *Name and Version.* Different versions of a component may have different vulnerabilities, and newer versions may address or introduce vulnerabilities. Different product versions may have varying levels of vulnerability, as security flaws can be discovered and addressed over time.
3. *Vendor.* The vendor refers to the entity or organization that develops or provides the component. Vendor name could help tracking vendor-specific security information.
4. *Common Platform Enumeration (CPE).* CPE is a standardized naming format hosted and maintained by the National Institute of Standards and Technology (NIST). It provides a unique identifier for software system components, including information technology systems, software, and packages. CPE can be used for easier identification of known vulnerabilities and automation of the whole security assessment process.

Table 1 below presents a snippet of software system components description, with a column for every needed information. It presents software running in two different hosts (Raspberry Pi and Pixhawk) of the target system.

Table 1: Description of system components organized in columns

Host	Software name	Software version	Software vendor	Software CPE
Raspberry Pi 4	arca Trusted OS	1.0.0	CYSEC	NA
	python	3.10.4	Python	cpe:2.3:a:python:python:3.10.4:*:*:*:*:*:*
	docker	20.10.15-ce	Docker	cpe:2.3:a:docker:docker:20.10.15:*:*:*:*:*
	docker-compose	1.29.2	Docker	NA
Pixhawk 6C	PX4	1.13.2	PX4	NA
	nuttX	10.10.0	Apache	cpe:2.3:a:apache:nuttx:10.0.0:*:*:*:*
...

Cyber vulnerabilities are the input for the *Identification of potential attacks* process, where they are coupled with attacks that can be conducted, exploiting each of these vulnerabilities. The discovered cyber-attacks are then compared with attacks mentioned in the Template Attack Trees for the definition of the final potential attack trees of the system in question. Template Attack Trees provide a structured framework for modeling potential attack scenarios and capturing the various steps an attacker may take to compromise the system's security. All these processes are described later in the document.

Physical vulnerabilities: To identify physical vulnerabilities and potential physical attacks in the system, additional information about the individual robots and the overall system hardware is required. While the previous process focused on cyber vulnerabilities and cyber-attacks, physical vulnerabilities and physical attacks pose a different set of challenges that need to be addressed.

Robotic systems are particularly susceptible to physical attacks due to their use of sensors, motors, and various physical surfaces. Additionally, robots often operate in close proximity to humans, either sharing the same workspace or collaborating closely to achieve specific tasks, such as in automotive manufacturing. While this collaboration can enhance productivity and precision, it also increases the feasibility of physical attacks and introduces safety risks. If a robot is compromised, it has the potential to harm human operators and others nearby.

Physical attacks can take various forms, including tampering/vandalism, theft, physical operation disruption, physical intrusion, and physical manipulation. Tampering/vandalism involves actions such as breaking or removing parts of the robot or altering its environment, which can impact its functionality. Theft entails the removal of the entire robot or essential components like batteries and sensors. Physical operation disruption may involve blocking the robot's path or interfering with the movement of its components, such as arms or rotors. Physical intrusion occurs when an unauthorized person gains physical access to the robot. Lastly, physical manipulation occurs when an attacker intentionally alters the behavior or performance of the robot.

There are some system scope questions that can be asked to gather information for potential physical attacks:

1. Are there any weaknesses in the premises' physical security?
2. Is there unauthorized access to your system?
3. Are there security measures for detecting physical intrusions?
4. Are there physical assets that can be stolen or damaged?

However, robot scope questions should be asked for every vulnerable robot:

1. Does any individual share the same space with the robot during its operation or while it is idle?
2. Does the robot have any parts that can be easily removed?
3. Is there direct access to the internals of the robot? Is any essential part exposed?
4. Do the robot components come from an authorized dealer?
5. What kind of sensors does the robot carry?
6. Are there any exposed ports on the robot?

This kind of questions can reveal the physical vulnerabilities of the system and its individual components, and help to identify potential physical attacks. Physical

vulnerabilities are coupled with physical attacks as cyber vulnerabilities are coupled to cyber-attacks. These physical attacks are compared with attacks of the physical attack trees of the Template Attack Tree repository. A match could identify one of the Template Attack Trees as a potential attack tree of a given system. Once again, there is a link with the safety assessment of a target robotic system. If a physical attack damages or impairs a component, that would often count as a failure and may have safety implications. Given that attack trees and fault trees are compatible, this link may arise naturally if the root of a physical attack tree is used as input to the corresponding fault tree for that component or hazard.

3.1.1.3 Architecture

Understanding the system architecture is crucial for a comprehensive security assessment. Architecture involves the overall design, communication channels, access points, and data flow within the system. This information may reveal additional attacks, or even attack patterns, to those discovered through the identification of the known vulnerabilities.

Defining the communication protocols that are used among the system components, allow the SESAME security assessment to reveal vulnerabilities that are not present in any individual component. This is especially relevant in the context of multi-robot systems where communication plays a vital role. The communication among the robots allows them to create swarms and work together, and at the same time, it introduces new vulnerabilities and new attack surfaces. Questions that towards understanding of system architecture include the following:

1. What is the overall design of the system?
2. How are the system components connected?
3. Which are the system access points?
4. What is the path that data follow? What is the input and the output of the system?
5. Are there any third-party integrations to the system?
6. Is the system monitored?

3.1.1.4 Scope

Defining the desired scope of the security assessment determines the extent of the analysis. The SESAME security assessment can be conducted on various levels, ranging from the entire target system to specific subsystems or individual components, depending on specific security requirements. Factors that may influence the security assessment include the system's complexity, the level of potential risk, the available resources, and the specific requirements or regulations governing the system.

Questions that fit this category include the following:

1. What are the boundaries of the system to be assessed?
2. What is the acceptable security level for the system to be assessed?

3. Are there any compliance requirements for the system to be assessed?

3.1.2 Identification of vulnerabilities

By gathering all the information provided by the system administrator, the deployed programs, libraries, and services along with the used hardware are pinpointed. This is the input for the next process of SESAME security assessment, the *Identification of vulnerabilities*.

During this process, publicly available repositories with known vulnerabilities are utilized, defining the system vulnerabilities. The CVE catalogue, mentioned in subsection 2.2.3, is the main repository, used by the security assessment solution that has been developing during the lifetime of the SESAME project. Nevertheless, given that our target systems are MRSs, the RVD directory is considered a necessary addition, which focuses on bugs, weaknesses, and vulnerabilities specifically related to robots. In that way, we envision to address the unique complexities and characteristics of robots, which may not be adequately covered in other general vulnerability lists. RVD aims to enhance vulnerability disclosure by providing robotics-specific information [19].

A parser is utilized to search these vulnerability directories, leveraging the name and version of each software component present in the system, in order to identify any associated vulnerabilities. Both vulnerability directories assign a unique identifier to each documented vulnerability. This identifier is called CVE identifier (CVE-ID) for the CVE list and ID for the RVD database. The output of *Identification of vulnerabilities* process is a list of vulnerability identifiers corresponding to the vulnerabilities that are considered relevant to the target system. The vulnerability directories are regularly updated with information about newly discovered vulnerabilities, making the *Identification of vulnerabilities* process dynamic, since it remains synchronized with the updated directories, ensuring the inclusion of the latest vulnerability information.

The process that was just described is also offered as functionality by a set of automated tools, called vulnerability scanners. Following the same principle, they scan a given network and/or subnetworks for available services and then use open vulnerability databases to discover known vulnerabilities. OpenVAS, OPENSCAP, OWASP ZAP are some of the open-source options. For the sake of completeness, the SESAME security assessment includes the use of such scanning tools, since the provider of the system information may not be aware of some services that are running in devices, which are part of the system, and have some known vulnerabilities. Of course, the prerequisite in this case is that the system must be up and running, otherwise the vulnerability scanning tools cannot produce an output, meaning that these automated scanning tools cannot be used during the design phase of a system

3.1.3 Identification of potential attacks

As it is already mentioned, the output of *Identification of vulnerabilities* process is a list of vulnerability identifiers that is used as input for the next process in line, the *Identification of potential attacks*. Additional input includes the CWE catalog, a list of software and hardware weakness types, and CAPEC, a dictionary of identifiers for attack patterns, both mentioned in 2.2.3. CWE plays the role of common language for security tools. Due to the wider acceptance of CWE it is used as a stepping stone between the spotted vulnerabilities and the potential attacks. Figure 2 depicts the route

from the pinpointed vulnerabilities of a target system to the information of the corresponding potential attacks, showing how all the aforementioned directories are connected with each other.

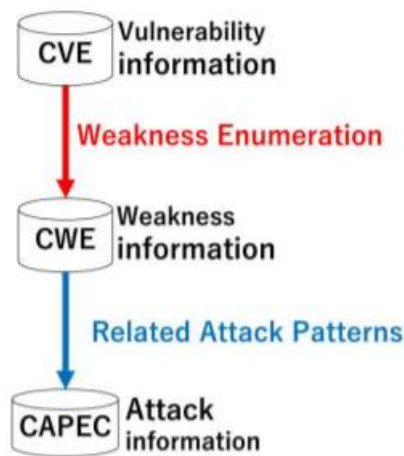


Figure 2: Discovering potential attacks from known vulnerabilities – from [20]

CWE is the connection point between CVE and CAPEC. “Weakness Enumeration” is one of the fields in the description of a given vulnerability. In this field a list of all the weakness types, in the form of CWE-IDs, that are related with the specific vulnerability is provided. Moreover, the description of every weakness type (CWE-ID) includes a field called “Related Attack Patterns”, presenting the attack patterns (CAPEC-ID) used for the exploitation of the corresponding weakness. In this way, it is possible to trace a list of CAPEC-IDs from a single CVE-ID. This process is repeated for each of the identified system vulnerabilities, meaning that the output of the *Identification of potential attacks* process is a number of different sets of CAPEC-IDs, one for each discovered vulnerability identifier.

Information related to a CAPEC-ID that is highly valuable to the *Identification of potential attacks* process, includes a description in natural language, relationship with other attacks, prerequisites for the attack to be performed, and mitigation actions.

3.1.4 Identification of mitigations

The hierarchical classification of CAPEC includes the category, meta, standard, and detailed attack levels. Documented attacks in the last two levels (standard and detailed) include, in their majority, mitigation actions. More specifically, especially for the detailed level, a very specific protection mechanism is required to mitigate the actual attacks and this mechanism is mentioned under the *Mitigations* section. During the *Identification of mitigations* process the information under *Mitigation* tab is collected for every defined CAPEC-ID. If the CAPEC-ID in question is of standard and detailed level, the information is directly available. On the other hand, a meta level attack pattern is more abstract, avoiding information about specific methodologies, techniques, implementations and protection mechanisms. Said attack pattern serves as generalization of a more well-defined group of standard level attack patterns. In such a case, mitigations that are mentioned in the corresponding standard level attack patterns will be utilized for gathering the mitigation actions.

3.1.5 Template Attack Trees

The SESAME security assessment methodology relies on specific security knowledge repositories that store valuable information related to system components vulnerabilities, software and hardware weaknesses, and various types of attacks. These repositories serve as a centralized source of knowledge and are essential for the methodology's effectiveness. By leveraging the information contained within these repositories, the methodology is able to establish connections between vulnerabilities and potential attacks. This enables the identification of potential attack scenarios that could be targeted against a specific system based on its vulnerabilities.

SESAME security assessment methodology incorporates an additional security knowledge repository called Template Attack Trees, which are essentially predefined attack patterns. These attack patterns describe well-known methods employed by malicious attackers to exploit known vulnerabilities or weaknesses in system components, with the intention of achieving their objectives. Each attack pattern outlines a series of steps that an attacker can follow to accomplish their ultimate goal, which may involve compromising a system host, gaining unauthorized access to sensitive data, or disrupting system operations. The Template Attack Trees repository used in SESAME is not a pre-existing resource. Instead, it is created specifically for each individual system based on its unique characteristics and security requirements. This customized approach allows the methodology to align closely with the specific use case and system being assessed, ensuring that the identified attack patterns are relevant and tailored to the system under evaluation.

An attack tree is a graphical representation of a hierarchical structure that can incorporate a set of different attack scenarios or attack steps of an attacker towards their ultimate objective. In that sense, an attack tree can be used to represent an attack pattern. Template Attack Trees are considered as attack patterns and can be very helpful to security experts since they reveal common attack methods and techniques and ease the development of corresponding mitigation actions. In that way, the security exposure and the possibility of successful attacks can be reduced.

Template Attack Trees are used in *Generation of attack trees* process, described in the next subsection. The created repository of Template Attack Trees includes a number of Templates with specific attacks (CAPEC-IDs) at their leaves, the ultimate goal at the root of the trees, and sub-goals in between. These sub-goals describe achievements of the attacker that bring them closer to their goal. Such goals could include the following:

- Control the movement of the robot
- Make the robot unresponsive (loss of availability)
- Steal/Change sensitive information (loss of integrity)
- Make a robot not to achieve a business goal

Once again, there is a separation as far as the cyber and physical system vulnerabilities are concerned, as it is described below.

Cyber vulnerabilities: The completion of the *Identification of potential attacks* process, described previously, provides us with a number of different sets of CAPEC-IDs, one for each of the system cyber vulnerabilities. These CAPEC-IDs, which are the attacks that potentially could be conducted against our system, are compared with the CAPECs in the Template Attack Trees' leaves. A matching CAPEC-ID, which through one of the tree paths, could lead us to the ultimate goal of the tree's root, is enough to characterize the given Template as a potential attack tree of the target system. Potential attack trees include at least one attack scenario, according to which, an attacker could take advantage of a known cyber vulnerability of the target system and conduct an attack that will lead them to achieve their objective. Such known vulnerabilities of products are available in repositories such as CVE. What follows are two examples of Template Attack Trees.

- *False situational assessment Template Attack Tree*

Figure 3 depicts a Template Attack Tree with three potential attacks at its leaves. These attacks can be conducted due to system known vulnerabilities of the antennas that are used as infrastructure for the establishment of Wi-Fi networks. Such means of communication is present in SESAME use cases.

The first two are known attacks, documented in the CAPEC repository. The first has CAPEC-ID 126 and title "Path traversal". This is an attack according to which an attacker uses path manipulation methods to obtain access to data that should not, normally, be retrievable by well-formed requests. The result of such an attack is the adversary being able to steal information or manipulate sensitive files. The second vulnerability's ID is 76 and its title is "Manipulating Web Input to File System Calls". This is a quite similar attack, where the attacker, once again, manipulates inputs to the target software. That input is then passed to the file system calls in the target operating system. In that way, the attacker is able to access and even modify areas of the file system that should not be accessible.

Either of these attacks can lead to a security state where the attacker gains root access of the Wi-Fi access point. By doing so, the attacker is able then to alter the path of data that are exchanged among the different components of a given system. A more specific example that applies in the KIOS use case is the adversary routing the video that is captured by the surveillance drones to an alternative destination and not the ground control station.

According to the third attack that is described in the Template Attack Tree, an attacker is able to send malicious traffic to the ground control station. This is a more sophisticated attacks with the precondition that the attacker has knowledge of the structure of the messages that are exchanged and is able to contrast and send manipulated traffic to the destination.

The combination of the "Attacker reroutes the traffic from the drone to an alternative destination" state and the last described attack can lead to another security state, where the operators of the ground control station receive manipulated data (video) and end up with a false situational assessment for the observed area. In the context of the KIOS use case, this means that the operators may not be able to spot a trapped person under a building ruins after a catastrophic earthquake. "False

situational assessment” state is the ultimate goal of an attacker in the False situational assessment Template Attack Tree.

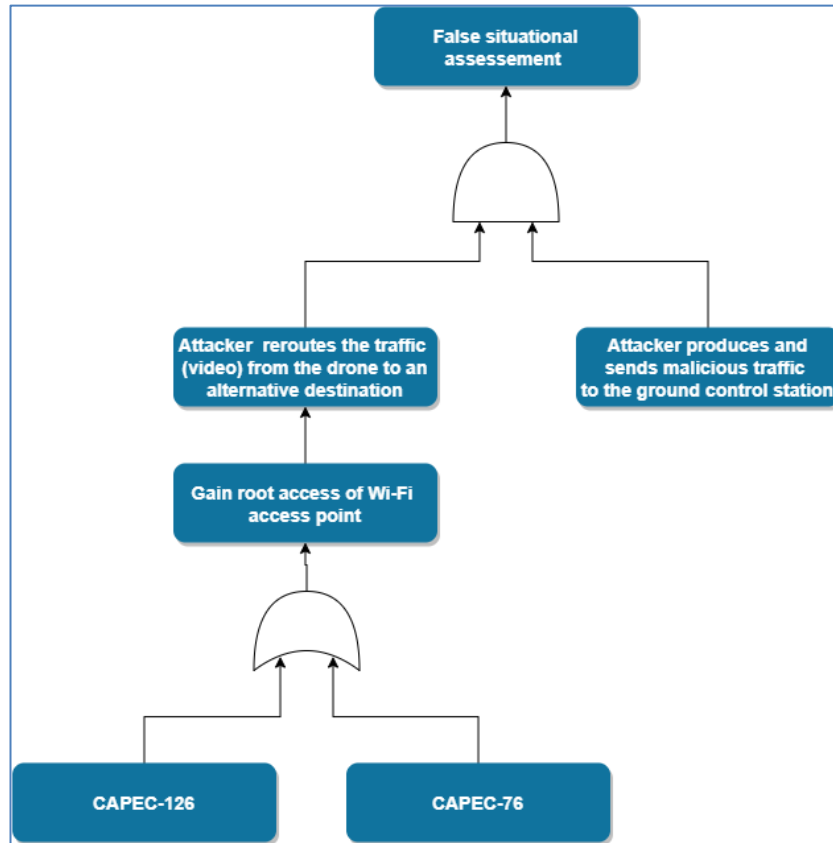


Figure 3: False situational assessment Template Attack Tree

- *Publish tampered messages Template Attack Tree*

Figure 4 depicts a Template Attack Tree where the ultimate goal of the adversary is publishing malicious messages to communication topics, and three known attacks reside at its leaves. All three attacks can be found in the CAPEC repository.

The first known attack has CAPEC-ID 94 and title “Adversary in the Middle (AiTM)”. The attacker places themselves within the communication channel between two components and tries to change the transmitted data. The communicated data flow through the attacker, who has the opportunity to observe and even alter it. The prerequisite here is the adversary being able to understand the nature of the communication between the targeted components. If this attack can be combined with a compromised robot, used by the attacker for publishing messages to individual topics, we are led to a security state where the attacker is able to publish its own tampered messages using a message queue command line interface.

The second known attack has CAPEC-ID 8 and title “Buffer Overflow in an API Call”. According to this attack, the adversary targets software that makes use of libraries, which are vulnerable to buffer overflow attacks. This software become also vulnerable by association. The third attack had CAPEC-ID 63, title “Cross-Site Scripting (XSS)” and is used by attackers that want to embed malicious scripts in content that will be served to web browsers. In that way, the target executes the

script with the user's privilege level. Any of these two attacks can lead to a security state where the attacker compromises the robot's API.

Either the "Attacker uses a message queue command line interface" or the "Attacker compromises robot's API" security state can lead to the ultimate goal of the attacker, which is to "Publish tampered messages to communication topic to change the robot trajectory". Message queue topics are a common way for SESAME use cases for the control station to communicate the trajectories to a robot. If an attacker manages to publish tampered messages to such topics, the trajectory of one or more robots could be altered, causing the affected robots to crash, intervene with other robots or cause safety issues.

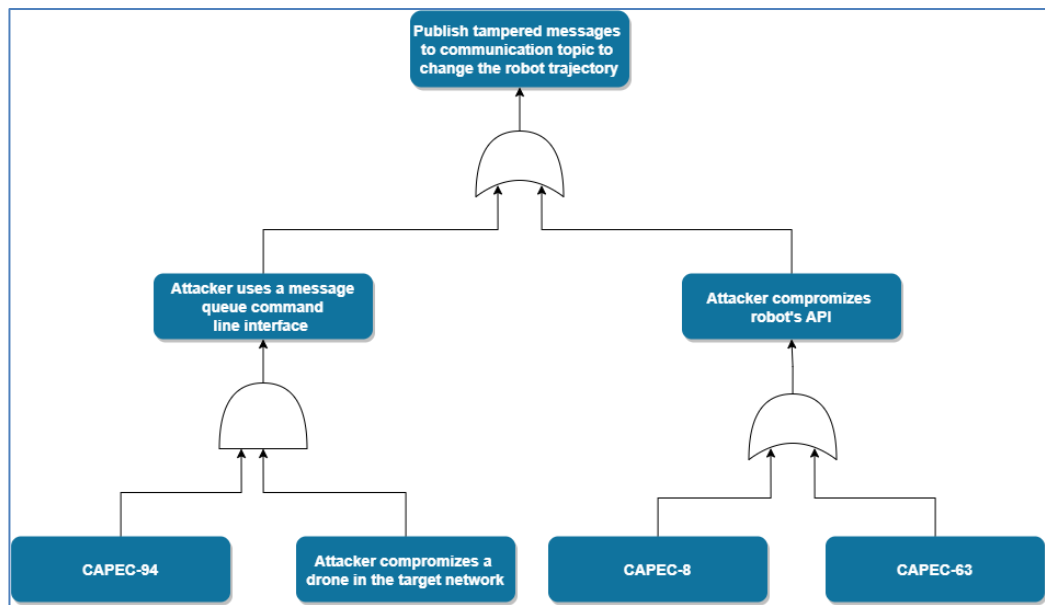


Figure 4: Publish tampered messages Template Attack Tree

Physical vulnerabilities: Physical vulnerabilities, on the other hand, are not included in a repository, where they can be discovered. They are essentially described by the system administrator, during the completion of the system description questionnaire, designed to reveal the physical vulnerabilities of the target system (robots and hardware in general, see 3.1.1.3). Cyber vulnerabilities are connected with specific attacks (CPAEC-IDs) during the Identification of potential attacks process. Similarly, physical vulnerabilities are connected with physical attacks. The Template Attack Tree repository includes trees with physical attacks at their leaves. These trees represent attack patterns of physical attacks. Through the matching process, such trees can also be identified as potential attack trees of the target system. Examples of Template Attack Trees with physical attacks at their leaves are presented below.

- *Lidar Template Attack Tree*

Drones that are used in the SESAME use cases are equipped with Lidar sensors. Lidar stands for "light detection and ranging" or "laser imaging, detection and ranging" and is a method for determining ranges by measuring the time a reflected light needs to reach the receiver.

Figure 5 depicts a Template Attack Tree with tree vulnerabilities at the lower level (leaves of the tree). “Unauthorized personnel can reach drones while they are landed” is the first one according to which, there is the possibility of human with no authorization being in the same physical space with the drones while they do not fly. That could happen at the drone storage space, during their transportation to the desired destination or during the flight preparation stage. “The lidar of the drones is exposed” is the second physical vulnerability. According to that, the lidar sensor of a drone is not enclosed in a protective casing. It is just attached to the drone, being exposed to anyone that could cause damage to it. As it is depicted by the Template Attack tree (AND gate), the presence of the two aforementioned physical vulnerabilities can lead to the security state where the drone can “become object of vandalism”. Unauthorized personnel could take advantage of an exposed sensor and spray paint on the laser source, destroy a part of the sensor, make the sensor point to an area of no importance, or even steal the whole sensor.

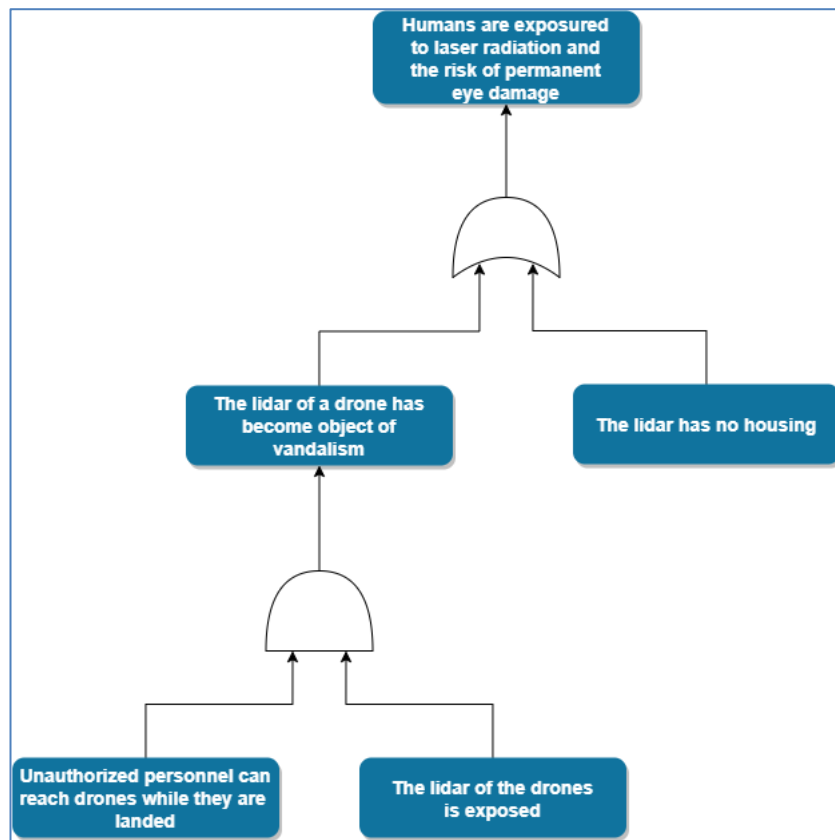


Figure 5: Template Attack Tree with Lidar physical vulnerabilities

The third physical vulnerability of the system is the fact that “The lidar has no housing”. Such a lidar sensor could allow the laser beam to be directed to locations different than the intended targets for range calculating. The OR gate of the Template Attack tree shows that either the security state, where the drone has become object of vandalism, or the “The lidar has no housing” vulnerability can lead to another security state where a human being is exposed to laser radiation and the risk of permanent eye damage. Both a vandalized lidar sensor or a sensor that has no proper housing could direct their laser beam towards a human being causing

physical damage to them and general safety issues. This final security state is the root of the Template Attack Tree and the ultimate goal of the attacker.

It should also be mentioned that there could be other versions of the ultimate goal of the attacker (= root of the tree). The mentioned physical vulnerabilities could lead to a more general outcome, such as that the damaged/non-functional lidar impairs navigation and detection. This could be depicted in a separate Template Attack Tree.

In both cases though, there is a strong link among the security and safety analysis processes. Both Template Attack Trees can be integrated with corresponding fault trees.

- *Compass Template Attack Tree*

A compass is a very common sensor in robot fleets. This is also the case for the SESAME use cases. A multi-robot system equipped with compasses may have a set of physical vulnerabilities. The Template Attack Tree of Figure 6 depicts such vulnerabilities on its leaves.

According to the first vulnerability, an adversary could use a magnetic field source to manipulate the compass readings. Such compass readings are used by a drone to define its direction and calculate its trajectory. The second physical vulnerability indicates that there is an exposed physical port on a drone. Such a port could be used as an entry point for malicious code. An attacker could put a USB stick on that port and inject its code to the drone operating system. That code could then be run and execute a large range of malicious commands, controlling the drone functionality. If the target of the attacker is the drone compass, the exposed port physical vulnerability could lead to the security state where the injected malicious code is used for altering the compass calibration or readings.

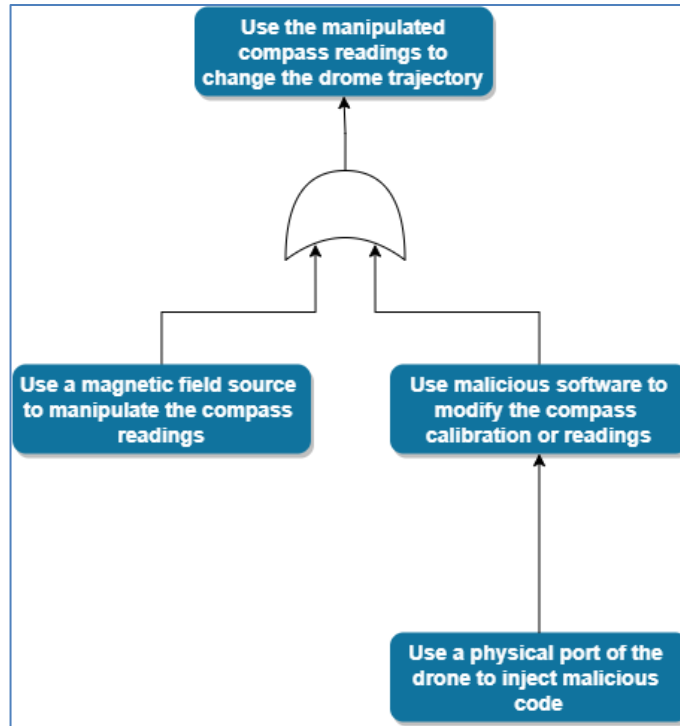


Figure 6: Template Attack Tree with Compass physical vulnerabilities

Either the “Use a magnetic field source to manipulate the compass readings” physical vulnerability or the “Use malicious software to modify the compass calibration or readings” security state can lead to the next security state, where the manipulated compass readings are used to change the drome trajectory. Changing the drone trajectory could have a range of unwanted results with crashing drone to the ground being one of them. According to the Template Attack Tree of Figure 6, altering the drone’s trajectory is the attacker’s ultimate goal.

3.1.6 Generation of attack trees

The next process in the SESAME security methodology is the *Generation of attack trees*, including two different steps. In the first step, the information provided by the CAPEC repository is once again leveraged. CAPEC employs a hierarchical classification that captures various relationships between different attack patterns, such as "CanFollow" and "CanPrecede". The "CanFollow" relationship indicates the attacks that may follow a specific attack pattern in a sequential manner. It provides insights into the potential progression of attacks based on a given attack pattern. On the other hand, the "CanPrecede" relationship reveals attacks that could have been executed prior to a particular attack, setting the stage for its successful execution. By examining these relationships, it becomes possible to construct distinct attack trees, where two or more CAPEC-IDs are connected. An example of such a graph can be seen in Figure 7.

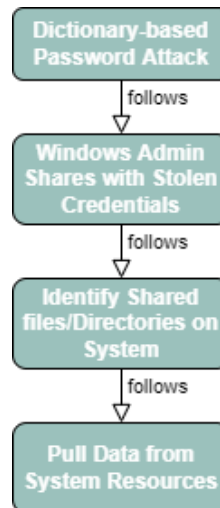


Figure 7: Example graph that can be produced utilizing the CanFollow relationship of CAPEC

Prerequisite for the creation of such a tree is each of the included attacks (CAPEC-IDs) to be included in the lists of CAPEC-IDs that are the output of the *Identification of potential attacks* process. Supposing all the attacks mentioned in Figure 7 are already identified as potential attacks of a target system, during the first step of the *Generation of attack trees* process the presented tree will be created based on the “CanFollow” relationship of the participating attacks. During a *Dictionary-based Password Attack*, an attacker tries all the words of a dictionary as passwords of a specific user account. If the chosen password is in the dictionary, the attack is successful and the attacker gains access. In case the broken account is a Windows administrator account, the attacker could conduct a *Windows Admin Shares with Stolen Credentials* attack. During such an attack, the attacker gets access to Windows Admin Shares, which allow administrators to access all disk volumes on a network-connected system and copy, write and execute files. This opens the way for another attack, the *Identify Shared Files/Directories on System*. During this attack, the adversary may locate and collect sensitive data through the use of shared folders or drives between systems or system parts. Another possible usage of required information is the design of routes in the network that serve other attacks. An attack that can follow is the so-called *Pull Data from System Resources*. During this attack, an adversary pulls data from resources that has access, such as files or memory. The attacker does not need to know what the information that they pull is. The scanning of the information can be done afterwards.

In the second step of the *Generation of attack trees* process utilization of Template Attack Trees takes place. As it is already mentioned in 3.1.5, a Template Attack Tree includes a number of specific attacks (CAPEC-IDs) at its leaves, the ultimate goal of an attacker at the root of the tree, and a number of attacker’s sub-goals in between. The *Identification of potential attacks* process creates a list of CAPEC-IDs that are considered relevant to the target system. By “relevant” we mean that there are system vulnerabilities that an attacker may take advantage of to conduct these attacks. This list of CAPEC-IDs is compared with the CAPEC-IDs at the leaves of each available Template Attack Tree. An identified match reveals a path from the leaves of the tree to its root (= ultimate goal of the attacker), or a series of actions that an attacker can follow to achieve their goal. The existence of a match in a Template Attack Tree leads to its characterization as “potential attack tree” for the target system. The output of the *Generation of attack trees* process is a list of potential attack trees.

Moreover, Template attack trees can be used for merging together graphs that have been created in the first step of this process. If the vulnerabilities mentioned in the leaves of a template are included in a graph, that graph could substitute the leaf. In that way, more than one graph could replace leaves and be merged in a Template attack tree.

3.1.7 Generation of security EDDIs

The output of the *Generation of attack trees* process serves as input in the *Generation of security EDDIs* process. During the latter, all the produced information is used for the creation of the security EDDI.

The EDDI solution represents a progression from the DDI concept, encompassing additional elements essential for real-time implementation and addressing concerns associated with MRS. EDDI functions as an extended version, serving both as a dependability artefact during the design phase and as a dynamic tool for managing dependability during runtime. Its primary role involves two aspects: firstly, facilitating online monitoring to oversee and regulate the safety and security of the system, and secondly, enabling distributed communication among various components of the system to effectively manage dependability within a broader MRS framework.

The EDDI's features include the following:

- Event monitoring to monitor dependability-related inputs from the system;
- Runtime diagnostics to determine probable causes and possible consequences of detected failure and security violation events;
- Dynamic risk prediction, to update design-time risk estimates with new information based on the current system state;
- Mitigating actions and recovery planning, such as recommending the system enter a safe failure state or a degraded mode to continue operation.
- Intercommunication with other connected EDDIs to both assure them of the system dependability status and respond to errors reported by other EDDIs.

More details about the EDDI and the DDI concepts can be found in D4.5 “Safety Analysis Concept and Methodology for EDDI development (Final Version)”, where an elaborated description of both can be found along with their architectures.

The information that is produced by the security assessment must conform to the ODE metamodel. Structured Assurance Case Meta-Model (SACM), a metamodel specialised for the creation of structured system assurance cases, provides the ODE with assurance case support. In our case, an assurance case incorporates the arguments and evidence that support the claim that a given system or service is able to satisfy safety and security requirements. The form such an assurance case can be expressed in is a machine-readable model carrying information such as the scope of the system, the operational context and the safety and/or security arguments [21].

The ODE metamodel has been extended during the lifetime of SESAME project to be able to model additional information that is produced by the SESAME security assessment process. Such information includes common software/hardware

vulnerabilities of systems and system components, related weaknesses due to these vulnerabilities, and common attacks that can be conducted based on defined system weaknesses. All the proposed ODE extensions are described in details in combined deliverable D4.2 – D5.2 “Safety and Security-Targeted ODE and EDDI specification”.

ODE includes a security-oriented package called Threat Analysis and Risk Assessment (TARA). This package captures *Risk Assessment* that is based on *Threat Agents*, which perform *Attacks* taking advantage of *Assets* with identified *Vulnerabilities*. The performed attacks can be addressed by Security Capabilities of the system, which are implemented by Security Controls [22]. The TARA package is one of the most important ODE packages as far as security is concerned. For convenience the proposed additions for the TARA package along with their relationships with classes of the FailureLogic and FTA packages are presented, in the form of a class diagram, in Figure 8.

EDDIs can also incorporate information for communicating with runtime security monitoring tools. Valuable input from tools such as IDSs, Anti-Viruses, and Breach Detection Systems (BDSs) can be included in the security part of an EDDI, towards to definition of remedy actions. IDSs are used for the identification of malicious packets. In case of protection of known attacks, attack signatures as used for the creation of rules that recognize specific patterns in the header or body of the traffic packets. As far as the unknown attacks are concerned, anomaly detection techniques are used, detecting alteration in the traffic from the normal one. Moreover, Anti-Viruses are another type of protection that detects and removes malware from the host. Breaches and side-channel attacks are detected by BDSs. Finally, anti-phishing solutions protect from phishing attempts.

The current implementation based on the SESAME security methodology utilizes Snort, an open-source intrusion detection and prevention system that utilizes signature-based detection to analyse network traffic, allowing it to identify and respond to known patterns of security threats.

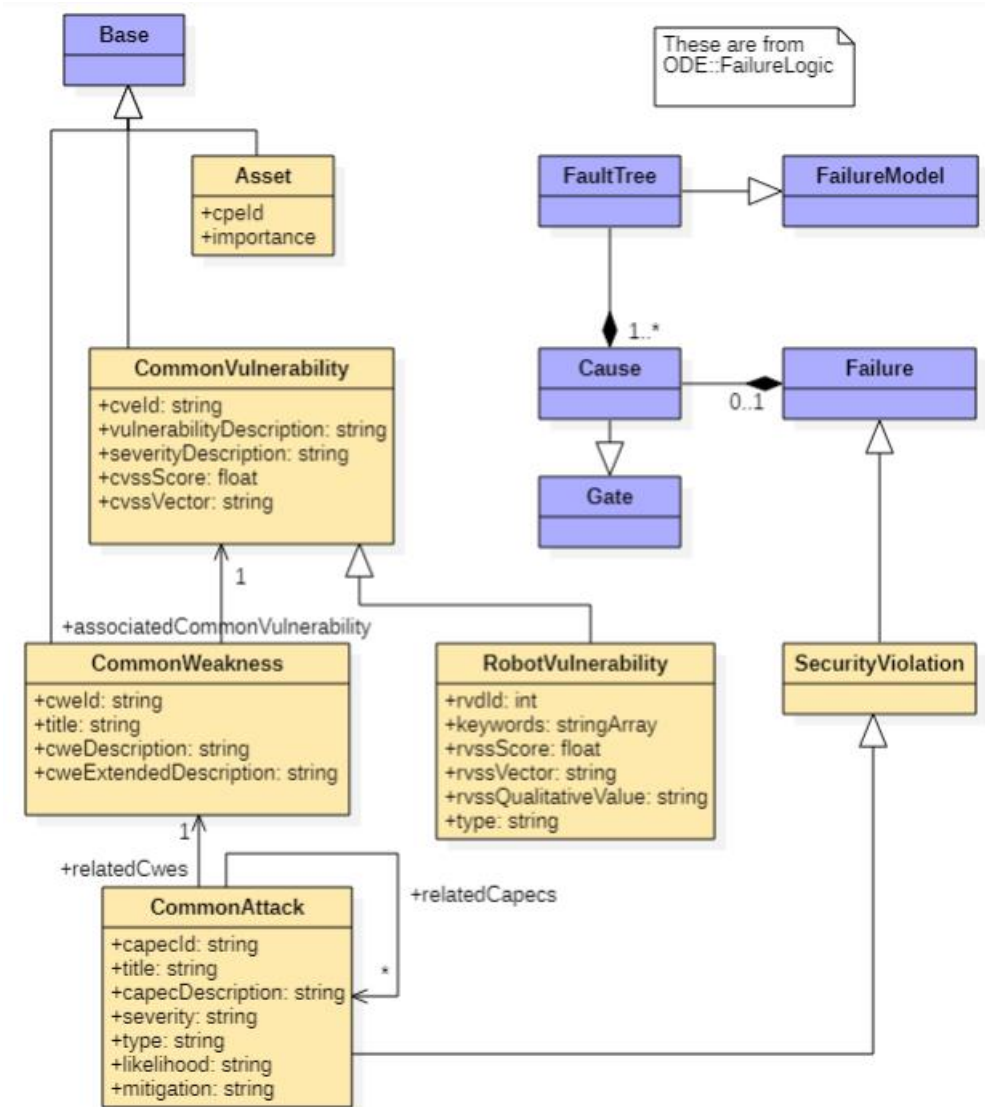


Figure 8: Proposed additions for the TARA package along with their relationships with classes of the FailureLogic and FTA packages

As it is described in D4.5 “Safety Analysis Concept and Methodology for EDDI development (Final Version)”, the deployment of an EDDI could be done two ways. It could be synthesized into code and run on the target platform or, in a virtual machine-style approach, the EDDI is executed by a target-specific native program.

3.2 SAFETY AND SECURITY

According to the definitions provided in [23], security refers to safeguarding plants or machinery against unauthorized external access and protecting sensitive data from corruption, loss, and unauthorized internal access. On the other hand, safety’s goal is the functional integrity of plants, specifically protecting individuals and the environment from foreseeable risks that may arise from machinery. While security and safety possess distinct meanings, their relationship is obvious.

A conducted attack that results in the manipulation of robot parameters, control or calibration, can lead to human injuries during human-machine interactions.

Furthermore, the integration of external safety sensors into a robotic system introduces additional attack surfaces, as mentioned in [24]. This complex relationship between security and safety makes it crucial to take care both these aspects to avoid faulty and unexpected robot behaviour. Security measures will protect against unauthorized access and malicious manipulation while safety mechanisms will safeguard human well-being and prevent potential harm during human-robot interactions.

In the concept of EDDIs, security related information such as the fact that a target system is under attack, the type of the attack, and potential consequences of the attack are part of the security part of an EDDI and are communicated to the corresponding safety part. To the EDDI monitoring the system, both a hardware fault and a security attack could result in the same danger. The EDDI is intended to assess the consequences of that danger and make recommendations to avert or mitigate it on the basis of the diagnosed cause. For safety, this might mean switching to a backup component, while for security it might entail filtering the incoming traffic or distributing it across multiple resources.

The security assessment process described herein can serve as input for the safety reasoning model. Security and safety are both important if the dependability goals are to be achieved, and so combining both security and safety assessment is necessary to ensure the dependability of a robotic system.

4. TOOLS FOR APPLYING SECURITY ASSESSMENT AND EDDI PRODUCTION

The description of the SESAME security assessment methodology was given in section 3. The primary objective of the security assessment process is to gather information regarding the security status of a particular system. The discovered vulnerabilities of the individual system components lead to the identification of potential attacks that an adversary could try to conduct, taking advantage of the existing system flaws.

The extension of the ODE metamodel allows us to store the gathered information in an EDDI model-based artefact along with all the dependability information of a system. While EDDIs are primarily intended for runtime usage, the information included within them is collected during the design and testing phases using various tools and techniques, described in the next subsections of this deliverable. Each subsection refers to one of the SESAME security methodology processes.

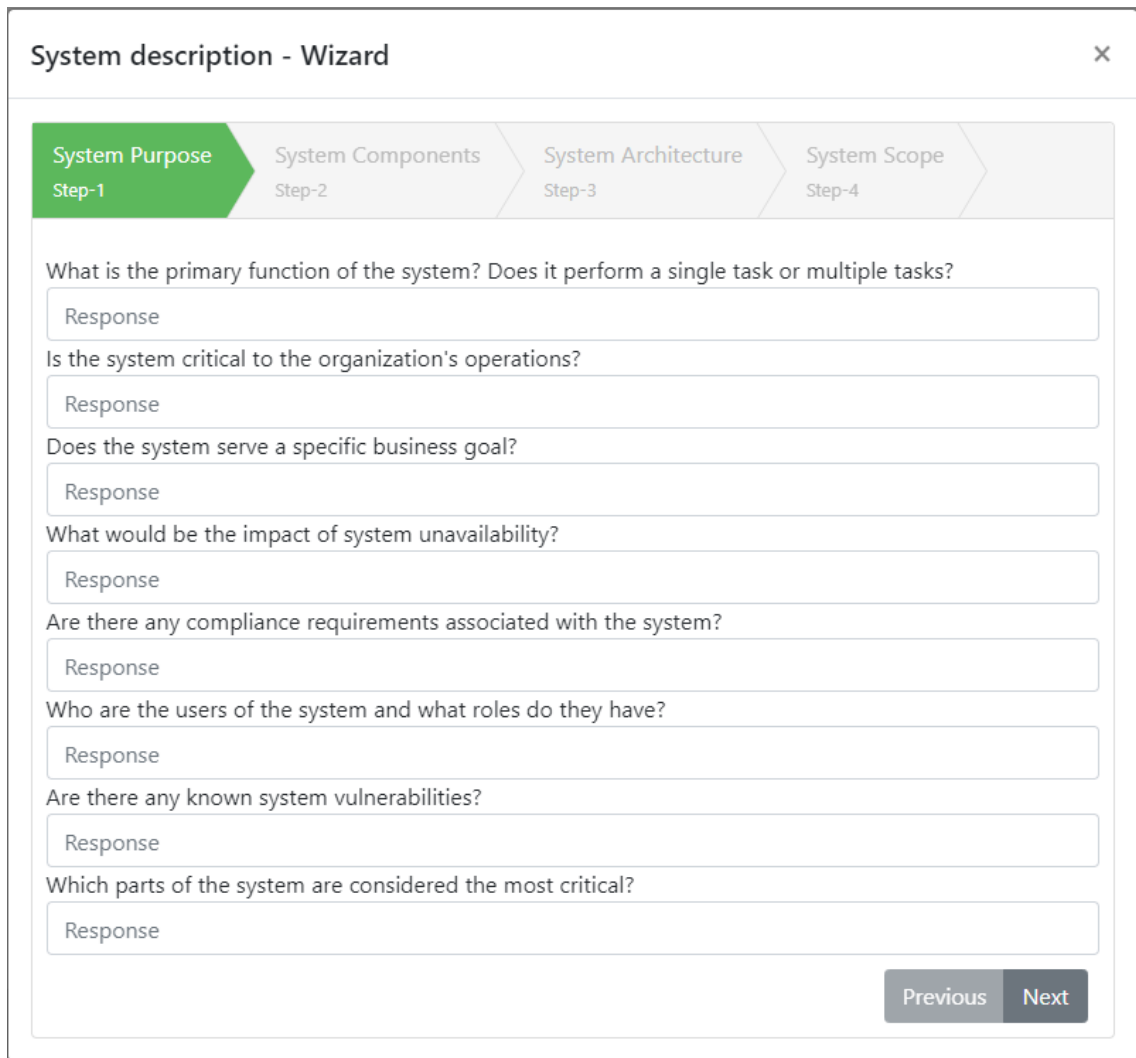
4.1.1 System description

To facilitate the collection of security information of the target system for the SESAME security assessment, two individual ways have been used during the lifetime of the project. The first one is a user interface (UI) for gathering the necessary details about the system, consisting of forms and questionnaires that the system administrator needs to fill out to provide the required information. The second one is OpenVAS, an automated scanner tool that can scan given network and/or subnetworks for available services. The advantage of the usage of such scanning tools is that they can reveal services that are running in devices, which are part of the target system, and the provider of the system information may not be aware of. On the other hand, their disadvantage is that they can be used only after the system is up and running, otherwise the vulnerability scanning tools cannot produce an output.

UI:

These forms and questionnaires of the created UI capture specific details about the system architecture, components, assets, entry points, and trust boundaries. The goal of each form or questionnaire is to guide the system administrator to provide relevant information.

Structuring the information collection process through a UI, it becomes easier to ensure that all necessary details are gathered consistently and in a standardized format. This allows the following steps of the security assessment methodology to take place. Moreover, the UI is designed in a user-friendly manner, providing clear instructions speeding the whole process of gathering information, enhancing the efficiency and effectiveness of the assessment. The UI is a web-based application that is developed in Java utilizing Bootstrap, a popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first websites. The UI application urges the user to provide security information of the target system in steps, in the form of a software wizard or setup assistant. Breaking down a complex, rare, or unfamiliar task into simpler components can significantly facilitate its execution. In that way, individuals are provided with step-by-step guidance that navigates them through each simplified piece of the task.



System description - Wizard [X]

System Purpose Step-1 | System Components Step-2 | System Architecture Step-3 | System Scope Step-4

What is the primary function of the system? Does it perform a single task or multiple tasks?

Is the system critical to the organization's operations?

Does the system serve a specific business goal?

What would be the impact of system unavailability?

Are there any compliance requirements associated with the system?

Who are the users of the system and what roles do they have?

Are there any known system vulnerabilities?

Which parts of the system are considered the most critical?

Previous Next

Figure 9: Step -1 of system description – SESAME security methodology

Figure 9 above depicts one of the steps of the wizard that refers to the overall purpose of the target system. Answering questions like “What would be the impact of system unavailability?” or “Which parts of the system are considered the most critical?” allows us to have an idea about how crucial each of the system parts is and, as a consequence, how critical a corresponding attack could be.

Figure 10 depicts yet another step of the wizard for the identification of each of the system components. As it is described in subsection 3.1.1.2, information such as component host, name and version, vendor, and CPE is considered necessary for pinpointing each individual system component. During this step, the user needs to fill out the corresponding fields. These fields correspond to just one component. Using the “Add component” button, the description of more components can be added.

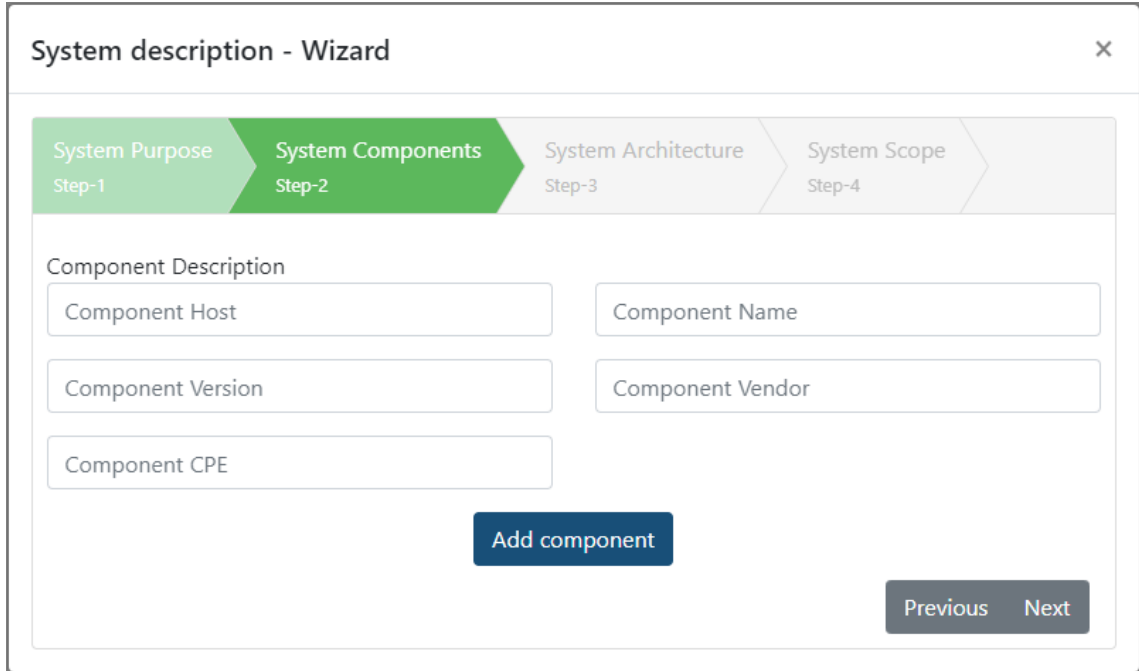


Figure 10: Step -2 of system description – SESAME security methodology

OpenVAS:

Although Open Vulnerability Assessment Scanner (OpenVAS) is mentioned in this subsection as a way to discover exposed services of a system, its capabilities are way beyond that. It is a generic scanner that offers several capabilities and considers a significant number of different vulnerabilities. Its core strength is that it can meticulously scan all ports on the target system for active services and provide a comprehensive report on the discovered assets, such as running software, specific version numbers etc. Furthermore, OpenVAS is able to conduct attacks to the discovered services by using a plethora of known exploits and reporting on the vulnerable ones by providing a high-level description of each vulnerability and the CVE’s assigned CVSS score and severity level. Another advanced capability of OpenVAS is that it already makes use of wrappers for other vulnerability scanners (e.g., Nmap, wapiti) and leverages them to enhance its coverage, as well as number and type of detected vulnerabilities. Finally, OpenVAS offers a set of predefined configurations that cover the most common scanning scenarios, including fast, fast ultimate, deep and deep ultimate scans. One last feature of OpenVAS is the addition of custom configurations through its administrator dashboard.

The following steps have to be followed for the installation of OpenVAS, in a containerized form (Listing 1):

```
~/# git clone https://github.com/mikesplain/openvas-docker.git
~/# sudo docker run -d -p 443:443 --name openvas mikesplain/openvas
```

Listing 1: OpenVAS installation commands

As it can be seen, the first command clones the code from the corresponding GitHub project and then the docker run command is used for the creation and start of the

OpenVAS container. By doing so, the individual OpenVAS components will be installed and become available. The whole installation, gsad, will be running on port 443. The OpenVAS Scanner (openvassd) will be running on TCP Port 9391 and the OpenVAS Manager (openvasmd) on TCP port 9390. Finally, the redis-server will be running on TCP 6379. The OpenVAS web interface is available in the browser, which shows the login screen for the Greenbone Security Assistant (Figure 11).

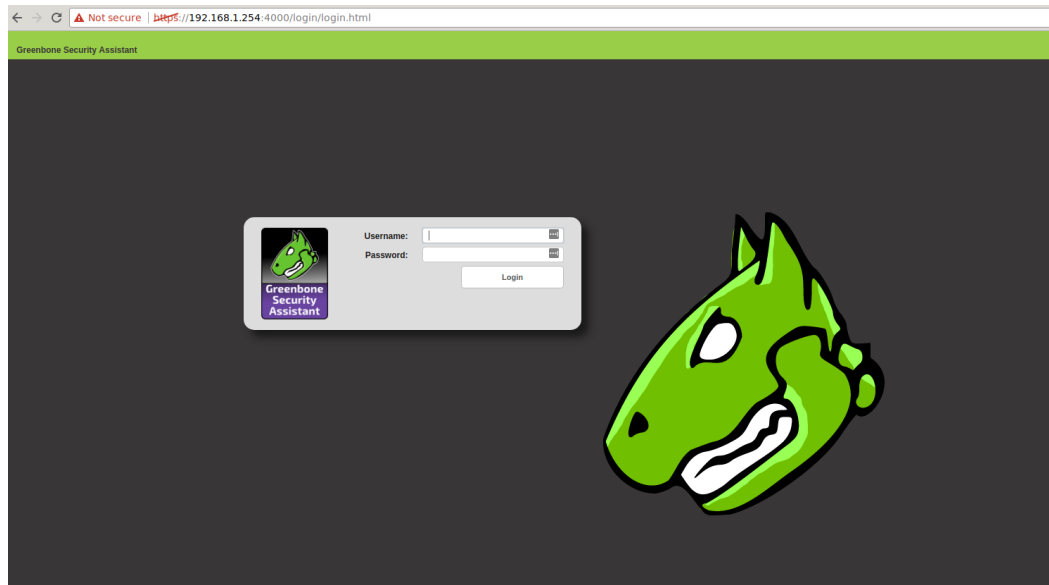


Figure 11: OpenVAS web interface

It should be mentioned that while OpenVAS was extensively used as part of the proposed security assessment process, a second scanner was also tested (see D5.3). Moreover, the modular design of the identification of vulnerabilities process allows for the easy addition of even more such tools.

4.1.2 Identification of vulnerabilities

Based on the information gathered with the methods and tools described in the previous subsection, the *Identification of vulnerabilities* process requests free repositories, catalogs and databases for known vulnerabilities of the recognized software, hardware and communication protocols.

Two main repositories are used by the SESAME security methodology, CVE and RVD. CVE has been already mentioned in 2.2.3 and a more extended description can be found in section 2.2.3 (Security knowledge repositories) of D5.1. CVE is a list of computer security flaws, cybersecurity vulnerabilities, and can be used for searching or incorporated into products and services for free. Each of these flaws is assigned an identifier called CVE-ID, which is used as a dependable way to uniquely recognise vulnerabilities. Likewise RVD is described in 2.2.3 and in section 2.2.3 of D5.1. RVD includes robot related vulnerabilities and bugs that are referred to software and hardware. The aim is to record and categorize robot related flaws. We decided to incorporate RVD to our methodology due to the fact that SESAME focuses on MRSs.

Towards the goal of *Identification of vulnerabilities* process, two parsers are needed, one for each of the two repositories. These parsers search for known vulnerabilities

based on the identified system components. These two parsers are CVE-search and RVD custom parser.

CVE-search: CVE-Search is an open-source vulnerability search and management tool based on the CVE repository. It allows users to search for specific vulnerabilities, explore detailed information about them, and track their status and associated resources. CVE-Search offers information about known vulnerabilities in various software and systems. The goal of the tool is to ease the process of vulnerability management.

CVE-search is a tool that imports CVE and CPE into a MongoDB, facilitating search and processing of CVEs. The main advantage of this tool is the fact that a local instance of CVE is created serving lookup requests. In that way, direct requests to the public CVE databases are reduced. At the same time, local requests are served faster without exposing sensitive information to the internet. Among the cve-search offerings are the following: i) a back-end to store vulnerabilities and related information, ii) an intuitive web interface for search and managing vulnerabilities, iii) a series of tools to query the system and a web API interface. cve-search is used by many organizations including the public CVE services of Computer Incident Response Center Luxembourg (CIRCL). The source code is available on GitHub¹. A whole community maintains it including CIRCL.

There are different ways to form a request asking for vulnerabilities.

- Request returning vulnerabilities directly assigned to a specific product (`./bin/search.py -p microsoft:windows_7 -a -o json`).
- Request returning vulnerabilities based on text search in the vulnerability summary (`./bin/search.py -f "robotic simulator" -a -o json`).
- Request for a specific CVE ID (`./bin/search.py -c CVE-2010-3333`).
- Request the last 2 CVE entries in atom format (`./bin/dump_last.py -f atom -l 2`).

More details about the tool's installation and usage can be found in section 2.1.1.1 of D5.3 (Tools for Automated Security Analysis of MRS and for Production of EDDIs (Initial Version)).

RVD custom parser: CVE-search is a great tool for searching and processing vulnerabilities from the CVE catalogue. However, we need an additional tool that will offer the same functionality for the RVD database. RVD comes with a set of tools for the management of the database entries and is available as an open source project at GitHub². Although RVD project developed by Alias Robotics provides its own set of access tools, they do not fulfill our specific requirements. These tools lack essential for us functionalities, such as the capability to search the vulnerability database for robot vulnerabilities based on a provided product description or a CPE identifier.

¹ <https://github.com/cve-search/cve-search>

² <https://github.com/aliasrobotics/RVD>

Additionally, it would be beneficial to have the ability to query the database for related CWEs associated with a particular CVE entry.

Towards the desired functionality described in the previous paragraph, a custom RVD parser has been created. The RVD installation offers the “`rvd list --dump --label vulnerability`” command that returns all the RVD database entries, which are labeled as vulnerabilities. An example of such an entry is depicted in Listing 2. The provided information for each robot vulnerability include related CVEs and CWEs, affected systems, severity scores (RVSS, CVSS), exploitation and mitigation descriptions.

```

id: 3337
title: Service DoS through arbitrary pointer dereferencing on KUKA
simulator
type: vulnerability
description: "Visual Components (owned by KUKA) is a robotic simulator
that allows simulating factories and robots in order to improve plan-
ning and decision-making processes. ... Accordingly, a DoS in the simu-
lation might have higher repercussions, depending on the Industrial Con-
trol System (ICS) ICS infrastructure."
cwe: CWE-248
cve: CVE-2020-10292
keywords:
- KUKA, RMS sentinel LM, Visual Components, DoS
system: Visual Components Network License Server 2.0.8
vendor: KUKA Roboter GmbH, Visual Components
severity:
  rvss-score: 6.1
  rvss-vector: RVSS:1.0/AV:IN/AC:L/PR:N/UI:N/S:U/Y:Z/C:N/I:L/A:H/H:N
  severity-description: High
  cvss-score: 8.2
  cvss-vector: CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:H
links:
- https://cwe.mitre.org/data/definitions/248.html
- https://www.visualcomponents.com/products/downloads/
- https://www.visualcomponents.com/products/visual-components/
flaw:
  phase: runtime-operation
  specificity: subject-specific
  architectural-location: application-specific
  application: Visual Components, RMS sentinel LM
  subsystem: simulation
  package: null
  languages: null
  date-detected: null
  detected-by: Sharon Brizinov (Claroty)
  detected-by-method: testing-dynamic
  date-reported: 2020-10-27
  reported-by: Sharon Brizinov (Claroty)
  reported-by-relationship: security researcher
  issue: https://gitlab.com/aliasrobotics/offensive/rvd/flaws/-
/issues/712
  reproducibility: always
  trace: null
  reproduction: null
  reproduction-image: null
exploitation:
  description: |
    To exploit this vulnerability the attacker needs to have network
    access to the license server (either because

```

```

it's exposed or because the internal network has been compromised.
Cause is related to the number of requested
strings to merge, which is not correlated to the number of strings
provided, and so arbitrary pointers from the
stack are popped out and dereferenced. This results with an un-
caught Access Violation exception which terminates
the program. PoC available constructs a response reply to
featureInfoToFile with is a mismatch between the
number of strings to merge and the requested amount leading to an
Access Violation exception and terminating the
program. See alurity's robotsplit/exploits/kuka/rms exploits.
exploitation-image: Not available
exploitation-vector: null
exploitation-recipe:
  networks:
  - network:
    - driver: bridge
    - name: kuka-simulation
    - subnet: 14.0.0.0/24
  vms:
  - vm:
    - name: vm1
    - path: $(pwd)/vms/visualcomponents_2.0.8
    - network: kuka-simulation
    - ip: 14.0.0.4
  containers:
  - container:
    - name: attacker
    - modules:
      - base: regis-
try.gitlab.com/aliasrobotics/offensive/alurity/alurity:latest
      - volume: regis-
try.gitlab.com/aliasrobotics/offensive/alurity/expl_robosploit/expl_ro
bosploit:latest
      - volume: regis-
try.gitlab.com/aliasrobotics/offensive/alurity/deve_atom:latest
      - volume: regis-
try.gitlab.com/aliasrobotics/offensive/alurity/reco_nmap:latest
      - volume: regis-
try.gitlab.com/aliasrobotics/offensive/alurity/expl_icssplit:latest
      - volume: regis-
try.gitlab.com/aliasrobotics/offensive/alurity/expl_metasploit:latest
      - volume: regis-
try.gitlab.com/aliasrobotics/offensive/alurity/fore_wireshark:latest
      - network: kuka-simulation
  mitigation:
  description: |
    Do not launch Visual Components while connected to local or wide
    area networks. Contain the simulation through
    virtualization.
  pull-request: null
  date-mitigation: null

```

Listing 2: Example robot vulnerability from the RVD database

The whole set of available robot vulnerabilities is the input for our custom parser. A set of Java classes has been created for storing and managing the information provided for the incoming robot vulnerabilities (Figure 12). The main class is called “RvdVulnerability”, while four more subclasses are needed, called “Severity”, “Exploitation”, “Flaw”, and “Mitigation”.

```

@PostMapping("/rvdinsert")
public String rvdInsert(@RequestBody ArrayList<RvdVulnerability> rvdJson) {
    //Generate the rvd database (rvdVulnerabilities) with the input from the
    rvdjson array

    ...

    System.out.println("Local RVD Repository has been updated");
    return "rvdresult";
}
    
```

Listing 3: REST API for the update of the local version of the RVD database



Figure 12: RVD Java classes of the custom RVD parser

A REST API has been created for the RVD parser to update the local version of the RVD database. The corresponding code is depicted in Listing 3. As it can be seen, the API endpoint is <http://ipAddress:port/rvdinsert> and the anticipated body of the request is a list of RVD vulnerabilities. The structure of the RvdVulnerability class can be seen in Figure 12. The `@PostMapping` annotation ensures that HTTP POST requests are mapped onto a specific handler method, the `rvdInsert` in this case. The request is expected to have a body in application/json format. The `@RequestBody` annotation enables the automatic deserialization of the request body onto a Java object. After the mapping of the request body to the corresponding Java instance, the instance is inserted into the `rvdVulnerabilities` array list in the file system. This API endpoint allows for the regular update of the RVD local version to include any newly added vulnerabilities.

Another exposed REST API is utilized for querying the local RVD database instance for CWEs based on a given CVE. Listing 4 depicts the corresponding code.

```

@PostMapping(value = "/searchwithcve")
public String searchWithCve(@RequestBody String cveId) {

    ArrayList<String> cweFilteredArrayList = new ArrayList<>();
    for (int i = 0; i < rvdVulnerabilities.size() ; i++) {

        if (rvdVulnerabilities.get(i).cve.equals(cveId)) {
            cweFilteredArrayList.add(rvdVulnerabilities.get(i).cwe);
        }
    }
    ...

    return "rvdresult";
}

```

Listing 4: REST API for querying for CWEs based on a given CVE

The API endpoint is <http://ipAddress:port/searchwithcve> and the anticipated body of the request is a CVE-ID. The `@PostMapping` annotation ensures that HTTP POST requests are mapped onto the `searchWithCve` handler method. What follows is the collection of all the related CWEs of the vulnerability with the given CVE-ID.

The implementation of the described custom RVD parser is an ongoing work that will be continued the coming months of the project lifetime. More functionality will be added, such as the ability to search for vulnerabilities using the name and version of each software present in the system in question.

A parser searches said vulnerability directories and, using the name and version of each software present in the system in question, spots the associated vulnerabilities. Each of those vulnerabilities are uniquely identified by the CVE identifiers (CVE-IDs). A list of such CVE-IDs is the output of this process. The said vulnerability directories are constantly updated with information regarding newly discovered vulnerabilities. Inherently the approach followed here is also not static as it will be in sync with the updated directories.

4.1.3 Identification of potential attacks

The output of the *Identification of vulnerabilities* process (a list of CVE-IDs) serves as input for the *Identification of potential attacks* process. The rationale behind this process has been described in 3.1.1.1. The fact that two different vulnerability repositories (CVE and RVD) are utilized from the SESAME security assessment methodology allows the usage of two individual tools in this particular process, CVE-search and CAPEC custom identifier.

CVE-search: CVE-search is already described in the previous subsection; however, it is also mentioned here since it can be requested for known attacks related to a provided CVE-ID or a specific product (software/hardware). In case of requesting for a specific vulnerability and if the output of the request is defined to be in JSON format, one part of the information that is returned is a list of attacks that are related to the given vulnerability.

Listing 5 and Listing 6 depict the search command and the corresponding output respectively.

```
~/# ./bin/search.py -p microsoft:windows_7 -a -o json
```

Listing 5: cve-search command for vulnerabilities related to specific software

```
{
  "Modified": "2017-09-19 01:31:00",
  "Published": "2011-03-03 20:00:00",
  "access": {
    "authentication": "NONE",
    "complexity": "HIGH",
    "vector": "NETWORK"
  },
  "assigner": "product-security@apple.com",
  "capec": [
    {
      "execution_flow": {
        "1": {
          "Description": "[Identify target application]...",
          "Phase": "Explore",
          "Techniques": []
        },
        "2": {
          "Description": "[Find injection vector] The ...",
          "Phase": "Experiment",
          "Techniques": ["Provide large input to a ..."]
        },
        "3": {
          "Description": "[Craft overflow content] The ...",
          "Phase": "Experiment",
          "Techniques": ["Create malicious shellcode ..."]
        },
        "4": {
```

```

        "Description": "[Overflow the buffer] Using the...",
        "Phase": "Exploit",
        "Techniques": []
    }
},
"id": "8",
"loa": "High",
"name": "Buffer Overflow in an API Call",
"prerequisites": "The target host exposes an API ...",
"related_capecs": ["100"],
"related_weakness": ["118", "119", "120", "20", "680", "697", "733",
"74"],
"solutions": "Use a language or compiler that ... ",
"summary": "This attack targets libraries or ...",
"taxonomy": {},
"typical_severity": "High"
}
],
"cvss": 7.6,
"cvss-time": "2017-09-19 01:31:00",
"cvss-vector": "AV:N/AC:H/Au:N/C:C/I:C/A:C",
"cvss3": null,
"cwe": "CWE-119",
"exploitabilityScore": 4.9,
"id": "CVE-2011-0112",
"impact": {
    "availability": "COMPLETE",
    "confidentiality": "COMPLETE",
    "integrity": "COMPLETE"
},
"impactScore": 10.0,
"last-modified": {
    "$date": 1505784660000
},
"products": ["itunes", "webkit"],
"references": ["http://support.apple.com/kb/HT4554"],
"summary": "WebKit, as used in Apple iTunes before 10.2 ...",
"vendors": ["apple"],
"vulnerable_configuration":
["cpe:2.3:a:apple:itunes:4.6.0:*:*:*:*:*:*"],
    "vulnerable_configuration_cpe_2_2": [],
    "vulnerable_configuration_stems": ["cpe:2.3:a:apple:itunes"],
    "vulnerable_product": ["cpe:2.3:a:apple:itunes:4.6.0:*:*:*:*:*:*"],
    "vulnerable_product_stems": ["cpe:2.3:a:apple:itunes"]
}

```

Listing 6: cve-search example vulnerability output

The command in Listing 5 requests the CVE repository for all the known vulnerabilities related with Microsoft Windows 7 operating system. The very last part of the command shows that the requested output should be in JSON format.

Listing 6 depicts just one of the vulnerabilities related to Microsoft Windows 7 operating system, with id “CVE-2011-0112” and CVSS score “7.6”. Under the “capec” element we see that the vulnerability is related to the attack named “Buffer Overflow in an API Call” and id “8”.

CVE-search is used for identifying the potential attacks that can be conducted against a target system taking advantage of known vulnerabilities that are documented in the CVE repository.

CAPEC custom identifier: A tool with similar functionality to the one described in the previous paragraphs has been developed for the CAPEC database. Similarly to the RVD parser, CAPEC custom identifier utilizes a local instance of the CAPEC catalogue for retrieving information about known attacks based on given known weaknesses. A set of Java classes has been created for storing all the information coming from the CAPEC repository (Figure 13). As we can see, the CAPEC repository includes an “AttackPatternCatalog”, which includes a list of “AttackPatterns”. The “AttackPattern” class corresponds to the known vulnerabilities including a lot information such as related weaknesses, related attacks and proposed mitigation actions.

A REST API has been created for inserting the latest version of the CAPEC catalogue, creating a local instance. The corresponding code snippet is depicted in Listing 7.

```
@PostMapping(value = "/capecinsert")
public String capecInsert(@RequestBody Capec capecJson) {
    //Generate the capec database (capecs) with the input from the capecJson
    object
    ...

    System.out.println("Local CAPEC repository has been updated");
    return "rvdresult";
}
```

Listing 7: REST API for the update of the local version of the CAPEC database

As it can be seen, the API endpoint is <http://ipAddress:port/capecinsert> and the anticipated body of the request is a CAPEC object, which includes a list of known attacks. The structure of the CAPEC class can be seen in Figure 13. The `@PostMapping` annotation ensures that HTTP POST requests are mapped onto a specific handler method, the `capecInsert` in this case. The request is expected to have a body in application/json format. The `@RequestBody` annotation enables the automatic deserialization of the request body onto a Java object. The request body is mapped to a corresponding Java instance and then inserted into the `capecs` array list in the file system. This API endpoint allows for the regular update of the CAPEC local version to include any newly added known attacks.

One of the desired functionalities of the CAPEC custom identifier is to be able to discover known attacks based on given CWEs. This list of CWEs is created by the RVD parser we have already described and includes the known weaknesses related to a given

vulnerability. A REST API is available for initiating the process of querying the local CAPEC catalogue instance, and is depicted in Listing 8. The API endpoint is <http://ipAddress:port/searchwithcwe> . The `@PostMapping` annotation ensures that HTTP POST requests are mapped onto the corresponding “searchWithCwe” handler method. Regarding the logic, the CAPEC catalogue is searched against every incoming CWE. If there is a match with the related CWEs of a known attack, the known attack is stored in the “capecFilteredArraylist” list, creating the output.

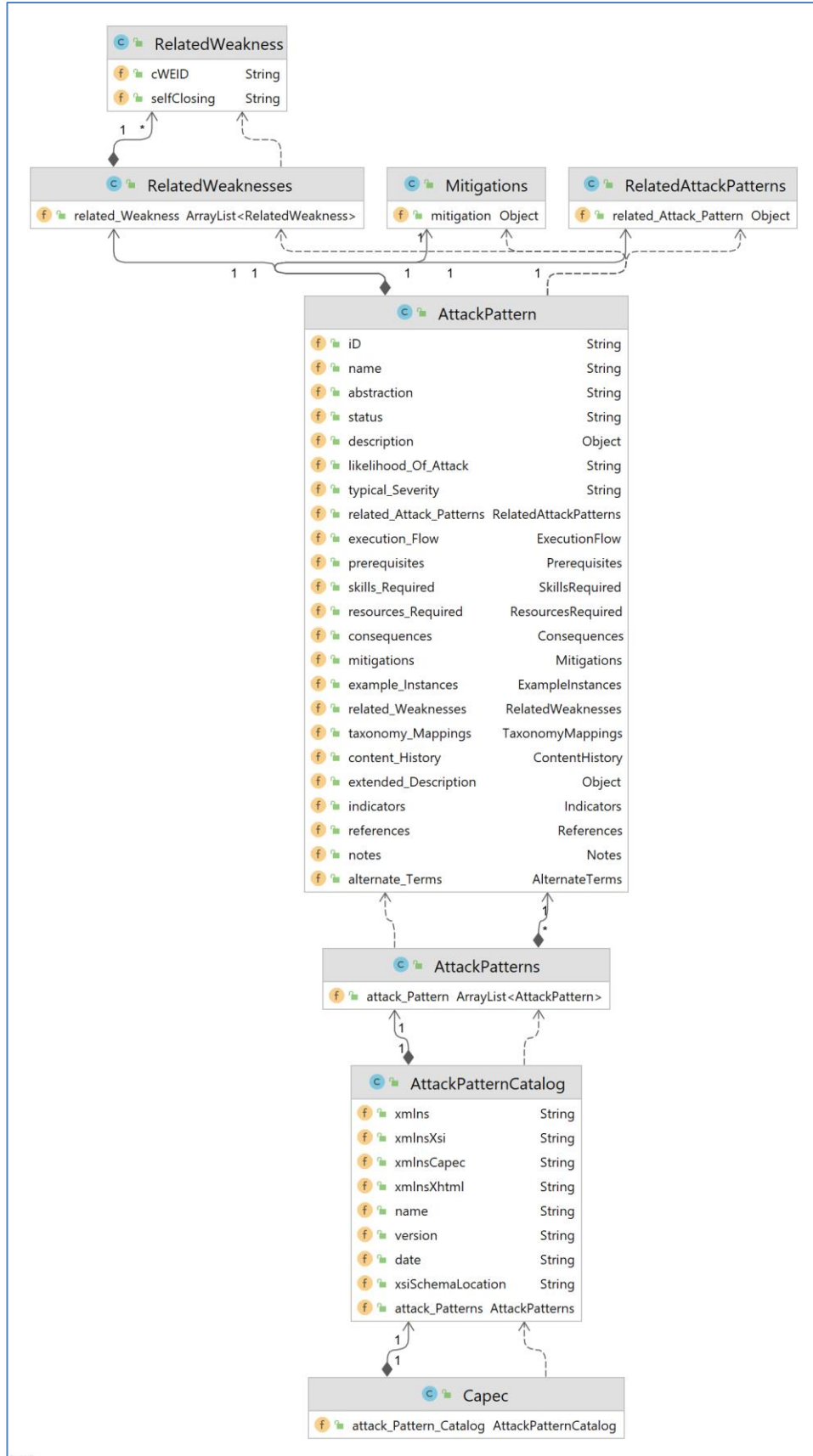


Figure 13: CAPEC classes of the custom CAPEC identifier

```

@PostMapping(value = "/searchwithcwe")
public String searchWithCwe(@RequestBody ArrayList<String> cweIds) {

    ArrayList<AttackPattern> capecFilteredArraylist = new ArrayList<>();
    //Iterate all cwe to find the capecs
    for (int i = 0; i < cweIds.size(); i++) {
        String tempCWE= cweIds.get(i).substring(4);
        for (int j = 0; j < capecs.size() ; j++) {
            AttackPattern tempCapec = capecs.get(j);
            for (int k = 0; k < temp-
Capec.related_Weaknesses.related_Weakness.size() ; k++) {
                RelatedWeakness tempRelatedWeak-
ness=tempCapec.related_Weaknesses.related_Weakness.get(k);
                if (tempRelatedWeakness.cweID.equals(tempCWE)){
                    capecFilteredArraylist.add(tempCapec);

                }
            }
        }
    }
    System.out.println("The following CAPECs have been identified as potential
attacks related to :" + cveId);
    for (int i = 0; i < capecFilteredArraylist.size(); i++) {
        System.out.print(" "+capecFilteredArraylist.get(i).id);

    }

    return "rvdresult";
}
}

```

Listing 8: REST API for querying for CAPECs based on a given CWE list

4.1.4 Generation of attack trees

The *Generation of attack trees* process includes two steps, the creation of relatively small attack trees based on the "CanFollow" and "CanPrecede" relationships between different attack patterns of the CAPEC repository, and the utilization of the Template Attack Trees.

Regarding the first step, a Java class has been created for storing the information of a "CanFollow" or "CanPrecede" tree (Listing 9). Such a tree is an ArrayList of corresponding nodes. Each node is directly connected to its parent node and children nodes. This connection stores the information of which attack could follow or precede a current attack. Roots of the "CanFollow" and "CanPrecede" trees compared with Template Attack Trees' leaves, and in case of a match, the former are incorporated to the later.

```

package canprecede.model;

```

```

import java.util.ArrayList;

public class CanFollowPrecedeTree {
    private ArrayList<CanFollowPrecedeNode> nodes = new ArrayList<>();

    public ArrayList<CanFollowPrecedeNode> getNodes() {
        return nodes;
    }

    public void setNodes(ArrayList<CanFollowPrecedeNode> nodes) {
        this.nodes = nodes;
    }

    public void setNode(CanFollowPrecedeNode node) {
        this.nodes.add(node);
    }

    @Override
    public String toString() {
        return "CanPrecedeTree{" +
            "nodes=" + nodes +
            '}';
    }
}

```

Listing 9: Java class for CanFollow/CanPrecede trees

The Java class depicted in Listing 9 is also used for storing Template Attack Trees. Such trees are pre-defined and stored in the form of a repository (list of Java classes). After the identification of the potential attacks of a target system, during the *Identification of potential attacks* process, the created list of attacks is compared with the leaves of all the available Template Attack Trees to end up with a subset of them called “matching Template Attack Trees”. Listing 10 includes the Java method that does exactly that. The “allTemplateAttackTrees” list includes all the predefined Template Attack Trees. The “checkLeavesFromCanPrecedeNode” method is called for each of them, characterizing it as “potential attack tree for the target system” or “NOT potential attack tree for the target system”.

```

// method to return all the matched template trees
public CanPrecedeTree getMatchingTemplateTrees () {
    CanPrecedeTree matchingTrees = new CanPrecedeTree2();

    System.out.println("Find matching Attack-tree templates based on identi-
fied CAPECs...");

    // check all the available template attack trees
    int matchedTemplates = 0;
    for (int i = 0; i < allTemplateAttackTrees.getNodes().size(); i++) {
        CanPrecedeNode currentTemplateTree = allTemplateAttack-

```



```

Trees.getNodes().get(i);
    if (checkLeavesFromCanPrecedeNode(currentTemplateTree)) {
        matchingTrees.setNode(currentTemplateTree);
        System.out.println("Attack tree with root \"" + currentTemplateTree.getData() + "\" is a potential attack tree for the target system.");
        matchedTemplates++;
    } else {
        System.out.println("Attack tree with root \"" + currentTemplateTree.getData() + "\" is NOT a potential attack tree for the target system.");
    }
}

return matchingTrees;
}

```

Listing 10: Java method for identifying “matching Template Attack Trees”

```

// method to check if a given Tree (=CanPrecedeNode2) is a match or not,
// based on the identified CAPECs
public boolean checkLeavesFromCanPrecedeNode(CanPrecedeNode node) {

    boolean matchFlag = true;
    // if the node has children
    if (node.getChildren().size() > 0) {

        // if the node is gate
        if (node.getNodeType().equals(CanPrecedeNode.Type.GATE)) {
            // check if each child matches or not (childrenMatches list)
            ArrayList<Boolean> childrenMatches = new ArrayList<>();
            for (int i = 0; i < node.getChildren().size(); i++) {
                CanPrecedeNode currentChild = node.getChildren().get(i);
                childrenMatches.add(checkLeavesFromCanPrecedeNode(currentChild));
            }
            // OR gate
            if (node.getData().equals("OR")) {
                matchFlag = false;
                for (int i = 0; i < childrenMatches.size(); i++) {
                    if (childrenMatches.get(i).booleanValue() == true) {
                        matchFlag = true;
                        break;
                    }
                }
            }
            // AND gate
            if (node.getData().equals("AND")) {
                matchFlag = true;
                for (int i = 0; i < childrenMatches.size(); i++) {

```

```

        if (childrenMatches.get(i).booleanValue() == false) {
            matchFlag = false;
            break;
        }
    }
}

// if the node is not a gate
if (!node.getNodeType().equals(CanPrecedeNode.Type.GATE)) {
    matchFlag = checkLeavesFromCanPrecede-
Node(node.getChildren().get(0));
}
return matchFlag;
} else {
    matchFlag = checkCapec(node.getData());
    return matchFlag;
}
}
}

```

Listing 11: Java method for matching a given tree with a set of potential attacks

The way the Java class for the Templates Attack Trees is implemented, allows for recursive parsing as the “checkLeavesFromCanPrecedeNode” method in Listing 11 depicts. The method calls itself for every child of a given tree node. The “childrenMatches” method is an auxiliary one that is responsible for the actual checking.

Along with the examples of Template Attack Trees of subsection 3.1.5, we present here yet another example that incorporates both cyber and physical vulnerabilities. As it can be seen, Figure 14 depicts a tree that is a combination of the trees presented earlier. The “Publish tampered messages” and “Lidar” Template Attack Trees are present. A similar with the “Lidar” tree is also incorporated but this time for the camera sensor.

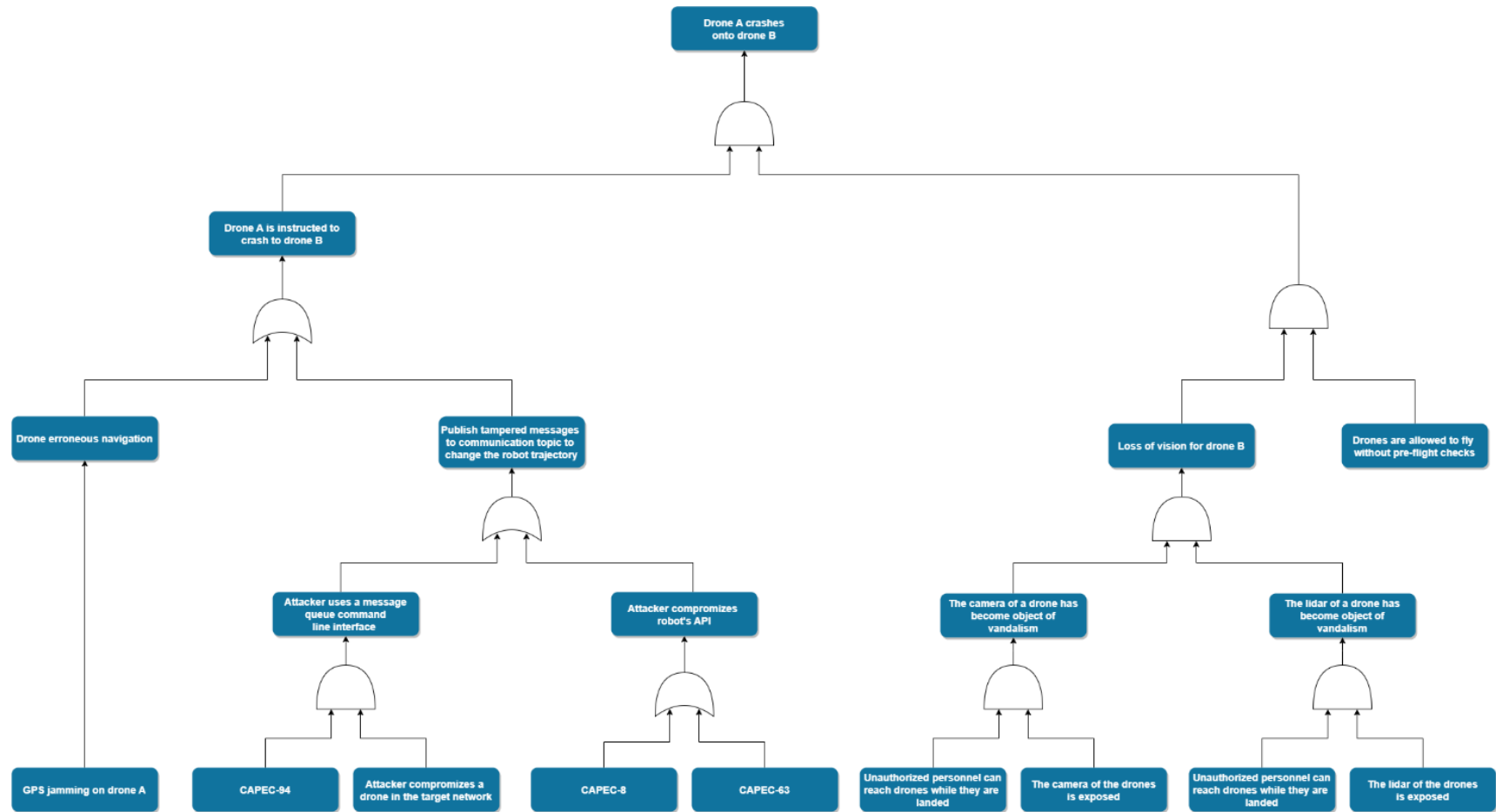


Figure 14: Template Attack Tree with cyber and physical vulnerabilities

The ultimate goal of the tree is one drone to be instructed to crash on another drone. The way for the attacker to accomplish their goal is drone A to be instructed to crash to drone B and at the same time drone B to suffer from loss of vision. The first prerequisite is

achieved either by a GPS jamming attack that causes erroneous navigation to the drone or by publishing tampered messages to the corresponding navigation topic of a message queue. The second prerequisite is necessary since drone may use their camera and appropriate software to avoid collision with obstacles. Since there might be two sensors (camera and lidar) that can be used by a drone for that purpose, both of them should be disabled, and no pre-flight checks should notice the vandalism.

We have already mention that security and safety are closely linked. This tree is another opportunity to do so.

In a combined safety/security tree (attack/fault tree), non-malicious causes of non-functional system components can be also incorporated. So, we could e.g. have a mixture of security attacks (e.g. tampered messages or GPS jamming) and safety faults (e.g. buildup of soot on the camera/lidar lenses). Such a tree could then form one constituent part of the whole.

4.1.5 Generation of security EDDIs

The set of “potential attack trees for the target system”, output of the *Generation of attack trees* process, incorporates all the possible attack scenarios that could take place, based on the vulnerabilities (cyber and physical) of the target system. The high-level information that is gathered for each attack can be seen in Listing 12, including “capecId”, “title”, “capecDescription”, “severity”, “likelihood”, “mitigation”, “relatedCapecs”, “relatedCwes”, and “relatedCves”.

```
{
  "capecId": "70",
  "title": "Try Common or Default Usernames and Passwords",
  "capecDescription": "An adversary may try certain common or default
  usernames and
  passwords to gain access into the system and perform unauthorized ac-
  tions. An adversary
  may try an intelligent brute force using empty passwords, known vendor
  default credentials,
  as well as a dictionary of common usernames and passwords. Many vendor
  products come
  preconfigured with default (and thus well-known) usernames and passwords
  that should be
  deleted prior to usage in a production environment. It is a common mis-
  take to forget to
  remove these default login credentials. Another problem is that users
  would pick very
  simple (common) passwords (e.g. \"secret\" or \"password\") that make it
  easier for the
  attacker to gain access to the system compared to using a brute force at-
  tack or even a
  dictionary attack using a full dictionary.",
  "severity": "High",
  "type": "Detailed",
  "likelihood": "Medium",
```

```

"mitigation": "[Delete all default account credentials that may be put in
by the product
vendor., Implement a password throttling mechanism. This mechanism should
take into account
both the IP address and the log in name of the user., Put together a
strong password policy
and make sure that all user created passwords comply with it. Alterna-
tively automatically
generate strong passwords for users., Passwords need to be recycled to
prevent aging, that
is every once in a while a new password must be chosen.]",
"relatedCapecs": [
{
"nature": "ChildOf",
"CAPECID": "49",
"selfClosing": "true"
},
{
"nature": "CanPrecede",
"CAPECID": "600",
"selfClosing": "true"
},
...
],
"relatedCwes": [
{
"cWEID": "521",
"selfClosing": "true"
},
...
],
"relatedCves": [
{
"cveId": "CVE-2019-5021",
"vulnerabilityDescription": "Versions of the Official Alpine Linux
Docker images
(since v3.3) contain a NULL password for the `root` user. This vul-
nerability
appears to be the result of a regression introduced in December of
2015. Due to
the nature of this issue, systems deployed using affected versions of
the Alpine
Linux container which utilize Linux PAM, or some other mechanism
which uses the
system shadow file as an authentication database, may accept a NULL
password for
the `root` user.",

```

```
"cvssScore": 9.8,  
"cvssVector": "CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H",  
"vulnerableAsset": [  
  {  
    "cpeId": "cpe:2.3:a:gliderlabs:docker-alpine:3.3:*:*:*:*:*:*:*",  
    "id": 0  
  },  
  ...  
],  
"relatedCapecs": [  
  "70",  
  "191"  
],  
"id": 0  
},  
...  
],  
"relatedCapecs": [  
  "70",  
  "191"  
],  
"id": 0  
}  
],  
"id": 0  
}
```

Listing 12: Snippet of code depicting the information that is gathered for every identified potential attack of the target system

This information is then translated into the security EDDIs. The security EDDI consists of a set of such Python scripts, one for each identified potential attack tree. Auxiliary applications are an MQTT broker and an Intrusion Detection System (IDS). The Python scripts hold the logic for discovering the ultimate goal of an attacker based on the information of each potential attack tree and the alerts that are created from the running IDS.

The IDS filters the network traffic of the system for malicious or suspicious packets and creates an alert every time it detects one. These alerts are then published in an MQTT topic. Every Python script listens to that topic, waiting for alerts of a conducted attack. As soon as such an alert is detected, the logic in the Python script parses the tree based on the parent/child relationships of the attacks, trying to identify the ultimate goal of the attacker. The conducted attack itself is checked against the leaves of the tree. If there is a match, the checking continues to the upper level of nodes, until the root is reached. If the parsing of the tree reaches the root, the ultimate goal of the attacker can be achieved based on the on-going cyber-attacks, recognised by the IDS, and present physical attacks. It should be mentioned that IDS is not able to detect physical attacks. In their majority, physical attacks are identified during design time. Sensors that could detect physical attacks are out of the expertise of the authors of this document. However, such a sensor of physical attacks could easily be incorporated in the security EDDI solution

due to the distributed nature of the latter. The only thing that is needed is a MQTT topic that such a sensor would publish its alerts to and the corresponding Python scripts to listen to it.

```
def connect_mqtt():
    def on_connect(client, userdata, flags, rc):
        if rc == 0:
            print("Connected to MQTT Broker!")
        else:
            print("Failed to connect, return code %d\n", rc)
    # Set Connecting Client ID
    client = mqtt_client.Client(client_id)
    #client.username_pw_set(username, password)
    client.on_connect = on_connect
    client.connect(broker, port)
    return client

...

def subscribe(client: mqtt_client):
    def on_message(client, userdata, msg):
        # convert json data to dictionary
        print(f"Received `{msg.payload.decode()}` from `{msg.topic}` topic")
        message_dict = json.loads(msg.payload)
        updateAllVariables(message_dict)
    client.subscribe(topic)
    client.subscribe(topic2)
    client.on_message = on_message

...

def checkFaultTree():
    print("checking fault tree...")
    global messageQueueCLI
    global compromisedRobotAPI
    global publishTamperedMessages

    # start of first layer
    if capec94.enabled and droneCompromised:
        print("Attacker uses a message queue cli interface")
        messageQueueCLI = True

    if capec8.enabled or capec63.enabled:
        print("Attacker compromises robot API")
        compromisedRobotAPI = True

    # end of first layer
    # Goal
    if messageQueueCLI or compromisedRobotAPI:
```

```
print("Publish tampered messages to communication "  
      "topic to change the robot trajectory")  
publishTamperedMessages = True
```

Listing 13: Python script for a specific “publish tampered messages” attack tree, part of the security EDDI

Listing 13 depicts snippets from such a Python script, part of a security EDDI. The script corresponds to the attack tree with ultimate goal “publish tampered messages” shown in Figure 4. The “connect_mqtt()” function connects the script to the MQTT broker. The “subscribe” function makes the subscription to the topic where the IDS alerts are published. Finally, the “checkFaultTree” function includes the logic for parsing the attack tree.

4.1.6 Runtime security- Intrusion Detection System

As it is mentioned already, the produced information from the security assessment process is incorporated in the security EDDI, transferring said information to the runtime to be used for the mitigation of threats. The prerequisite, in this case is the monitoring of the security events that need to take place also during runtime. The tool that is responsible for monitoring the network incoming malicious packets is an IDS.

An IDS is a monitoring system designed to detect suspicious activities in the communication between a system and its external entities. When such activities are identified, the IDS generates alerts to notify a system administrator or incident responder for further investigation and implementation of mitigation measures. IDSs can be categorized into three main types: host-based, network-based, and application-based. Host-based IDSs are installed directly on individual hosts and monitor both incoming and outgoing network traffic. These IDSs are typically deployed on systems that handle sensitive data, cannot easily receive patches, or require additional security measures. Network-based IDSs, on the other hand, are positioned at strategic points within a network, such as the gateway, and analyse the traffic exchanged between different network devices. They act as filters to identify potential intrusions. Application-based IDSs focus on monitoring specific application protocols, such as the SQL protocol, in order to detect and respond to intrusions that target those protocols.

The tool that was chosen for the monitoring of the incoming packets is a signature-based IDS, which detects suspicious packets based on specific patterns in their headers or body. More specifically, Snort is the tool that will be used due to its wide adoption.

Snort is a well-known open-source IDS that has gained popularity and has been extensively studied in the literature. It employs a rule-based approach to define malicious network activity, triggering alerts when a rule is matched. Developed by Sourcefire since 1998, Snort utilizes a single-threaded architecture and relies on the TCP/IP stack to capture and inspect network packets, including their headers and bodies. It offers the flexibility to be configured as a comprehensive network Intrusion Prevention System (IPS) that not only monitors network activity but also detects and blocks potential attack vectors. When satisfied rules are triggered, alerts are generated and logged, enabling the creation of reports based on these alerts. Snort is particularly suitable for fulfilling lightweight IDS requirements.

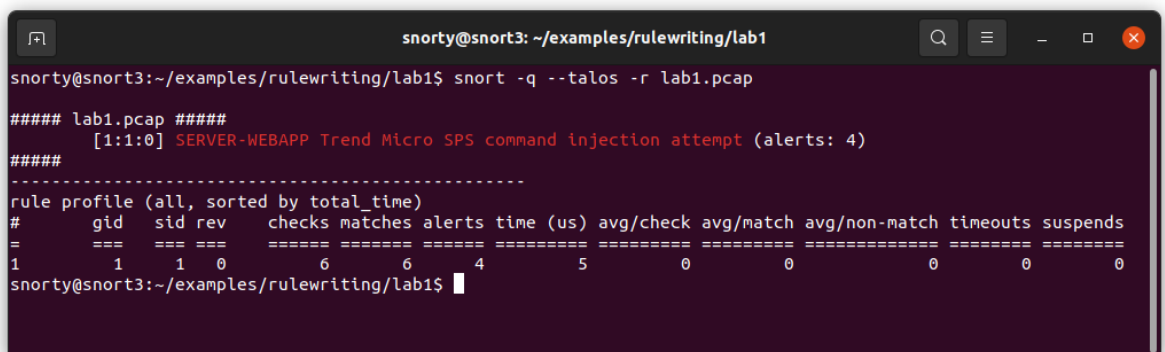
According to the CVE-2016-6267 vulnerability (NVD), Trend Micro Smart Protection Server allows remote authenticated users to execute arbitrary commands. To illustrate the functionality of Snort and provide an example output, we will utilize the Snort rule presented in Figure 15. This rule is designed to identify packets originating from a user attempting to exploit the aforementioned vulnerability and carry out an attack. The rule is triggered when the string "/admin_notification.php" is detected in the http_uri path (lines 4-5) and the string "spare_Community=" is present in the http_client_body (lines 6-7). The message to be printed when the alert is generated is specified in line 2, while line 3 denotes the direction of the packets that could trigger this rule. Line 9 includes a reference to the associated vulnerability, and line 11 assigns a unique identifier to this rule.

```

1 alert http (
2     msg:"SERVER-WEBAPP Trend Micro SPS command injection attempt";
3     flow:to_server,established;
4     http_uri:path;
5     content:"/admin_notification.php",nocase;
6     http_client_body;
7     content:"spare_Community=",nocase;
8     pcre:"/(^|&)spare_Community=[^&]*?([\x60\x3b\x7c]|[\x3c\x3e\x24]\x28)/i";
9     reference:cve,2016-6267;
10    classtype:web-application-attack;
11    sid:1;
12 )
    
```

Figure 15: Snort - example rule

Using the Lab1 pcap file, available with the installation of Snort, we were able to send traffic to Snort with packets that can trigger the above rule. The rule was matched against four packets included in the pcap file (alerts: 4). An alert was created for each match, including the message “SERVER – WEBAPP Trend Micro SPS command injection attempt” (Figure 16).



```

snort@snort3: ~/examples/rulewriting/lab1$ snort -q --talos -r lab1.pcap

##### lab1.pcap #####
[1:1:0] SERVER-WEBAPP Trend Micro SPS command injection attempt (alerts: 4)
#####

-----
rule profile (all, sorted by total_time)
#   gid  sid  rev  checks matches alerts time (us) avg/check avg/match avg/non-match timeouts suspends
=   ==  ==  ==  =====
1   1    1    0    6         6         4         5         0         0         0         0         0
snort@snort3:~/examples/rulewriting/lab1$
    
```

Figure 16: Snort example output

5. APPLYING SESAME METHODOLOGY

In this section, we will present how the SESAME security assessment can be applied to the project use cases. ROS is present in all three use cases that SESAME security assessment is being evaluated for: unmanned aerial vehicles (UAVs) for fighting fungal diseases in vineyards, inspection of a power station using UAVs in two operation modes (normal and emergency), and safe and secure deployment of a fleet of AMR with task exchangeability. Based on that common ground we conducted a security assessment according to the proposed methodology and its individual processes.

Using the CVE-search open-source tool and the RVD custom parser tool we requested both CVE and RVD for known vulnerabilities. The outcome of the requests was 11 vulnerabilities, presented in Table 2.

Table 2: Identified ROS-related vulnerabilities in CVE and NVD repositories

CVE-IDs	CWE-IDs	CWE Name
CVE-2016-10681	CWE-300	Channel Accessible by Non-Endpoint
CVE-2016-10681	CWE-310	Cryptographic Issues
CVE-2019-13445	CWE-190	Integer Overflow or Wraparound
CVE-2019-13465	CWE-noinfo	NA
CVE-2019-13566	CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
CVE-2019-19625	CWE-200	Exposure of Sensitive Information to an Unauthorized Actor
CVE-2019-19627	CWE-200	Exposure of Sensitive Information to an Unauthorized Actor
CVE-2020-10271	CWE-668	Exposure of Resource to Wrong Sphere
CVE-2020-10272	CWE-306	Missing Authentication for Critical Function
CVE-2020-10289	CWE-20	Improper Input Validation
CVE-2020-16124	CWE-190	Integer Overflow or Wraparound

As it can be seen, the CVE-IDs along with the corresponding CWE-IDs and the CWE names are presented. The CVE-IDs represent identified vulnerabilities, which are associated with specific weaknesses (CWE-IDs) such as buffer overflow, missing authentication, exposure of sensitive information, and more. Each weakness is connected to particular attack patterns that adversaries can exploit when leveraging the identified weakness. Consequently, the defined weaknesses give rise to a range of potential attacks, as outlined in Table 3.

Table 3: Identified ROS-related attack patterns in CAPEC repository

CWE-IDs	CAPEC-IDs	CAPEC Name
CWE-300	CAPEC-466	Leveraging Active Adversary in the Middle Attacks to Bypass Same Origin Policy
	CAPEC-57	Utilizing REST's Trust in the System Resource to Obtain Sensitive Data
	CAPEC-589	DNS Blocking

CWE-IDs	CAPEC-IDs	CAPEC Name
	CAPEC-590	IP Address Blocking
	CAPEC-612	WiFi MAC Address Tracking
	CAPEC-613	WiFi SSID Tracking
	CAPEC-615	Evil Twin Wi-Fi Attack
	CAPEC-662	Adversary in the Browser (AiTB)
	CAPEC-94	Adversary in the Middle (AiTM)
CWE-310	-	
CWE-190	CAPEC-92	Forced Integer Overflow
CWE-120	CAPEC-10	Buffer Overflow via Environment Variables
	CAPEC-100	Overflow Buffers
	CAPEC-14	Client-side Injection-induced Buffer Overflow
	CAPEC-24	Filter Failure through Buffer Overflow
	CAPEC-42	MIME Conversion
	CAPEC-44	Overflow Binary Resource File
	CAPEC-45	Buffer Overflow via Symbolic Links
	CAPEC-46	Overflow Variables and Tags
	CAPEC-47	Buffer Overflow via Parameter Expansion
	CAPEC-67	String Format Overflow in syslog()
	CAPEC-8	Buffer Overflow in an API Call
	CAPEC-9	Buffer Overflow in Local Command-Line Utilities
	CAPEC-92	Forced Integer Over
CWE-200	CAPEC-116	Excavation
	CAPEC-13	Subverting Environment Variable Values
	CAPEC-169	Footprinting
	CAPEC-22	Exploiting Trust in Client
	CAPEC-224	Fingerprinting
	CAPEC-285	ICMP Echo Request Ping
	CAPEC-287	TCP SYN Scan
	CAPEC-290	Enumerate Mail Exchange (MX) Records
	CAPEC-291	DNS Zone Transfers
	CAPEC-292	Host Discovery
	CAPEC-293	Traceroute Route Enumeration
	CAPEC-294	ICMP Address Mask Request
	CAPEC-295	Timestamp Request
	CAPEC-296	ICMP Information Request
	CAPEC-297	TCP ACK Ping
	CAPEC-298	UDP Ping
	CAPEC-299	TCP SYN Ping
	CAPEC-300	Port Scanning
	CAPEC-301	TCP Connect Scan
	CAPEC-302	TCP FIN Scan
	CAPEC-303	TCP Xmas Scan
	CAPEC-304	TCP Null Scan
	CAPEC-305	TCP ACK Scan
	CAPEC-306	TCP Window Scan

CWE-IDs	CAPEC-IDs	CAPEC Name
	CAPEC-307	TCP RPC Scan
	CAPEC-308	UDP Scan
	CAPEC-309	Network Topology Mapping
	CAPEC-310	Scanning for Vulnerable Software
	CAPEC-312	Active OS Fingerprinting
	CAPEC-313	Passive OS Fingerprinting
	CAPEC-317	IP ID Sequencing Probe
	CAPEC-318	IP 'ID' Echoed Byte-Order Probe
	CAPEC-319	IP (DF) 'Don't Fragment Bit' Echoing Probe
	CAPEC-320	TCP Timestamp Probe
	CAPEC-321	TCP Sequence Number Probe
	CAPEC-322	TCP (ISN) Greatest Common Divisor Probe
	CAPEC-323	TCP (ISN) Counter Rate Probe
	CAPEC-324	TCP (ISN) Sequence Predictability Probe
	CAPEC-325	TCP Congestion Control Flag (ECN) Probe
	CAPEC-326	TCP Initial Window Size Probe
	CAPEC-327	TCP Options Probe
	CAPEC-328	TCP 'RST' Flag Checksum Probe
	CAPEC-329	ICMP Error Message Quoting Probe
	CAPEC-330	ICMP Error Message Echoing Integrity Probe
	CAPEC-472	Browser Fingerprinting
	CAPEC-497	File Discovery
	CAPEC-508	Shoulder Surfing
	CAPEC-573	Process Footprinting
	CAPEC-574	Services Footprinting
	CAPEC-575	Account Footprinting
	CAPEC-576	Group Permission Footprinting
	CAPEC-577	Owner Footprinting
	CAPEC-59	Session Credential Falsification through Prediction
	CAPEC-60	Reusing Session IDs (aka Session Replay)
	CAPEC-616	Establish Rogue Location
	CAPEC-643	Identify Shared Files/Directories on System
	CAPEC-646	Peripheral Footprinting
	CAPEC-651	Eavesdropping
	CAPEC-79	Using Slashes in Alternate Encoding
CWE-668	-	
CWE-306	CAPEC-12	Choosing Message Identifier
	CAPEC-166	Force the System to Reset Values
	CAPEC-36	Using Unpublished Interfaces
	CAPEC-62	Cross Site Request Forgery
CWE-20	CAPEC-10	Buffer Overflow via Environment Variables
	CAPEC-101	Server Side Include (SSI) Injection
	CAPEC-104	Cross Zone Scripting
	CAPEC-108	Command Line Execution through SQL Injection
	CAPEC-109	Object Relational Mapping Injection

CWE-IDs	CAPEC-IDs	CAPEC Name
	CAPEC-110	SQL Injection through SOAP Parameter Tampering
	CAPEC-120	Double Encoding
	CAPEC-13	Subverting Environment Variable Values
	CAPEC-135	Format String Injection
	CAPEC-136	LDAP Injection
	CAPEC-14	Client-side Injection-induced Buffer Overflow
	CAPEC-153	Input Data Manipulation
	CAPEC-182	Flash Injection
	CAPEC-209	XSS Using MIME Type Mismatch
	CAPEC-22	Exploiting Trust in Client
	CAPEC-23	File Content Injection
	CAPEC-230	XML Nested Payloads
	CAPEC-231	Oversized Serialized Data Payloads
	CAPEC-24	Filter Failure through Buffer Overflow
	CAPEC-250	XML Injection
	CAPEC-261	Fuzzing for garnering other adjacent user/sensitive data
	CAPEC-267	Leverage Alternate Encoding
	CAPEC-28	Fuzzing
	CAPEC-3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters
	CAPEC-31	Accessing/Intercepting/Modifying HTTP Cookies
	CAPEC-42	MIME Conversion
	CAPEC-43	Exploiting Multiple Input Interpretation Layers
	CAPEC-45	Buffer Overflow via Symbolic Links
	CAPEC-46	Overflow Variables and Tags
	CAPEC-47	Buffer Overflow via Parameter Expansion
	CAPEC-473	Signature Spoof
	CAPEC-52	Embedding NULL Bytes
	CAPEC-53	Postfix, Null Terminate, and Backslash
	CAPEC-588	DOM-Based XSS
	CAPEC-63	Cross-Site Scripting (XSS)
	CAPEC-64	Using Slashes and URL Encoding Combined to Bypass Validation Logic
	CAPEC-664	Server Side Request Forgery
	CAPEC-67	String Format Overflow in syslog()
	CAPEC-7	Blind SQL Injection
	CAPEC-71	Using Unicode Encoding to Bypass Validation Logic
	CAPEC-72	URL Encoding
	CAPEC-73	User-Controlled Filename
	CAPEC-78	Using Escaped Slashes in Alternate Encoding
	CAPEC-79	Using Slashes in Alternate Encoding
	CAPEC-8	Buffer Overflow in an API Call
	CAPEC-80	Using UTF-8 Encoding to Bypass Validation Logic
	CAPEC-81	Web Logs Tampering
	CAPEC-83	XPath Injection

CWE-IDs	CAPEC-IDs	CAPEC Name
	CAPEC-85	AJAX Footprinting
	CAPEC-88	OS Command Injection
	CAPEC-9	Buffer Overflow in Local Command-Line Utilities

The first column of the table includes the IDs of the weaknesses mentioned in Table 2. The remaining two columns in the table present the IDs and names of the attack patterns associated with each weakness. These attack patterns encompass a range of attacks that could potentially be executed against our system. The level of detail provided for each attack varies, with some being more abstract and others offering a more granular description. Detailed attack patterns are particularly valuable for our process, especially when identifying mitigations. Although each of the attacks included in the table above is a threat for our system on its own, some of them can be combined and create more complex attacks. Using the CanFollow and CanPrecede relationships between the attack patterns four small trees are created, presented in Figure 17.

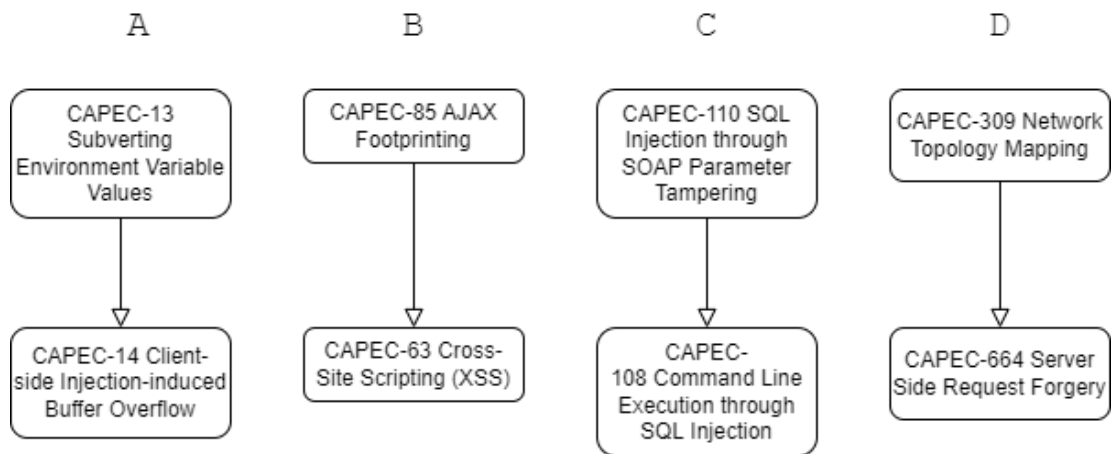


Figure 17: Combined attack patterns based on the CanFollow and CanPrecede relationships

These trees group together attacks that can be conducted in sequence. One attack can create the necessary prerequisites for another attack to happen.

The process of constructing the Template Attack Trees can start as soon as the identification of potential attacks is concluded. During this process, security experts create attack trees, which include a subset of the identified potential attacks. This can be done utilizing knowledge published in the security related literature or available from real-life conducted attacks incidents of the past. Such a Template Attack Tree that includes CAPECs 8, 63, and 94 has already been depicted in Figure 4.

An updated version of it is shown in Figure 18. This version includes one of the CanPrecede trees that were created according to the methodology described in 3.1.6. As we saw in Figure 17, CAPEC-85 AJAX Footprinting is an attack that can open the way for another attack with id CAPEC-63 Cross-Site Scripting (XSS). This exact information is added in the updated version of the "publish tempered messages" Template Attack Tree, as the leaf at the very right side depicts.

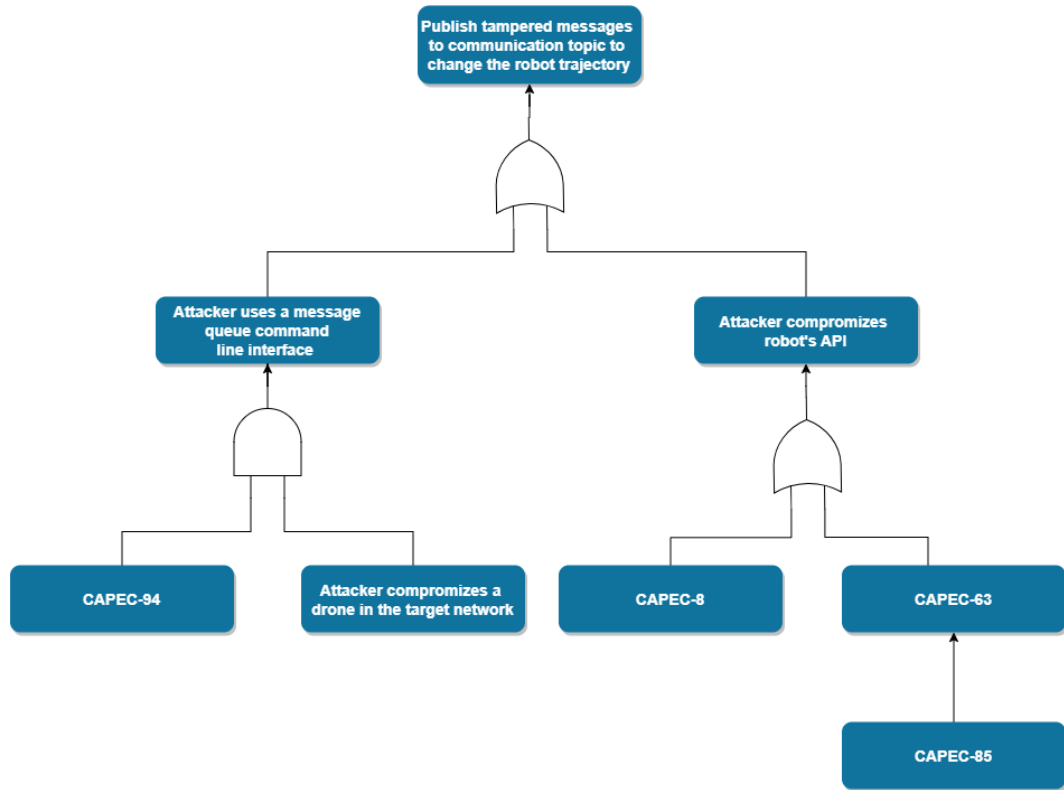


Figure 18: Updated version of the "publish tempered messages" Template Attack Tree

The next in line process of the proposed methodology is the creation of the security EDDI. In our example, since a Template Attack Tree from the 'Template Attack Tree' repository is considered a tree that describes an attack that could be conducted against the target system, the next step is to create a Python script related to that tree.

```

class commonAttack:
    def __init__(self, capecID):
        self.name = capecID
        self.enabled = False

droneCompromised = False
messageQueueCLI = False
compromisedRobotAPI = False
publishTamperedMessages = False
capec8 = commonAttack(8)
capec63 = commonAttack(63)
capec94 = commonAttack(94)

listOfAttacks = [capec8, capec63, capec94, droneCompromised]

from paho.mqtt import client as mqtt_client
import json
broker = '127.0.0.1'
port = 1883
topic2 = "snort"
    
```

```

client_id = f'python-mqtt-eddi'

def connect_mqtt():
    def on_connect(client, userdata, flags, rc):
        if rc == 0:
            print("Connected to MQTT Broker!")
        else:
            print("Failed to connect, return code %d\n", rc)
    # Set Connecting Client ID
    client = mqtt_client.Client(client_id)
    #client.username_pw_set(username, password)
    client.on_connect = on_connect
    client.connect(broker, port)
    return client

def subscribe(client: mqtt_client):
    def on_message(client, userdata, msg):
        # convert json data to dictionary
        print(f"Received `{msg.payload.decode()}` from `{msg.topic}` topic")
        message_dict = json.loads(msg.payload)
        updateAllVariables(message_dict)
    client.subscribe(topic2)
    client.on_message = on_message

def checkFaultTree():
    print("checking fault tree...")
    global messageQueueCLI
    global compromisedRobotAPI
    global publishTamperedMessages

    # start of first layer
    if capec94.enabled and droneCompromised:
        print("Attacker uses a message queue cli interface")
        messageQueueCLI = True

    if capec8.enabled or capec63.enabled:
        print("Attacker compromises robot API")
        compromisedRobotAPI = True
    # end of first layer
    # Goal
    if messageQueueCLI or compromisedRobotAPI:
        print("Publish tampered messages to communication "
              "topic to change the robot trajectory")
        finalMsg = ""
        publishTamperedMessages = True
        print(listOfAttacks)
        sendToSafetyEDDI()

```



```
def sendToSafetyEDDI():
    print("Sending to SafetyEDDI")
def run():
    client = connect_mqtt()
    subscribe(client)
    client.loop_forever()

# Press the green button in the gutter to run the script.
if __name__ == '__main__':
    run()
```

Listing 14: Python script (part of security EDDI) for the "publish tampered messages" Template Attack Tree

Listing 14 includes the code of the Python script correlated with the "publish tampered messages" Template Attack Tree. The “connect_mqtt()”, “subscribe”, and “checkFaultTree” functions that we saw in 4.1.5 are also included. The presented script subscribes to topic “snort”, since this is the topic that the running IDS publishes its alerts.

This section showed a very simple example of how the SESAME security assessment can be conducted. The set of vulnerabilities that we used as a starting point was small (11 in total). However, even with that small set of vulnerabilities we end up with a significantly larger set of potential attacks. Table 4 shows the identified attacks of the UAVs for fighting fungal diseases in vineyards use case, based on a system description that was provided. The first and the forth columns include known vulnerabilities of system components. Columns 2 and 5 present the related weaknesses, while the third and the sixth columns mention the attacks that are related to the given weaknesses.

Table 4: UAVs for fighting fungal diseases in vineyards use case identified attacks (Aero41)

CVE-IDs	CWE-IDs	CAPEC-IDs	CVE-IDs	CWE-IDs	CAPEC-IDs
CVE-2015-20107	CWE-77	CAPEC-40	CVE-2021-3162	CWE-295	CAPEC-459
		CAPEC-136			CAPEC-475
		CAPEC-248	CVE-2016-9962	CWE-362	CAPEC-26
		CAPEC-43			CAPEC-29
		CAPEC-15	CVE-2019-14271	CWE-665	CAPEC-26
		CAPEC-183			CAPEC-29
		CAPEC-76	CVE-2021-44719	CWE-552	CAPEC-150
CVE-2022-45061	CWE-400	CAPEC-147			CAPEC-639
		CAPEC-492	CVE-2020-15360	CWE-862	CAPEC-665
CVE-2019-5736	CWE-78	CAPEC-6	CVE-2017-11468	CWE-770	CAPEC-230
		CAPEC-43			CAPEC-493
		CAPEC-88			CAPEC-528
		CAPEC-108			CAPEC-489
		CAPEC-15			CAPEC-125
CVE-2014-8179	CWE-20	CAPEC-182			CAPEC-147
		CAPEC-230			CAPEC-488

		CAPEC-28			CAPEC-496
		CAPEC-3			CAPEC-487
		CAPEC-42			CAPEC-130
		CAPEC-664			CAPEC-491
		CAPEC-67			CAPEC-229
		CAPEC-78			CAPEC-495
		CAPEC-13			CAPEC-197
		CAPEC-135			CAPEC-231
		CAPEC-14			CAPEC-469
		CAPEC-153			CAPEC-486
		CAPEC-262			CAPEC-490
		CAPEC-45			CAPEC-482
		CAPEC-72			CAPEC-494
		CAPEC-83	CVE-2021-26461	CWE-190	CAPEC-92
		CAPEC-109	CVE-2020-10281	CWE-319	CAPEC-383
		CAPEC-110			CAPEC-102
		CAPEC-120			CAPEC-117
		CAPEC-136			CAPEC-477
		CAPEC-22			CAPEC-65
		CAPEC-24	CVE-2020-10282	CWE-306	CAPEC-166
		CAPEC-250			CAPEC-12
		CAPEC-52			CAPEC-36
		CAPEC-71			CAPEC-62
		CAPEC-79	CVE-2020-28436	CWE-77	CAPEC-40
		CAPEC-73			CAPEC-136
		CAPEC-81			CAPEC-248
		CAPEC-85			CAPEC-75
		CAPEC-64			CAPEC-43
		CAPEC-7			CAPEC-15
		CAPEC-8			CAPEC-183
		CAPEC-31			CAPEC-76
		CAPEC-43	CVE-2022-38216	CWE-190	CAPEC-92
		CAPEC-588	CVE-2021-3749	CWE-1333	CAPEC-492
		CAPEC-80	CVE-2021-3749	CWE-400	CAPEC-147
		CAPEC-88			CAPEC-227
		CAPEC-10			CAPEC-492
		CAPEC-101	CVE-2021-3757	CWE-1321	CAPEC-77
		CAPEC-104			CAPEC-1
		CAPEC-108			CAPEC-180
		CAPEC-209	CVE-2022-31129	CWE-400	CAPEC-492
		CAPEC-267			CAPEC-147
		CAPEC-473			
		CAPEC-23			
		CAPEC-231			

		CAPEC-46			
		CAPEC-63			
		CAPEC-9			
		CAPEC-47			
		CAPEC-53			
CVE-2021-21284	CWE-22	CAPEC-78			
		CAPEC-79			
		CAPEC-64			
		CAPEC-126			
		CAPEC-76			
CVE-2014-6407	CWE-59	CAPEC-132			
		CAPEC-17			
		CAPEC-76			
CVE-2019-13509	CWE-532	CAPEC-219			
CVE-2018-15514	CWE-502	CAPEC-586			
CVE-2019-15752	CWE-732	CAPEC-127			
		CAPEC-17			
		CAPEC-180			
		CAPEC-206			
		CAPEC-60			
		CAPEC-61			
		CAPEC-1			
		CAPEC-122			
		CAPEC-234			
		CAPEC-62			
		CAPEC-642			

Identifying all the known vulnerabilities related to the three use cases that SESAME security assessment will be integrated is a very demanding task and is going to be finalized the last months of the project. Defining the whole set of known vulnerabilities of each use case will reveal the corresponding attacks and the possible mitigation actions.

6. CONCLUSIONS

This deliverable presents the reader with a short description of the challenge of conducting security assessments for robotic systems. The unique aspect of modern robotic systems is that they operate in an environment that is connected to the external world, interacting with various systems, devices, and services of uncertain trustworthiness. Additionally, these robotic systems often operate in close proximity to humans and engage in human-machine interactions. This environment differs significantly from the traditional industrial robot setting, which was closed and trustworthy.

What follows is the introduction to the SESAME security assessment methodology. Each of the processes of the methodology are presented analysing the rationale behind

them. Section 4 focuses on the tools that were developed and used for materializing the proposed methodology. Both open-source tools and custom applications, developed by the authors, were used for that purpose.

Finally, an extensive example is provided, beginning with the identification of common vulnerabilities found in all three use cases integrated with the proposed methodology. The example then walks through each step of the methodology, showing the individual outcome of each process.

7. REFERENCES

- [1] Ruffin White, Dr Christensen, I Henrik, Dr Quigley, et al. SROS: Securing ROS over the wire, in the graph, and through the kernel. *arXiv preprint arXiv:1611.07060*, 2016.
- [2] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.
- [3] David D Mascarenas, Jarrod McClean, Christopher J Stull, and Charles R Farrar. A Preliminary Cyber-Physical Security Assessment of the Robot Operating System (ROS). Technical report, Los Alamos National Lab (LANL), Los Alamos, NM (United States), 2013.
- [4] Davide Quarta, Marcello Pogliani, Mario Polino, Federico Maggi, Andrea Maria Zanchettin, and Stefano Zanero. An Experimental Security Analysis of an Industrial Robot Controller. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 268–286. IEEE, 2017.
- [5] International Organization for Standardization (ISO). Robots and robotic devices—safety requirements for industrial robots—part 2: Robot systems and integration, 2011.
- [6] https://www.youtube.com/watch?v=yFi7UL70zTo&ab_channel=TUBerlin-IndustrielleAutomatisierungstechnik, (accessed December 18, 2021).
- [7] Siegfried Hollerer, Clara Fischer, Bernhard Brenner, Maximilian Papa, Sebastian Schlund, Wolfgang Kastner, Joachim Fabini, and Tanja Zseby. Cobot attack: a security assessment exemplified by a specific collaborative robot. *Procedia Manufacturing*, 54:191–196, 2021.
- [8] Gelei Deng, Yuan Zhou, Yuan Xu, Tianwei Zhang, and Yang Liu. An investigation of byzantine threats in multi-robot systems. In *24th International Symposium on Research in Attacks, Intrusions and Defenses*, pages 17–32, 2021.
- [9] Global Times. Mainframe malfunction causes dozens of drones to crash into building in SW China. <https://www.globaltimes.cn/page/202101/1214165.shtml>, 2021 (accessed December 18, 2021).
- [10] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al. Experimental security analysis of a modern automobile. In *2010 IEEE symposium on security and privacy*, pages 447–462. IEEE, 2010.
- [11] Gabriel Vasconcelos, Rodrigo Miani, Vitor Guizilini, and Jefferson Souza. Evaluation of DoS attacks on commercial Wi-Fi-based UAVs. *International Journal of Computer Network and Information Security*, 11:212, 04 2019.
- [12] Yuan Xu, Gelei Deng, Tianwei Zhang, Han Qiu, and Yungang Bao. Novel denial-of-service attacks against cloud-based multi-robot systems. *Information Sciences*, 576:329–344, 2021.
- [13] Alberto Giaretta, Michele De Donno, and Nicola Dragoni. Adding salt to pepper: A structured security assessment over a humanoid robot. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, pages 1–8, 2018.
- [14] Yosef Ashibani and Qusay H Mahmoud. Cyber physical systems security: Analysis, challenges and solutions. *Computers & Security*, 68:81–97, 2017.
- [15] MITRE. Common Vulnerabilities and Exposures. <https://cve.mitre.org/cve/>, (accessed December 18, 2021).
- [16] NIST. National vulnerability database. <https://nvd.nist.gov/>, (accessed December 18, 2021).
- [17] MITRE. Common Weakness Enumeration. <https://cwe.mitre.org/>, (accessed December 18, 2021).

- [18] MITRE. Common Attack Pattern Enumerations and Classifications. <https://capec.mitre.org/>, (accessed December 18, 2021).
- [19] Víctor Mayoral Vilches, Lander Usategui San Juan, Bernhard Dieber, Unai Ayucar Carbajo, and Endika Gil-Uriarte. Introducing the robot vulnerability database (rvd), 2021.
- [20] Kenta Kanakogi, Hironori Washizaki, Yoshiaki Fukazawa, Shinpei Ogata, Takao Okubo, Takehisa Kato, Hideyuki Kanuka, Atsuo Hazeyama, and Nobukazu Yoshioka. Tracing CAPEC attack patterns from CVE vulnerability information using natural language processing technique. In *Proceedings of the 54th Hawaii International Conference on System Sciences*, page 6996, 2021.
- [21] Jan Reich, Daniel Schneider, Ran Wei Rasmus Adler, Marc Zeller Tim Kelly, Ioannis Sorokos, Joe Guo, Georg Macher Christof Kaukewitsch, and Eric Armengaud. Digital Dependability Identities and the Open Dependability Exchange Meta-Model. https://deis-project.eu/fileadmin/user_upload/DEIS_D3.1_Specification_of_the_ODE_meta-model_and_documentation_of_the_fundamental_concept_of_DDI_PU.pdf, (accessed December 18, 2021).
- [22] Jackson Wynn. Threat assessment and remediation analysis (TARA). <https://www.mitre.org/sites/default/files/publications/pr-2359-threat-assessment-and-remediation-analysis.pdf>, 2014 (accessed December 18, 2021).
- [23] PILZ. White paper security. https://www.pilz.com/mam/pilz/content/uploads/wp_security_en_2018_10.pdf, 2018 (accessed December 18, 2021).
- [24] Gilbert Tang and Phil Webb. Human–robot shared workspace in aerospace factories. In *Human–Robot Interaction*, pages 72–79. Chapman and Hall/CRC, 2019.